Marlin

Table of contents:

Marlin

> An awesome project.

- Разворачивание Marlin
- Руководство администратора
- Руководство пользователя
- Термины и определения
- Command Line Interface `marlinctl`
- Marlin's gRPC API
- Marlin's Confluence

Разворачивание Marlin

1. Описание поставки и системные требования для запуска платформы Marlin
2. Конфигурация HashiCorp Vault
3. Конфигурация PAAS-кластеров Kubernetes или Openshift/OKD для работы с платформой Marlin
4. Конфигурация Keycloak для работы платформы Marlin
5. Установка и конфигурация Sonatype Nexus Repository
6. Конфигурация Bucket в MinIO
7. Установка и конфигурация Kubernetes External Secrets в PAAS-кластеры Kubernetes или Openshift/OKD
8. Конфигурация SonarQube
9. Конфигурация GitLab для работы с платформой Marlin
10. Установка и конфигурация Jenkins
11. Установка и конфигурация Портала Управления Marlin с помощью Helm-чарта

1. Описание поставки и системные требования для запуска платформы Marlin

Подготовка площадки для установки платформы Marlin

Для установки в минимальной конфигурации потребуется:

1. Персональный компьютер с которого будет производиться установка
    - ОС: Windows 10, Linux, Mac OS
    - Сетевые доступы до сети, в которую будет развернута платформа
    - Утилиты:
        - helm
        - helmwave
        - kubectl или oc
2. 3 кластера Kubernetes или Openshift/OKD для минимальной конфигурации:

    Master x 3 шт.

Параметр	Значение
CPU (vCPU)	4
RAM	16 ГБ
HDD	100 ГБ

    Worker x 3 шт.

Параметр	Значение
CPU (vCPU)	4

Параметр	Значение
RAM	16 ГБ
HDD	100 ГБ

Bastion

Параметр	Значение
CPU (vCPU)	2
RAM	4 ГБ
HDD	20 ГБ

3. Кластер Docker Swarm для установки Jenkins следующей конфигурации:

Jenkins controller

Параметр	Значение
CPU (vCPU)	2
RAM	8 ГБ
HDD	100 ГБ

Worker

Параметр	Значение
CPU (vCPU)	4
RAM	16 ГБ

Параметр	Значение
HDD	100 ГБ

4. Сторонние компоненты/продукты необходимые для функционирования платформы Marlin, установка которых может быть детально не описана в данном руководстве:
    - PostgreSQL 13+
    - Keycloak 20+
    - Jenkins 2.375.3
    - SonarQube 9.1+
    - Sonatype Nexus Repository 3.46+
    - HashiCorp Vault 1.7.3+
    - Minio
    - GitLab 14.10+

Требования к топологии сети

1. Все узлы кластеров должны принадлежать одному сегменту сети
2. Все узлы должны иметь доступ к ресурсам сети Интернет

Список Docker-образов, необходимых для запуска платформы Marlin

> Необходимо поместить в hosted-репозиторий хранилища артефактов Sonatype Nexus Repository перечисленные Docker-образы. Подробнее в инструкции

Системные требования по типам компонентов для развертывания в рекомендуемой конфигурации

Дополнительные требования, влияющие на ресурсы:

- Объем запускаемых приложений увеличит требования к Kubernetes или Openshift/OKD, если выходит за пределы минимальной/рекомендуемой

конфигурации

- Объем хранимой приложениями информации в Nexus/PostgreSQL/MinIO
- Требование о сетевой изоляции между сетевыми зонами LAN и DMZ увеличит требуемые ресурсы Kubernetes или Openshift/OKD в 2 раза
- Изоляция на уровне сред компонентов, таких как Vault, Keycloak, Jenkins увеличит
- Увеличение количества сред (например, наличие дополнительной среды development)

Компонент	Тип	Среда	Рекомендуемая конфигурация	Примечание
OKD Platform	Master		3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Infra		3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Worker		3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Bastion		2 шт., 1 CPU, 2 RAM, 30 GB SSD	
PostgreSQL Platform	HAProxy		2 шт., 2 CPU, 2 RAM, 20 GB SSD	Может использоваться cloud managed решение
PostgreSQL Platform	PostgreSQL		3 шт., 6 CPU, 24 RAM, 500 GB SSD	Может использоваться cloud managed решение
MinIO Platform	HAProxy		2 шт., 2 CPU, 2 RAM, 20 GB SSD	

Компонент	Тип	Среда	Рекоммендуемая конфигурация	Примечание
MinIO Platform	MinIO		4x, 2 CPU, 4 RAM, 500 GB SSD	
SonarQube			1x, 2 CPU, 4 RAM, 100 GB SSD	
GitLab	Base		1x, 4 CPU, 8 RAM, 100 GB SSD	
GitLab	Runner		1x, 8 CPU, 16 RAM, 100 GB SSD	
Nexus			1x, 4 CPU, 8 RAM, 500 GB SSD	
Prometheus			2 шт., 4 CPU, 8 RAM, 250 GB SSD	
ElasticSearch			3 шт., 4 CPU, 16 RAM, 500 GB SSD	
OKD Application	Master	Production	3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Infra	Production	3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Worker	Production	3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Bastion	Production	2 шт., 1 CPU, 2 RAM, 30 GB SSD	

Компонент	Тип	Среда	Рекоммендуемая конфигурация	Примечание
OKD Application	Master	Staging	3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Infra	Staging	3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Worker	Staging	3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Bastion	Staging	2 шт., 1 CPU, 2 RAM, 30 GB SSD	
OKD Application	Master	Testing	3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Infra	Testing	3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Worker	Testing	3 шт., 4 CPU, 16 RAM, 100 GB SSD	
	Bastion	Testing	2 шт., 1 CPU, 2 RAM, 30 GB SSD	
PostgreSQL Application	HAProxy	Production	2 шт., 2 CPU, 2 RAM, 20 GB SSD	Может использоваться cloud managed решение
PostgreSQL Application	PostgreSQL	Production	3 шт., 6 CPU, 24 RAM, 500 GB SSD	Может использоваться cloud managed решение

Компонент	Тип	Среда	Рекоммендуемая конфигурация	Примечание
PostgreSQL Application	HAProxy	Staging	2 шт., 2 CPU, 2 RAM, 20 GB SSD	Может использоваться cloud managed решение
PostgreSQL Application	PostgreSQL	Staging	3 шт., 6 CPU, 24 RAM, 500 GB SSD	Может использоваться cloud managed решение
PostgreSQL Application	HAProxy	Testing	2 шт., 2 CPU, 2 RAM, 20 GB SSD	Может использоваться cloud managed решение
PostgreSQL Application	PostgreSQL	Testing	3 шт., 6 CPU, 24 RAM, 500 GB SSD	Может использоваться cloud managed решение
MinIO Application	HAProxy	Production	2 шт., 2 CPU, 2 RAM, 100 GB SSD	
MinIO Application	MinIO	Production	4x, 2 CPU, 4 RAM, 500 GB SSD	
MinIO Application	HAProxy	Staging	2 шт., 2 CPU, 2 RAM, 20 GB SSD	
MinIO Application	MinIO	Staging	4x, 2 CPU, 4 RAM, 500 GB SSD	
MinIO Application	HAProxy	Testing	2 шт., 2 CPU, 2 RAM, 20 GB SSD	

Компонент	Тип	Среда	Рекоммендуемая конфигурация	Примечание
MinIO Application	MinIO	Testing	4x, 2 CPU, 4 RAM, 500 GB SSD	
Keycloak	HAProxy	Production	2 шт., 2 CPU, 2 RAM, 20 GB SSD	
Keycloak	Keycloak	Production	3 шт., 4 CPU, 8 RAM, 100 GB SSD	
Keycloak	HAProxy	Staging	2 шт., 2 CPU, 2 RAM, 20 GB SSD	
Keycloak	Keycloak	Staging	3 шт., 4 CPU, 8 RAM, 100 GB SSD	
Keycloak	HAProxy	Testing	2 шт., 2 CPU, 2 RAM, 20 GB SSD	
Keycloak	Keycloak	Testing	3 шт., 4 CPU, 8 RAM, 100 GB SSD	
Jenkins		Production	3 шт., 8 CPU, 16 RAM, 100 GB SSD	
Jenkins		Staging	3 шт., 8 CPU, 16 RAM, 100 GB SSD	
Jenkins		Testing	3 шт., 8 CPU, 16 RAM, 100 GB SSD	
HashiCorp Vault		Production	2 шт., 2 CPU, 4 RAM, 25 GB SSD	

Компонент	Тип	Среда	Рекоммендуемая конфигурация	Примечание
HashiCorp Vault		Staging	2 шт., 2 CPU, 4 RAM, 25 GB SSD	
HashiCorp Vault		Testing	2 шт., 2 CPU, 4 RAM, 25 GB SSD	
ИТОГ			438 CPU, 1328 RAM, 63780 GB SS	

2. Конфигурация HashiCorp Vault

> ⓘ **ПРИМЕЧАНИЕ**
>
> Предварительные требования:
>
> - Развернуный HashiCorp Vault версии не ниже 1.7.3
> - Vault в статусе "распечатан" (Unsealed)

Потребуется выполнить конфигурацию через Vault command-line interface (CLI) в командной оболочке Linux.

1. Выполнить вход cli-утилитой с использованием Root-токена:

```
vault login <root-токен>
```

2. Создать KV-хранилище секретов версии 1 (KV Secrets Engine - Version 1) с путем marlin:

```
vault secrets enable -path=marlin -version=1 kv
```

3. Создать следующие политики: **Политика генерации паролей**

```
cat << EOF | vault write sys/policies/password/jenkins policy=-
length=12

rule "charset" {
    charset = "abcdefghijklmnopqrstuvwxyz"
    min-chars = 1
}

rule "charset" {
    charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    min-chars = 1
}

rule "charset" {
    charset = "0123456789"
```

```
    min-chars = 1
}
EOF
```

Jenkins

```
cat << EOF | vault policy write jenkins-policy -
path "marlin/*" {
    capabilities = ["create", "read", "update", "delete", "list"]
}
path "/sys/auth*" {
    capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}
path "auth/*" {
    capabilities = ["create", "read", "update", "delete", "list"]
}
path "/sys/policies*" {
    capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}
EOF
```

Портал Marlin

```
cat << EOF | vault policy write java-backend-policy -
path "marlin/*" {
    capabilities = ["create", "read", "update", "delete", "list"]
}
path "/sys/auth*" {
    capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}
path "auth/*" {
    capabilities = ["create", "read", "update", "delete", "list"]
}
path "/sys/policies*" {
    capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}
EOF
```

4. Включить метод аутентификации AppRole для интеграции с Vault портала Marlin и Jenkins:

```
```

```
vault auth enable approle
```

5. Создать AppRole-роли **jenkins** и **java-backend** с соотвествующими политиками для Jenkins и портала Marlin: Jenkins

```
vault write auth/approle/role/jenkins \
    secret_id_ttl=0 \
    token_num_uses=0 \
    token_ttl=3600 \
    token_max_ttl=86400 \
    secret_id_num_uses=0 \
    policies=jenkins-policy
```

Портал Marlin

```
vault write auth/approle/role/java-backend \
    secret_id_ttl=0 \
    token_num_uses=0 \
    token_ttl=3600 \
    token_max_ttl=86400 \
    secret_id_num_uses=0 \
    policies=java-backend-policy
```

6. Прочитать Role ID для аутентификации методом AppRole Получение Role ID:

```
vault read auth/approle/role/<наименование роли>/role-id
```

Получение Role ID для Jenkins

```
vault read auth/approle/role/jenkins/role-id
```

Получение Role ID для портала Marlin

```
vault read auth/approle/role/java-backend/role-id
```

Пример успешного выполнения команды

```
vault read auth/approle/role/java-backend/role-id
# Key     Value
# role_id ffadc082-b9d1-fa1b-8525-e756659b5d9e
```

7. Сгенерировать Secret ID для ролей:

```
vault read auth/approle/role/<наименование роли>/role-id
```

Генерация Secret ID для роли **jenkins**

```
vault write -force auth/approle/role/jenkins/secret-id
```

Генерация Secret ID для роли **java-backend**

```
vault write -force auth/approle/role/java-backend/secret-id
```

Пример успешного выполнения команды

```
vault write -force auth/approle/role/java-backend/secret-id
# Key                 Value
# secret_id           bc493a4a-7e34-da21-5f12-57624415f220
# secret_id_accessor f382db47-6708-49f2-323e-ff934f36e1de
# secret_id_ttl       0
```

8. Сохранить полученные на шагах 6 и 7 Role ID и Secret ID

Добавление секретов в Vault

Перейти в веб-интерфейс **Vault → Secret Engines → marlin → Create secret** Далее, создать перечисленные ниже секреты:

1. **portal/gitlab_credentials**. Раздел 9. Конфигурация GitLab для работы с платформой Marlin

Ключ	Значение
login	marlin_git
password	пароль для пользователя marlin_git из раздела

2. **jenkins/sonarqube**. Раздел 8. Конфигурация SonarQube

Ключ	Значение
login	административная учетная запись jenkins-admin
password	пароль пользователя jenkins-admin
scanner-token	токен пользователя jenkins-scanner
token	токен пользователя jenkins
webhook-token	jenkins-webhook token

3. **jenkins/artifactory**. Раздел 5. Конфигурация Sonatype Nexus Repository

Ключ	Значение
login	marlin-system
password	пароль пользователя marlin-system из Sonatype Nexus

4. **jenkins/nexus**. Раздел 5. Конфигурация Sonatype Nexus Repository

Ключ	Значение
login	marlin-system
password	пароль пользователя marlin-system из Sonatype Nexus

5. **jenkins/ca**

> **ПРИМЕЧАНИЕ**
>
> Требуется создать приватный удостоверяющий центр (УЦ) сертификации для выпуска TLS-сертификатов с помощью Jenkins

i. Перейти в директорию с дистрибутивом openssl-scripts

```
cd openssl-scripts
```

ii. Выполнить скрипт init-ca.sh для выпуска корневого сертификата и следовать указаниям на экране:

```
./init-ca.sh root-ca
```

Пример выполнения скрипта init-ca.sh

```
Creating the root key...
Generating RSA private key, 4096 bit long modulus
.....................................................................................
.....................................................++
Then you need to enter and repeet a pass phrase to protect your CA key. It may be

Enter pass phrase for private/ca.key.pem:
Verifying - Enter pass phrase for private/ca.key.pem:

Creating the root certificate...
Enter pass phrase for private/ca.key.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:RU
State or Province Name [Germany]:Russia
Locality Name []:
Organization Name [Acme Test]:HD Tech
Organizational Unit Name []:
Common Name []:HD Tech Root CA
Email Address []:admin@hd-tech.ru
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            a6:de:34:45:64:bf:db:c4
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=RU, ST=Russia, O=HD Tech, CN=HD Tech Root CA/emailAddress=admin@
        Validity
            Not Before: Aug  3 01:51:40 2023 GMT
            Not After : Jul 29 01:51:40 2043 GMT
        Subject: C=RU, ST=Russia, O=HD Tech, CN=HD Tech Root CA/emailAddress=admin
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:e0:6d:ac:3b:85:62:c8:bb:a5:b3:2e:2e:b3:40:
                    b7:0e:96:19:bb:92:ab:d6:a7:0a:42:f9:2e:c1:b5:
                    d4:45:80:2f:91:3e:2f:41:fb:e2:b3:df:f1:73:25:
                    31:86:23:5a:da:96:ba:11:6f:f1:3f:97:9c:59:b5:
                    a9:e1:8f:7a:5d:bf:53:0b:a9:69:c9:0b:65:55:ce:
                    2e:7a:1d:29:bd:ca:6b:c2:ba:b5:b9:28:8d:22:f1:
                    32:8a:36:e3:e1:c6:df:06:b0:fb:ac:b9:45:35:2e:
                    f9:47:b3:6d:79:19:a3:98:1c:f4:de:6c:9f:24:20:
                    93:d7:fb:63:4c:6e:39:0d:42:9f:d8:b1:da:f0:6a:
                    9e:55:b0:ed:33:a8:97:10:76:2f:88:fa:39:82:e0:
                    09:03:2b:6b:60:d3:77:35:b1:bf:38:0a:3b:93:b8:
                    00:d6:5e:e6:0d:04:7b:6b:c7:d6:71:54:4d:99:0b:
                    1d:fa:78:f3:6e:3c:50:d5:5b:e5:f0:a4:2b:95:38:
                    b2:e0:d3:95:44:a2:3c:41:db:63:2b:77:80:67:a8:
                    85:e4:a1:87:27:13:7b:48:1f:64:bf:2c:71:71:e9:
                    e5:0a:3e:56:21:99:65:7e:ee:ad:c9:34:ca:c3:d2:
                    6e:7a:a4:db:b5:50:ef:0c:47:38:7b:f5:54:75:d7:
                    f6:7c:f6:41:57:4b:2e:33:45:fd:88:66:d2:e2:e5:
                    2c:bf:87:95:10:99:92:ba:33:62:96:db:9a:40:3c:
                    0c:1a:b5:b7:68:da:10:c9:fb:23:fb:8f:72:1f:fe:
                    31:08:8e:bb:9a:2e:4c:5e:b7:d1:3f:e0:36:b3:ff:
                    a7:0f:ea:d5:14:9f:b7:46:c0:06:4c:ee:e0:e7:5d:
                    0e:9c:6c:d0:45:73:64:83:ec:6e:23:a2:07:92:2b:
                    34:a1:ba:db:3b:f9:ff:57:35:ca:28:8d:3f:85:e4:
                    04:84:f1:ff:4a:26:f8:d4:8f:a1:8d:20:f5:1d:a7:
                    6d:a3:d5:ca:67:ca:6d:68:52:e6:3e:e0:72:da:08:
                    be:d2:52:62:b2:44:11:31:65:41:0f:2a:4e:44:bb:
                    f9:d5:d1:3e:76:c8:6c:5b:59:e1:02:5b:b2:4d:64:
                    52:c2:68:d6:61:fd:7d:6c:02:ca:8d:26:31:c4:1d:
                    48:b3:c2:6f:ea:9f:dc:08:61:a4:cd:b0:d9:0e:b1:
                    01:2c:4c:e7:2e:5d:d0:68:6f:d4:4d:6f:0c:bf:59:
                    33:ae:2d:f4:5e:ec:ac:a9:9b:31:72:2b:a1:5d:c3:
```

```
                58:1a:33:ac:3e:50:ed:92:02:be:67:6a:f2:76:5c:
                22:58:67:f1:45:c4:29:52:d9:cd:01:f1:66:6a:70:
                b2:57:53
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Subject Key Identifier:
            76:20:6D:3E:3B:5E:84:B4:74:33:B8:E5:7A:74:80:9E:5A:60:AE:EC
        X509v3 Authority Key Identifier:
            keyid:76:20:6D:3E:3B:5E:84:B4:74:33:B8:E5:7A:74:80:9E:5A:60:AE:EC

        X509v3 Basic Constraints: critical
            CA:TRUE
        X509v3 Key Usage: critical
            Digital Signature, Certificate Sign, CRL Sign
Signature Algorithm: sha256WithRSAEncryption
    98:c9:de:4e:cd:29:42:82:6c:d8:59:3c:97:8b:3d:11:e3:cb:
    0d:1a:70:6b:a2:77:de:5c:f0:fa:62:2c:4b:38:79:f2:5d:0d:
    4f:c7:a8:fe:c3:23:43:aa:55:17:ab:23:dc:6f:04:e7:9a:2e:
    1d:d1:5c:40:22:50:61:31:dc:c9:8e:43:e7:aa:79:01:89:f5:
    c5:76:d5:0b:52:38:a2:12:03:fd:65:ff:eb:9e:f3:ed:d6:f0:
    7e:d9:a8:09:29:e3:71:33:5a:68:2e:29:96:6a:15:6a:7d:32:
    61:0c:e7:c0:ec:18:46:8d:8b:d9:fd:ff:82:5d:fc:d0:62:49:
    5c:a7:da:91:1e:25:35:ae:22:cd:cd:db:f7:ff:a9:0d:8c:71:
    b7:0e:17:2d:ea:9b:c5:5a:9e:b1:2f:34:c5:ad:52:f9:01:47:
    4a:97:0d:d7:3f:94:52:8b:42:f1:e8:94:90:3e:1a:df:3e:96:
    5d:fb:59:83:8e:58:f5:8c:50:19:8d:06:5f:a0:8e:74:cd:34:
    15:af:43:98:6d:dd:0d:32:fc:44:c0:5e:1a:ff:a2:2d:31:f8:
    de:9c:cc:95:1c:e7:04:ca:79:ed:91:9f:f5:c5:7c:39:0a:e3:
    fc:81:fd:b4:17:c3:cd:bd:b2:ef:a0:30:a7:77:d3:a0:a2:c0:
    04:6f:30:04:2e:93:5e:f8:5e:0e:cf:e7:45:16:c2:1f:c9:76:
    92:9a:d6:63:61:e8:32:6f:30:a7:6c:88:94:2b:10:12:e2:85:
    75:18:1a:f5:9c:7a:59:3c:07:13:a4:9f:f0:a5:52:77:f6:0f:
    68:87:b6:d0:6d:ac:49:47:f7:03:46:2e:25:c5:12:e2:5b:6a:
    84:cd:d9:e3:22:79:c2:8f:c0:f3:1e:95:4c:33:8a:18:41:01:
    94:01:cc:a2:1a:1d:d8:fa:43:eb:bc:69:fc:e4:5a:33:a5:37:
    10:33:e4:26:d9:89:31:9b:af:80:08:d4:1e:1a:1f:f1:36:02:
    2d:17:0b:ed:41:a5:fd:bf:c1:29:03:85:b9:a8:5e:cc:32:94:
    38:7b:b8:e0:ca:12:e8:c6:11:f5:6f:bc:91:5f:5d:98:a9:3f:
    5e:a3:d8:5b:5d:73:c2:31:30:21:ec:ae:cc:83:ae:b7:2a:4a:
    d4:06:e5:37:ba:0e:c5:31:90:2c:9b:3a:2d:67:37:b6:43:03:
    b0:18:93:07:56:f8:06:00:37:52:6a:a0:e2:7a:a2:9e:2c:9c:
    1d:f4:68:69:a2:95:24:9b:c7:56:ff:8f:9f:bb:4b:08:f1:18:
    6a:28:77:0d:82:66:1e:53:7d:0f:d8:1d:f6:7e:ac:aa:4e:60:
    c0:7b:fc:4e:15:65:5c:1e
```

iii. Перейти в директорию root-ca и выполнить скрипт **init-interm.sh** для создания промежуточного сертификата. Далее, следовать указаниям на экране:

```
cd root-ca && ./bin/init-interm.sh IntermCA 0
```

Пример выполнения скрипта init-interm.sh

```
./bin/init-interm.sh IntermCA 0
Creating the IntermCA key...
Generating RSA private key, 4096 bit long modulus
..++
.............................................++
e is 65537 (0x10001)
Enter pass phrase for private/IntermCA.key.pem:
Verifying - Enter pass phrase for private/IntermCA.key.pem:
Creating the IntermCA certificate request...
Enter pass phrase for private/IntermCA.key.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:RU
State or Province Name [Germany]:Russia
Locality Name []:
Organization Name [Acme Test]:HD Tech
Organizational Unit Name []:
Common Name []:HD Tech Interm CA
Email Address []:admin@hd-tech.ru
Creating the IntermCA certificate...
Using configuration from /dev/fd/63
Enter pass phrase for /home/kerimov/openssl-scripts-master/root-
ca/private/ca.key.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Aug  3 01:59:56 2023 GMT
            Not After : Jul 31 01:59:56 2033 GMT
        Subject:
            countryName               = RU
            stateOrProvinceName       = Russia
```

```
            organizationName          = HD Tech
            commonName                = HD Tech Interm CA
            emailAddress              = admin@hd-tech.ru
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                5C:0B:D4:EE:28:A6:C7:8D:CB:89:73:A0:C0:E6:35:46:21:ED:4F:E7
            X509v3 Authority Key Identifier:

keyid:76:20:6D:3E:3B:5E:84:B4:74:33:B8:E5:7A:74:80:9E:5A:60:AE:EC

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
Certificate is to be certified until Jul 31 01:59:56 2033 GMT (3650 days)
Sign the certificate? [y/n]:y
```

```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
Verify the certificate...
certs/IntermCA.cert.pem: OK
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=RU, ST=Russia, O=HD Tech, CN=HD Tech Root
CA/emailAddress=admin@hd-tech.ru
        Validity
            Not Before: Aug  3 01:59:56 2023 GMT
            Not After : Jul 31 01:59:56 2033 GMT
        Subject: C=RU, ST=Russia, O=HD Tech, CN=HD Tech Interm
CA/emailAddress=admin@hd-tech.ru
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:da:bb:28:80:17:e8:e1:6d:ec:a4:4d:a3:9a:18:
                    80:d9:89:5e:a8:75:68:bb:d6:86:83:18:f3:58:1d:
                    31:0b:f4:12:b5:81:34:c9:05:51:80:e4:8b:6c:49:
                    a1:49:28:07:4d:b1:3a:a7:cb:03:93:4e:87:bf:76:
                    7c:81:0c:2f:30:24:5c:d0:da:f7:1a:01:82:4e:2b:
                    d5:1f:16:0a:75:38:bd:87:50:80:d4:f6:36:34:96:
                    c2:b1:eb:b1:2e:52:0d:53:77:5b:11:78:2e:ad:a9:
                    b9:0e:5b:7f:8a:30:41:a7:34:3f:53:e3:aa:5c:ae:
```

```
                3a:bf:dd:2c:95:cb:f6:12:19:47:ec:63:69:12:3c:
                56:41:5a:a8:be:dc:39:8e:37:85:e9:a5:78:83:12:
                a3:7b:cb:da:94:5c:83:f9:da:08:d3:62:70:f5:21:
                b9:3e:4d:a3:0c:6b:5d:69:2f:c4:33:2f:8c:f2:8f:
                0c:bc:92:55:da:9d:ab:ab:70:cc:1d:2c:05:f2:bd:
                c1:99:7b:d6:7d:48:b7:27:a6:5e:7f:56:35:14:fa:
                98:e2:e5:5c:2c:10:5f:64:bd:4e:8c:2a:7c:53:c1:
                d8:78:24:b3:c5:74:35:3a:d3:38:d2:00:5d:7c:d7:
                77:0d:c2:63:56:6d:07:a9:c5:a0:e0:d6:c6:fc:ee:
                3a:4a:4e:f6:7b:78:d6:a3:e1:c8:06:33:9f:cc:d1:
                de:d3:a5:fa:cc:ec:7a:80:5d:20:7e:c4:33:e8:c9:
                d8:9d:fc:a3:ac:7e:7e:2c:b9:7c:0f:74:9e:8b:fd:
                1d:9c:09:66:e9:7c:6d:f9:76:57:7f:c2:38:7b:3d:
                61:0d:0c:7d:3f:80:37:58:80:32:25:17:18:58:aa:
                51:a6:c5:78:d8:30:4d:0f:81:d7:e4:17:e7:53:c0:
                0e:07:3a:af:97:6b:be:c8:e3:83:23:8d:b3:d2:0d:
                6e:61:8f:ef:f6:89:2f:85:4d:63:82:2e:d2:ba:84:
                7a:62:f8:6e:77:d5:6c:5f:3c:40:e4:2c:89:ac:93:
                4b:bd:90:fe:46:c7:6b:39:bb:fb:6c:6a:c1:d8:64:
                28:52:f3:69:15:09:4c:67:9c:86:fb:28:ad:2b:a4:
                64:6d:93:a0:03:aa:8b:a9:95:4d:f6:71:55:11:4f:
                c9:88:19:76:bf:d8:59:d4:da:8e:ff:61:8b:f9:19:
                4d:95:12:55:17:78:3f:45:8f:05:d9:f4:96:84:ca:
                c5:5b:65:fe:ee:2b:cc:4b:14:b1:b9:23:6d:f8:02:
                77:36:e4:0d:34:2f:fe:47:7b:2f:c1:51:35:a9:91:
                f8:87:dc:86:71:6c:19:f5:b4:32:12:78:60:df:fb:
                0f:46:bf
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Subject Key Identifier:
            5C:0B:D4:EE:28:A6:C7:8D:CB:89:73:A0:C0:E6:35:46:21:ED:4F:E7
        X509v3 Authority Key Identifier:
            keyid:76:20:6D:3E:3B:5E:84:B4:74:33:B8:E5:7A:74:80:9E:5A:60:AE:EC

        X509v3 Basic Constraints: critical
            CA:TRUE, pathlen:0
        X509v3 Key Usage: critical
            Digital Signature, Certificate Sign, CRL Sign
Signature Algorithm: sha256WithRSAEncryption
    90:44:18:10:97:e6:80:b1:40:72:dd:17:16:e8:6e:ad:b8:03:
    0c:08:7e:6b:c0:ae:93:28:05:78:2f:81:c0:f7:6f:91:9b:2f:
    b9:ae:dd:34:17:95:33:b5:ca:a7:1e:5a:88:16:06:32:55:b1:
    36:2c:c8:04:0b:df:b5:cf:5e:9b:6c:da:fc:b8:84:83:c5:1e:
    9a:af:e5:d7:61:28:5e:2c:26:30:eb:06:4a:f2:bc:5b:dd:e5:
    e6:39:67:bb:00:16:77:55:b3:8c:ed:2e:b0:3c:10:b3:1a:6e:
    50:39:d8:e5:1b:b9:07:5d:61:8e:06:bd:34:2a:de:29:04:e7:
    d8:a0:ca:cf:ca:1e:06:4c:31:f2:66:3d:45:12:01:55:84:c2:
```

```
        3d:a0:86:c3:76:92:87:39:90:e6:7c:c9:d4:14:31:90:69:c2:
        ac:e5:c6:f9:6f:4e:d3:92:e1:6d:67:a3:48:09:84:bb:78:bf:
        f2:bd:af:54:32:65:c5:cd:50:26:0e:09:84:75:69:23:33:2b:
        0d:33:16:9e:29:e7:e9:6e:b9:cd:e5:6e:4a:79:0d:2d:34:56:
        43:f3:4c:cb:df:73:3e:63:90:29:bd:b8:16:49:cb:95:28:93:
        e2:ab:64:46:1c:95:df:6d:8a:11:39:b5:56:fc:88:4e:e1:1f:
        20:47:c2:1a:26:00:9c:16:96:48:df:03:b6:5f:b4:a8:86:50:
        6b:d8:a6:fc:e9:3c:48:6e:47:a0:16:69:38:e4:a3:13:5c:21:
        13:84:2c:cd:91:93:f1:30:14:cf:27:44:a5:5c:f7:36:a4:01:
        c2:eb:69:af:11:e0:ff:04:e0:57:b6:e7:9e:a9:7e:61:e8:2a:
        c4:48:28:11:ba:43:54:b0:b4:52:37:b5:88:5e:b5:20:de:8e:
        fb:6c:cb:c8:f2:ce:65:d5:1b:10:1d:9a:3f:e8:ba:2f:d9:81:
        8f:01:43:6f:ae:f0:6c:82:0e:dd:a4:4a:f8:b9:17:df:f3:e7:
        1b:c3:b6:34:8e:a4:8e:d0:88:bb:6b:9b:f0:60:9c:05:3b:af:
        af:35:77:48:c6:5e:8e:d3:29:c5:d3:c7:c7:95:a8:dc:a7:12:
        e7:27:0c:b1:0b:8d:6c:53:d6:cf:26:d1:c3:84:c5:86:d3:e5:
        88:9a:d1:10:1b:41:da:ef:d1:26:a8:c5:2e:31:77:44:32:44:
        e9:99:63:e6:14:bc:8c:9e:26:f5:9b:bb:82:a8:28:65:5f:39:
        e7:41:13:55:0e:57:80:94:94:c1:cd:eb:c9:76:c1:a3:cf:50:
        bd:8d:8b:1e:6d:e7:10:a9:e9:a8:9a:9d:d9:62:95:33:76:f9:
        ab:22:3a:51:d3:7d:72:bd

```
4. Скопировать содержимое файлов IntermCA/IntermCA.cert.pem и IntermCA.key.pem в соотвующие поля

Ключ	Значение
key	приватный ключ промежуточного сертификата для выпуска TLS-ключей и сертификатов для поддоменов основного домена
cert	публичный сертификат промежуточного сертификата для выпуска TLS-ключей и TLS-сертификатов для поддоменов основного домена

6. **paas/<среда>/<наименование хоста с kubeapi>**, где <среда> каждое из перечисленных значений: dev, test, stage, prod

   Выполнить из консоли команду для получения токена от Service Account **marlin-system** соотвествующей среды:

   ```
   kubectl get secret marlin-system -o jsonpath='{.data.token}' | base64 --decode
   ```

   Поместить значение токена полученное шагом выше в Vault:

Ключ	Значение
systemToken	значение токена

Выполнить из консоли команду для получения токена от Service Account **marlin** соотвествующей среды:

```
kubectl get secret marlin -o jsonpath='{.data.token}' | base64 --decode
```

Поместить значение токена полученное шагом выше в Vault:

Ключ	Значение
token	значение токена

7. **paas/<среда>/keycloak/admin**, где <среда> каждое из перечисленных значений: dev, test, stage, prod

Ключ	Значение
login	admin
password	пароль пользователя admin из keycloak Keycloak LAN

8. **paas/<среда>/keycloak/admin-dmz**, где <среда> каждое из перечисленных значений: dev, test, stage, prod

Ключ	Значение
login	admin
password	пароль пользователя admin из keycloak Keycloak DMZ

9. **paas/<среда>/camunda/admin**, где <среда> каждое из перечисленных значений: dev, test, stage, prod

Ключ	Значение
login	dummy
password	dummy

3. Конфигурация PAAS-кластеров Kubernetes или Openshift/OKD для работы с платформой Marlin

> ⓘ **ПРИМЕЧАНИЕ**
>
> Предварительные требования:
>
> - Развернутый кластер Kubernetes или Openshift/OKD в качестве PAAS
> - Пользователь с привилегиями Cluster-admin в кластере
> - Helm

1. Создать Sevice Account **marlin-system** и **marlin** для функционирования платформы Marlin в PAAS-кластерах Kubernetes или Openshift/OKD

```
kubectl create sa marlin-system
kubectl create sa marlin
```

2. Создать токены для Sevice Account **marlin-system** и **marlin marlin-service-account-token.yaml**

```
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: marlin
  annotations:
    kubernetes.io/service-account.name: marlin
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: marlin-system
  annotations:
    kubernetes.io/service-account.name: marlin-system
```

Выполнить команду:

```
kubectl apply -f marlin-service-account-token.yaml
```

3. Создать Cluster Role **clusterrole_rbac_marlin.yaml**

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  name: cluster-manager
rules:
- apiGroups: [""]
  resources: [namespaces]
  verbs: [ create, get, list, delete ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: vsk_marlin_system
rules:
- apiGroups:
  - ""
  resources:
  - namespaces
  - namespaces/status
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - user.openshift.io
  resources:
  - groups
  verbs:
  - get
  - list
  - watch
  - create
  - delete
  - deletecollection
  - patch
  - update
```

```
 - apiGroups:
   - rbac.authorization.k8s.io
   resources:
   - rolebindings
   verbs:
   - get
   - list
   - watch
   - create
   - delete
   - deletecollection
   - patch
   - update
```

4. Назначить Cluster Role для Sevice Account **marlin-system** и **marlin**: Выполнить команду для **Kubernetes**

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: view
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
- kind: ServiceAccount
  name: marlin
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: edit
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
- kind: ServiceAccount
  name: marlin
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRoleBinding
metadata:
  name: cluster-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-manager
subjects:
- kind: ServiceAccount
  name: marlin
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: vsk_marlin_system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: vsk_marlin_system
subjects:
  - name: marlin-system
    namespace: default
    kind: ServiceAccount
```

```
kubectl apply -f clusterrolebinding_marlin.yaml
```

Выполнить команду для назначения роли self-provisioner на **Openshift/OKD**

```
oc adm policy add-cluster-role-to-user self-provisioner
system:serviceaccount:default:marlin
oc adm policy add-cluster-role-to-user self-provisioner
system:serviceaccount:default:marlin-system
```

5. Назначить дополнительные роли: **marlin-kubernetes-external-secrets** и **vsk_monitor_crd_edit**

   RBAC-marlin-kubernetes-external-secrets.yaml

```
---
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRole
metadata:
  name: marlin-kubernetes-external-secrets
rules:
  - apiGroups: ["apiextensions.k8s.io"]
    resources: ["customresourcedefinitions"]
    resourceNames: ["externalsecrets.kubernetes-client.io"]
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
  - apiGroups: ["kubernetes-client.io"]
    resources: ["externalsecrets"]
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
  - apiGroups: ["kubernetes-client.io"]
    resources: ["externalsecrets/status"]
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: marlin-kubernetes-external-secrets
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: marlin-kubernetes-external-secrets
subjects:
  - name: marlin-system
    namespace: default
    kind: ServiceAccount
```

clusterrole-vsk_monitor_crd_edit.yaml

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: vsk_monitor_crd_edit
rules:
- apiGroups: ["monitoring.coreos.com"]
  resources: ["servicemonitors", "podmonitors"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

Выполнить команды:

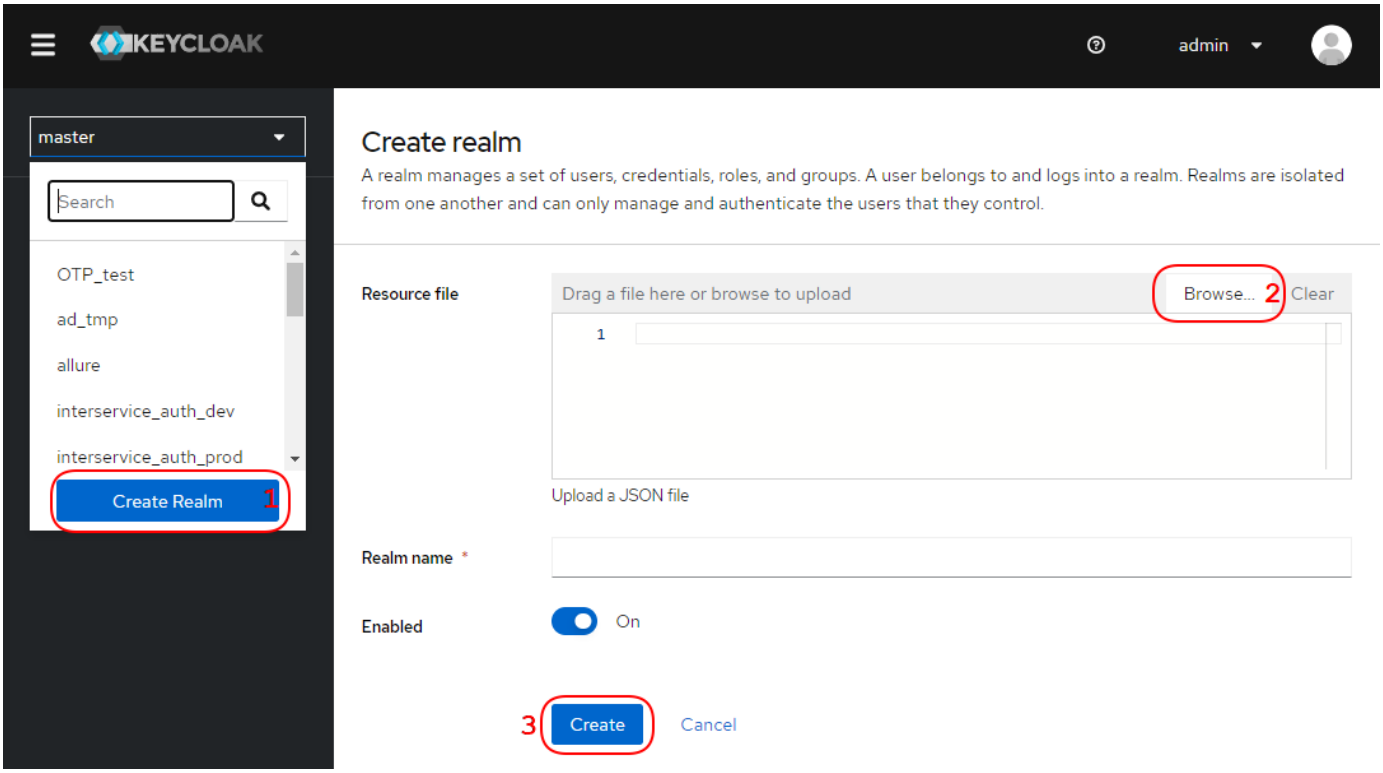
```
kubectl apply -f RBAC-marlin-kubernetes-external-secrets.yaml
kubectl apply -f clusterrole-vsk_monitor_crd_edit.yaml
```

6. Для корректной работы платформы Marlin требуется назначить дополнительные права на Service Account marlin (раздел 7, пункт 3 данного руководства)

4. Конфигурация Keycloak для работы платформы Marlin

> **ⓘ ПРИМЕЧАНИЕ**
>
> Предварительные требования:
>
> - Развернутый Keycloak в одиночной или кластерной конфигурации версии не ниже 20

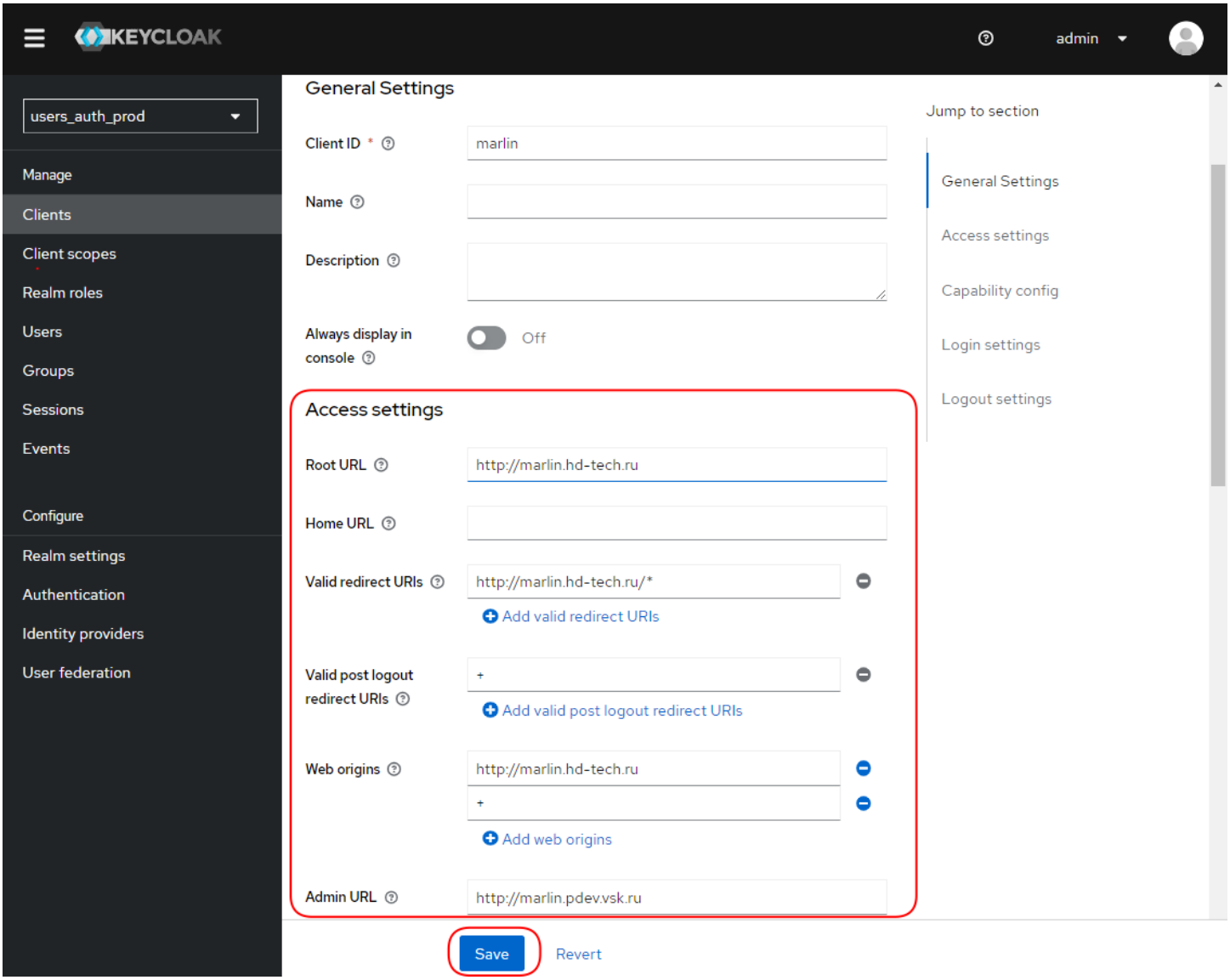
1. Перейти в веб-интерфейсе Keycloak по пути /auth/admin/master/console/#/master/add-realm для добавления нового Realm и импортировать заранее подготовленные JSON-файлы:
    - realm-export-interservice-auth-dev
    - realm-export-interservice-auth-test
    - realm-export-interservice-auth-stage
    - realm-export-interservice-auth-prod
    - realm-export-users-auth-dev
    - realm-export-users-auth-test
    - realm-export-users-auth-stage
    - realm-export-users-auth-prod



2. По необходимости отредактировать настройки в зависимости от требований нового стенда для каждого Realm'а.

3. Отредактировать парольные политики, настройки анти-брутфорса, аудита действий администратора Keycloak и администратора Realm'ов, настроить интеграции с AD(LDAP) в меню Identity Providers и/или User Federation (при необходимости).

4. Отредактировать Root URL, Valid redirect URL, Admin URL согласно имени нового стенда


KEYCLOAK
admin
users_auth_prod
Manage
Clients
Client scopes
Realm roles
Users
Groups
Sessions
Events
Configure
Realm settings
Authentication
Identity providers
User federation
General Settings
Client ID *
marlin
Name
Description
Always display in console
Off
Access settings
Root URL
http://marlin.hd-tech.ru
Home URL
Valid redirect URIs
http://marlin.hd-tech.ru/*
Add valid redirect URIs
Valid post logout redirect URIs
+
Add valid post logout redirect URIs
Web origins
http://marlin.hd-tech.ru
+
Add web origins
Admin URL
http://marlin.pdev.vsk.ru
Save
Revert
Jump to section
General Settings
Access settings
Capability config
Login settings
Logout settings

5. Установка Sonatype Nexus Repository

> ⓘ **ПРИМЕЧАНИЕ**
>
> Предварительные требования:
>
> - Серверный TLS-сертификат и закрытый ключ в PEM-формате (base64)
> - Загруженный образ sonatype/nexus3:3.46.0 из tar-архива
> - Предстановленный Docker в одиночной или кластерной конфигурации версии не ниже 20.10

1. Предварительно загрузить Docker-образ Sonatype Nexus Repository 3 из комплекта поставки. Подробно: https://docs.docker.com/engine/reference/commandline/load/

   ```
   docker load < marlin-platform-nexus3_3.46.0.tar.gz
   ```

2. На сервере, с предустановленным Docker, выполнить инициализацию Docker Swarm командой:

   ```
   docker swarm init
   ```

   Пример выполнения команды

   ```
   docker swarm init
   Swarm initialized: current node (oq8p0b1fu3c5vku5a29bnza5k) is now a manager.

   To add a worker to this swarm, run the following command:

       docker swarm join --token SWMTKN-1-
   5105e85tptmdkyigm6zqzlml9vqspap5bhdj2o21w2hc8v1lfn-dg89hni6p53fdao8zdzoplt6o
   10.0.2.15:2377

   To add a manager to this swarm, run 'docker swarm join-token manager' and
   follow the instructions.
   ```

3. Создать директории для данных и конфигурации, а следом задать права:

```
mkdir -p /data/nexus-data /data/nexus-plugins /nexus/nginx/ssl
chown -R 200:200 /data
```

4. Скопировать содержимое **nexus-stack.yaml** в файл /nexus/nexus-stack.yaml

   Содержимое файла nexus-stack.yaml:

```
version: '3.8'

services:
nexus:
    image: marlin-platform/nexus3:3.46.0
    volumes:
    - /data/nexus-data:/nexus-data
    - /etc/pki/ca-trust/extracted/:/etc/pki/ca-trust/extracted/:ro
    environment:
    - INSTALL4J_ADD_VM_PARAMS=-Xms3G -Xmx3G -XX:+AlwaysPreTouch -XX:+UseG1GC -
XX:MaxDirectMemorySize=3G -XX:+ExitOnOutOfMemoryError -
Djava.util.prefs.userRoot=/nexus-data/javaprefs
    user: 200:200
    networks:
    - net
nginx:
    image: nginx:1.19.8
    volumes:
    - /nexus/nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    - /nexus/nginx/ssl:/etc/nginx/ssl
    networks:
    - net
    ports:
    - 80:80
    - 443:443
    - 5000-5030:5000-5030
networks:
net:
    driver: overlay
    attachable: true
```

5. Серверный TLS-сертификат и закрытый ключ в PEM-формате (base64) разместить в директории /nexus/nginx/ssl. Например, для адреса nexus.hd-tech.ru соотвествующие наименования файлов: **/nexus/nginx/ssl/nexus.hd-**

tech.ru.cert.pem - для сертификата и **/nexus/nginx/ssl/nexus.hd-tech.ru.key.pem** - для ключа.

6. Отредактировать в приложенной ниже конфигурации ssl_certificate и ssl_certificate_key. Далее, поместить конфигурацию в файл /nexus/nginx/nginx.conf

Содержимое файла nginx.conf:

```
worker_processes auto;

events {
    worker_connections 1024;
}

http {

    resolver 127.0.0.11 valid=30s;
    error_log /dev/stdout;
    access_log  /dev/stdout;
    proxy_intercept_errors off;
    proxy_send_timeout 120;
    proxy_read_timeout 300;

    server {
        listen 80 default_server;
        server_name _;

        keepalive_timeout  5 5;
        proxy_buffering    off;

        set $nexus_backend nexus;

        # allow large uploads
        client_max_body_size 1G;

        location / {
            proxy_pass http://$nexus_backend:8081;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }

    server {
```

```
        listen 443 ssl;
        server_name _;
        ssl_certificate /etc/nginx/ssl/nexus.hd-tech.ru.cert.pem;
        ssl_certificate_key /etc/nginx/ssl/nexus.hd-tech.ru.key.pem;
        ssl_stapling off;

        set $nexus_backend nexus;
        client_max_body_size 1G;
        location / {
            proxy_pass http://$nexus_backend:8081;
            proxy_http_version 1.1;
            proxy_request_buffering off;
            proxy_buffering off;
            proxy_redirect          off;
            proxy_set_header        Host            $host;
            proxy_set_header        X-Real-IP       $remote_addr;
            proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header        X-Forwarded-Proto   $scheme;
        }
    }

    server {
        listen 5000-5100 ssl;
        server_name _;
        ssl_certificate /etc/nginx/ssl/nexus.hd-tech.ru.cert.pem;
        ssl_certificate_key /etc/nginx/ssl/nexus.hd-tech.ru.key.pem;
        ssl_stapling off;

        set $nexus_backend nexus;
        client_max_body_size 4G;
        location / {
            proxy_pass http://$nexus_backend:$server_port;
            proxy_http_version 1.1;
            proxy_request_buffering off;
            proxy_buffering off;
            proxy_redirect          off;
            proxy_set_header        Host            $http_host;
            proxy_set_header        X-Real-IP       $remote_addr;
            proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header        X-Forwarded-Proto   $scheme;
        }
    }
}
```

7. Развернуть Docker-сервисы командой:

```
docker stack deploy -c /nexus/nexus-stack.yml nexus
```

8. Проверить запуск сервисов командой:

```
docker service ls
```

Пример успешного запуска

```
docker service ls
ID                  NAME                MODE                REPLICAS
IMAGE                               PORTS
botpp8luxnts        nexus_nexus         replicated          1/1
marlin-platform/nexus3:3.46.0
8pe06fzabwpj        nexus_nginx         replicated          1/1
nginx:1.19.8                        *:80->80/tcp, *:443->443/tcp, *:5000-5030-
>5000-5030/tcp
```

Конфигурация Sonatype Nexus Repository

1. Перейти к веб-интерфейсу Nexus (от имени пользователя admin) по пути: **/#admin/security/users** или через элемент меню "Зубчатое колесо" → Security → Users → New user**. Далее, нажать кнопку** Create local user** и заполнить поля:

Поле	Значение
ID	marlin-system
Email Address	любой email-адрес
Password	пароль пароль пользователя
Confirm password	пароль пароль пользователя
Status	Active
Roles	nx-admin

2. Перейти к веб-интерфейсу Nexus (от имени пользователя admin) по пути: **/#admin/repository/repositories** или через элемент меню "Зубчатое колесо" → Repository → Repositories. Далее, нажать кнопку **Create repository**

3. Выбрать из списка Docker (hosted)

4. Задать следующие обязательные параметры и нажать кнопку **Create repository**:

   - Name - **marlin-platform**
   - HTTP - **5013**
   - Blob store - **default**

5. Загрузить в репозиторий образы из дистрибутива, например:

   ```
   docker load --input marlin-admiral-be.tar.gz
   ```

   Список образов:

   - marlin-platform/marlin-smtp-stub:latest
   - marlin-platform/marlin-admiral-fe:latest
   - marlin-platform/marlin-admiral-ng:latest
   - marlin-platform/marlin-admiral-jbe:v5.81.0
   - marlin-platform/marlin-admiral-be:latest
   - marlin-platform/marlin-admiral-swagger:latest
   - marlin-platform/marlin-docs:v1.6.1
   - marlin-platform/nexus3:3.46.0
   - marlin-platform/jenkins:2.249.3-lts
   - marlin-platform/jenkins-swarm-agent-java17:latest-jdk17
   - marlin-platform/jenkins-swarm-agent-dotnet3:latest
   - marlin-platform/jenkins-swarm-agent-java11:latest-jdk11
   - marlin-platform/externalsecrets/kubernetes-external-secrets:8.1.3
   - marlin-platform/envoy:v1.22.5
   - marlin-platform/openjdk:11
   - marlin-platform/openjdk:17
   - marlin-platform/dotnet/aspnet:5.0

- marlin-platform/dotnet/aspnet:6.0
- marlin-platform/dotnet/aspnet:7.0
- marlin-platform/dotnet/sdk:5.0
- marlin-platform/dotnet/sdk:6.0
- marlin-platform/dotnet/sdk:7.0
- marlin-platform/nginxinc/nginx-unprivileged:1-alpine
- marlin-platform/marlin/helm-kubectl-oc:latest
- marlin-platform/library/node:12-alpine
- marlin-platform/library/node:14-alpine
- marlin-platform/library/node:16-alpine
- marlin-platform/library/php:7.3-fpm-alpine
- marlin-platform/library/php:7.4-fpm-alpine
- marlin-platform/library/php:8.0-fpm-alpine

6. Перейти по пути: **/#admin/repository/repositories** или через элемент меню "Зубчатое колесо" → Repository → Repositories и нажать кнопку **Create repository**

7. Выбрать из списка maven2 (proxy). Далее, заполнить поля как указано ниже и нажать кнопку **Create repository**:

   - Name - **maven-central**
   - Remote storage - **https://repo1.maven.org/maven2/**
   - Blob store - **default**

8. Выбрать из списка maven2 (hosted). Далее, заполнить поля как указано ниже и нажать кнопку **Create repository**:

   - Name - **maven-hosted**
   - Version policy - **Mixed**
   - Blob store - **default**

9. Выбрать из списка npm (hosted). Далее, заполнить поля как указано ниже и нажать кнопку **Create repository**:

   - Name - **npm-hosted**
   - Blob store - **default**

6. Конфигурация Bucket в MinIO

> ⓘ **ПРИМЕЧАНИЕ**
>
> Предварительные требования:
>
> - Развернуный MinIO версии не ниже RELEASE.2020-08-27T05-16-20Z
> - MinIO Client соответсвующий серверу MinIO версии

1. Установить MinIO Client, согласно официальной документации: https://min.io/docs/minio/linux/reference/minio-mc.html#install-mc
2. Сконфигурировать подключение к серверу MinIO, например:

   ```
   mc alias set marlin-minio http://<адрес сервера minio>:9000 <access key>
   <secret key>
   ```

   где \<access key> - логин, \<secret key> - пароль
3. Создать Bucket **portal-uploads** командой:

   ```
   mc mb portal-uploads marlin-minio
   ```

7. Установка и конфигурация Kubernetes External Secrets в PAAS-кластеры Kubernetes или Openshift/OKD

> **ПРИМЕЧАНИЕ**
>
> Предварительные требования:
>
> - Развернутый кластер Kubernetes или Openshift/OKD в качестве PAAS
> - Развернуный HashiCorp Vault версии не ниже 1.7.3
> - Vault в статусе "распечатан" (Unsealed)
> - Helm

Инсталляция Kubernetes External Secrets

1. (Опционально) Создать Secret с корневым сертификатом для Пример команды по созданию сертификата:

```
cat << EOF | kubectl -n external-secrets create secret generic cacustom --from-
file=ca.pem=/dev/stdin
-----BEGIN CERTIFICATE-----
MIIFvDCCA6SgAwIBAgIJAKbeNEVkv9vEMA0GCSqGSIb3DQEBCwUAMGsxCzAJBgNV
BAYTAlJVMQ8wDQYDVQQIDAZSdXNzaWExEDAOBgNVBAoMB0hEIFRlY2gxGDAWBgNV
BAMMD0hEIFRlY2ggUm9vdCBDQTEfMB0GCSqGSIb3DQEJARYQYWRtaW5AaGQtdGVj
aC5ydTAeFw0yMzA4MDMwMTUxNDBaFw00MzA3MjkwMTUxNDBaMGsxCzAJBgNVBAYT
AlJVMQ8wDQYDVQQIDAZSdXNzaWExEDAOBgNVBAoMB0hEIFRlY2gxGDAWBgNVBAMM
D0hEIFRlY2ggUm9vdCBDQTEfMB0GCSqGSIb3DQEJARYQYWRtaW5AaGQtdGVjaC5y
dTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAOBtrDuFYsi7pbMuLrNA
tw6WGbuSq9anCkL5LsG11EWAL5E+L0H74rPf8XMlMYYjWtqWuhFv8T+XnFm1qeGP
el2/UwupackLZVXOLnodKb3Ka8K6tbkojSLxMoo24+HG3waw+6y5RTUu+UezbXkZ
o5gc9N5snyQgk9f7Y0xuOQ1Cn9ix2vBqnlWw7TOolxB2L4j6OYLgCQMra2DTdzWx
vzgKO5O4ANZe5g0Ee2vH1nFUTZkLHfp48248UNVb5fCkK5U4suDTlUSiPEHbYyt3
gGeoheShhycTe0gfZL8scXHp5Qo+ViGZZX7urck0ysPSbnqk27VQ7wxHOHv1VHXX
9nz2QVdLLjNF/Yhm0uLlLL+HlRCZkrozYpbbmkA8DBq1t2jaEMn7I/uPch/+MQiO
u5ouTF630T/gNrP/pw/q1RSft0bABkzu4OddDpxs0EVzZIPsbiOiB5IrNKG62zv5
/1c1yiiNP4XkBITx/0om+NSPoY0g9R2nbaPVymfKbWhS5j7gctoIvtJSYrJEETFl
```

```
QQ8qTkS7+dXRPnbIbFtZ4QJbsk1kUsJo1mH9fWwCyo0mMcQdSLPCb+qf3AhhpM2w
2Q6xASxM5y5d0Ghv1E1vDL9ZM64t9F7srKmbMXIroV3DWBozrD5Q7ZICvmdq8nZc
Ilhn8UXEKVLZzQHxZmpwsldTAgMBAAGjYzBhMB0GA1UdDgQWBBR2IG0+O16EtHQz
uOV6dICeWmCu7DAfBgNVHSMEGDAWgBR2IG0+O16EtHQzuOV6dICeWmCu7DAPBgNV
HRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBhjANBgkqhkiG9w0BAQsFAAOCAgEA
mMneTs0pQoJs2Fk8l4s9EePLDRpwa6J33lzw+mIsSzh58l0NT8eo/sMjQ6pVF6sj
3G8E55ouHdFcQCJQYTHcyY5D56p5AYn1xXbVC1I4ohID/WX/657z7dbwftmoCSnj
cTNaaC4plmoVan0yYQznwOwYRo2L2f3/gl380GJJXKfakR4lNa4izc3b9/+pDYxx
tw4XLeqbxVqesS80xa1S+QFHSpcN1z+UUotC8eiUkD4a3z6WXftZg45Y9YxQGY0G
X6COdM00Fa9DmG3dDTL8RMBeGv+iLTH43pzMlRznBMp57ZGf9cV8OQrj/IH9tBfD
zb2y76Awp3fToKLABG8wBC6TXvheDs/nRRbCH8l2kprWY2HoMm8wp2yIlCsQEuKF
dRga9Zx6WTwHE6Sf8KVSd/YPaIe20G2sSUf3A0YuJcUS4ltqhM3Z4yJ5wo/A8x6V
TDOKGEEBlAHMohod2PpD67xp/ORaM6U3EDPkJtmJMZuvgAjUHhof8TYCLRcL7UGl
/b/BKQOFuahezDKUOHu44MoS6MYR9W+8kV9dmKk/XqPYW11zwjEwIeyuzIOutypK
1AblN7oOxTGQLJs6LWc3tkMDsBiTB1b4BgA3Umqg4nqiniycHfRoaaKVJJvHVv+P
n7tLCPEYaih3DYJmHlN9D9gd9n6sqk5gwHv8ThVlXB4=
-----END CERTIFICATE-----
EOF
```

2. Произвести инсталляцию Kubernetes External Secrets на каждом кластере Kubernetes, который планируется использовать в качестве PAAS.

   Заполнить на примере конфигурации **marlin.dev.values.yaml** для DEV-среды ниже, с указанием адреса до Hashicorp Vault, Docker-образа и корневого сертификата (при необходимости):

```yaml
# Переменные окружения
env:
POLLER_INTERVAL_MILLISECONDS: 10000  # Caution, setting this frequency may
incur additional charges on some platforms
WATCH_TIMEOUT: 60000
WATCHED_NAMESPACES: ""  # Список список наблюдаемых неймспейсов (пусто - все)
LOG_LEVEL: info
LOG_MESSAGE_KEY: "msg"
USE_HUMAN_READABLE_LOG_LEVELS: true
METRICS_PORT: 3001
VAULT_ADDR: https://vault-dev.hd-tech.ru
#
# Корневой сертификат в доверенное хранилище сертифкатов (при необходимости)
# NODE_EXTRA_CA_CERTS: "/usr/local/share/ca-certificates/ca.pem"
#
# filesFromSecret:
# certificate-authority:
#     secret: cacustom
```

```
#     mountPath: /usr/local/share/ca-certificates
#

envVarsFromSecret: {}
envVarsFromConfigMap: {}
envFrom: {}
rbac:
create: true
serviceAccount:
create: true
annotations: {}
# The name of the service account to use.
# If not set and create is true, a name is generated using the fullname
template
name:
# Не рекомендуется использовать более чем в одном экземпляре
replicaCount: 1

# Docker-образ
image:
repository: nexus.hd-tech.ru:5013/marlin-platform/externalsecrets/kubernetes-
external-secrets
tag: 8.1.3
pullPolicy: IfNotPresent

nameOverride: ""
fullnameOverride: ""

podAnnotations: {}
podLabels: {}

priorityClassName: ""
dnsConfig: {}

securityContext:
runAsNonRoot: true
resources: {}
nodeSelector: {}
tolerations: []
affinity: {}
podDisruptionBudget: {}
serviceMonitor:
enabled: false
interval: "30s"
namespace:
deploymentInitContainers: {}
extraVolumes: []
```

```
extraVolumeMounts: []
customClusterRoles: {}
```

Пример команды:

```
helm install -n external-secrets external-secrets -f ./marlin.dev.values.yaml .
--create-namespace --debug
```

3. Создать Service Account в кластере:

```
kubectl create sa marlin
```

4. Создать кластерную роль (ClusterRole) и назначить роль (ClusterRoleBinding) созданному шагом ранее Service Account **marlin**:

```
cat << EOF | kubectl apply -f -
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
    name: marlin-kubernetes-external-secrets
rules:
- apiGroups: ["apiextensions.k8s.io"]
  resources: ["customresourcedefinitions"]
  resourceNames: ["externalsecrets.kubernetes-client.io"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["kubernetes-client.io"]
  resources: ["externalsecrets"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["kubernetes-client.io"]
  resources: ["externalsecrets/status"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
    name: marlin-kubernetes-external-secrets
roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: marlin-kubernetes-external-secrets
subjects:
```

```
- name: marlin
  namespace: default
  kind: ServiceAccount
EOF
```

5. Вручную создать Secret для Service Account:

```
cat <<EOF | kubectl -n external-secrets apply -f -
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
    name: external-secrets-kubernetes-external-secrets-token-1
    annotations:
        kubernetes.io/service-account.name: external-secrets-kubernetes-
external-secrets
EOF
```

Конфигурация Kubernetes External Secrets

Потребуется выполнить конфигурацию через Vault command-line interface (CLI) в командной оболочке Linux.

1. Включить метод Kubernetes-аутентификации в Vault:

```
vault auth enable --path=<наименование хоста с kubeapi> kubernetes
```

Пример:

```
vault auth enable --path=api.k8s.hd-tech.ru kubernetes
```

2. Получить значения переменных SA_TOKEN (токен от Service Account), CLUSTER_HOST (ссылка на кластер API), CA_CERT (TLS-сертификат API) для Kubernetes-аутентификации:

> **ПРИМЕЧАНИЕ**
>
> Выполнить команды на кластерах Kubernetes или Openshift/OKD, которые планируется использовать в качестве PAAS.**

Openshft/OKD:

```
SA_TOKEN=$(oc -n external-secrets sa get-token external-secrets-kubernetes-
external-secrets)
CLUSTER_HOST=$(oc whoami --show-server)
CA_CERT=$(openssl s_client -connect $(echo $CLUSTER_HOST | awk -F/ '{ print $3 }')
</dev/null 2>/dev/null | openssl x509)
```

Kubernetes:

```
SA_TOKEN=$(kubectl -n external-secrets get secret/external-secrets -o jsonpath="
{.data.token}" | base64 -d)
CLUSTER_HOST=$(kubectl config view -o jsonpath="{.clusters[*].cluster.server}")
CA_CERT=$(openssl s_client -connect $(echo $CLUSTER_HOST | awk -F/ '{ print $3 }')
</dev/null 2>/dev/null | openssl x509)
```

3. Сконфигурировать созданные ранее Kubernetes-аутентификации в Vault, где SA_TOKEN, CLUSTER_HOST, CLUSTER_HOST - это переменные со значениями полученными на предыдущем шаге. Конфигурацию выполнить для каждого PAAS-кластера: **Потребуется выполнить конфигурацию через Vault command-line interface (CLI) в командной оболочке Linux.**

```
vault write auth/<наименование хоста с kubeapi>/config \
    token_reviewer_jwt="${SA_TOKEN}" \
    kubernetes_ca_cert="${CA_CERT}" \
    kubernetes_host="${CLUSTER_HOST}" \
    disable_iss_validation="true" \
    disable_issuer_verification="true" \
    disable_local_ca_jwt="true"
```

8. Конфигурация SonarQube

> ⓘ **ПРИМЕЧАНИЕ**
>
> Предварительные требования:
>
> - Развернутый Sonarqube версии не ниже 9.1

1. Перейти в веб-интерфейсе SonarQube (от пользователя с правами администратора) по пути /admin/webhooks или через меню Administration → Configuration (Webhooks) → кнопка Create. Далее, заполнить перечисленные ниже поля и нажать кнопку Create:

Поле	**Значение**
Name	jenkins-webhook
URL	`http://<ссылка на jenkins>/sonarqube-webhook/`
Secret	Значение 20-символьного токена Secret вставить в value ключа webhook-token

2. Перейти в конфигурацию профиля SonarQube (Administrator) по пути /account или через меню My Account→ Security, заполнить перечисленное ниже поле и нажать кнопку Generate:

Поле	**Значение**
Enter token name	jenkins

3. Сохранить полученный на предыдущем шаге токен.

4. Перейти в веб-интерфейсе SonarQube (от пользователя с правами администратора) по пути /admin/groups или через меню Administration → Security → Groups

5. Создать группы **sonar-admins** и **sonar-scanners**

6. Перейти по пути /admin/users или через меню Administration → Security → Users
7. Создать пользователей **jenkins-admin** и **jenkins-scanner**
8. Задать пароль для пользователя jenkins-admin и сохранить пароль
9. Перейти по пути /admin/permissions или через меню Administration → Security → Global Permissions
10. Далее, отметить для группы **sonar-admins** опцию **Administer system**
11. Перейти в конфигурацию профиля по пути /account или через меню My Account→ Security
    - Enter token name - scanner-token → Generate → Скопировать и сохранить полученное значение

9. Конфигурация GitLab для работы с платформой Marlin

> ⓘ **ПРИМЕЧАНИЕ**
>
> Предварительные требования:
>
> - Развернутый Gitlab версии не ниже 14.10

1. Перейти к веб-интерфейсу GitLab по пути /admin/users/new или через меню **Menu → Admin → Users → New User**
2. Создать пользователя со следующими параметрами:
    - Name - **marlin_git**
    - Username - **marlin_git**
    - Email - **<email-адрес>**
    - Access level - **Administrator**
3. Перейти по пути /admin/users/marlin_git/impersonation_tokens или через меню **Menu → Admin → Overview → Users → marlin_git → вкладка Impersonation Tokens**
4. В разделе **Select scopes** отметить опцию **read_repository**
5. В разделе **Add an impersonation token** задать имя токена в поле **Token name** и нажать кнопку **Create impersonation Token**
6. Скопировать значение из поля "Your New Personal Access Token" и сохранить
7. Перейти по пути /admin/users/marlin_git/edit или через меню Menu → Admin → Overview → Users → marlin_git → Edit
8. Задать пароль для пользователя **marlin_git** в разделе Password и нажать кнопку **Save changes**
    - Gitlab → Menu → Admin → Settings → Network → Outbound Requests → Поставить галочку "Allow requests to the local network from hooks and services"

9. Перейти к веб-интерфейсу GitLab → Menu → Projects → Explore Projects → New project → Create blank project

    - Project name - toolsjenkins
    - Pick a group or namespace - marlin
    - Project slug - toolsjenkins

10. Перейти к веб-интерфейсу GitLab → Menu → Admin → Overview → Groups → marlin

    - Search for a user - marlin_git
    - Назначить права Owner

    Склонировать репозиторий и поместить в него содержимое директории toolsjenkins

    ```
    git clone https://gitlab.hd-tech.ru/marlin/toolsjenkins.git
    cp -R toolsjenkins/* 1/toolsjenkins/
    git add .
    git commit -m "Initial commit"
    ```

11. marlin_git personal access token "for portal" добавить в портал управления - компоненты СК - GitLab

12. Перейти к веб-интерфейсу GitLab → Menu → Projects → Explore Projects → New project → Create blank project

    - Project name - helm-rbac
    - Pick a group or namespace - root
    - Project slug - helm-rbac

13. Перейти к веб-интерфейсу GitLab → Menu → Admin → Overview → Groups → marlin

    - Search for a user - marlin_git
    - Назначить права Developer
    - Поменять protected branch на master

    ```
    git clone https://gitlab.hd-tech.ru/root/helm-rbac.git
    git checkout main
    ```

10. Установка и конфигурация Jenkins

1. Предварительно загрузить Docker-образ Jenkins из комплекта поставки. Подробно: https://docs.docker.com/engine/reference/commandline/load/

```
docker load --input jenkins.tar
```

2. Выполнить инициализацию Docker Swarm на всех узлах, с предустановленным Docker, команду вида:

```
docker swarm init
```

   Пример выполнения команды

```
docker swarm init
Swarm initialized: current node (oq8p0b1fu3c5vku5a29bnza5k) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-
5105e85tptmdkyigm6zqzlml9vqspap5bhdj2o21w2hc8v1lfn-dg89hni6p53fdao8zdzoplt6o
10.0.2.15:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and
follow the instructions.
```

3. Добавить все узлы в Docker Swarm-кластер командой:

```
docker swarm join --token <Join-токен> <IP-адрес>:2377
```

   Пример выполнения команды

```
docker swarm join --token SWMTKN-1-
5105e85tptmdkyigm6zqzlml9vqspap5bhdj2o21w2hc8v1lfn-dg89hni6p53fdao8zdzoplt6o
10.0.2.15:2377
```

4. Присвоить роль "manager" на всех узлах кластера командой:

```
docker swarm join-token manager
```

5. Вывести список узлов Docker Swarm-кластера командой:

```
docker node ls
```

   Пример выполнения команды

```
docker node ls

ID                            HOSTNAME                        STATUS
AVAILABILITY  MANAGER STATUS
dxn1zf6l61qsb1josjja83ngz *  jenkins-controller.hd-tech.ru  Ready   Active
Leader
oq8p0b1fu3c5vku5a29bnza5k    jenkins-worker1.hd-tech.ru     Ready   Active
Reachable
zfh8jl6al0ulvlx1f4c751qtk    jenkins-worker2.hd-tech.ru     Ready   Active
Reachable
```

6. Выбрать узел для размещения Jenkins-controller (master) и пометить его соотвествующей меткой:

```
docker node update --label-add jenkins-master=true <наименование узла>
```

   Пример выполнения команды

```
docker node update --label-add jenkins-master=true jenkins-controller.hd-
tech.ru
jenkins-controller.hd-tech.ru
```

7. Пометить оставшиеся узлы меткой jenkins-worker=true:

```
docker node update --label-add jenkins-worker=true <наименование узла>
```

Пример выполнения команды

```
docker node update --label-add jenkins-worker=true jenkins-worker1.hd-tech.ru
jenkins-worker1.hd-tech.ru
```

8. Проверить назначенные метки командой:

```
docker node inspect <наименование узла> --pretty
```

Пример выполнения команды

```
$ docker node inspect jenkins-controller.hd-tech.ru --pretty
ID:         dxn1zf6l61qsb1josjja83ngz
Labels:
- jenkins-worker=true
Hostname:               jenkins-controller.hd-tech.ru
Joined at:              2023-07-14 08:38:52.45508288 +0000 utc
Status:
State:          Ready
Availability:           Active
Address:        10.0.2.15
Manager Status:
Address:        10.0.2.15:2377
Raft Status:        Reachable
Leader:     Yes
Platform:
Operating System:   linux
Architecture:       x86_64
Resources:
CPUs:           2
Memory:     3.701GiB
Plugins:
Log:        awslogs, fluentd, gcplogs, gelf, journald, json-file, local, logentrie
Network:        bridge, host, ipvlan, macvlan, null, overlay
Volume:     local
Engine Version:     20.10.17
TLS Info:
TrustRoot:
-----BEGIN CERTIFICATE-----
MIIBaTCCARCgAwIBAgIUVmsGZUxxGRTPF+7Nr3hQ9XKAPokwCgYIKoZIzj0EAwIw
EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMjMwNzE0MDgzNDAwWhcNNDMwNzA5MDgz
NDAwWjATMREwDwYDVQQDEwhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
A0IABI/m/0mp4QRXKq7Dt98xK630rw7CPzyaSoC+thkgvXuPQ4Ey+OkUGWNvaGkj
```

```
dn6iQVur1Q7fXnt8iDWidEluCj6jQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
Af8EBTADAQH/MB0GA1UdDgQWBBSV0FuBD8m90lydgJ2ilfPpRXNzrTAKBggqhkjO
PQQDAgNHADBEAiBucqULsFo+6RbP4un9zRdyaZfPU17oJrkm0jLkZfwy/AIga5vn
So0laVPmlmKJHbdFgLbUHBZCnjPjPxv9AdbYOyw=
-----END CERTIFICATE-----

Issuer Subject: MBMxETAPBgNVBAMTCHN3YXJtLWNh
Issuer Public Key:
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEj+b/SanhBFcqrsO33zErrfSvDsI/PJpKgL62GSC9e49Dg
```

9. Создать директории для данных и конфигурации на всех узлах, а следом задать права:

```
mkdir -p /data/jenkins
chown -R 1005:1005 /data
```

10. Скопировать содержимое **jenkins-stack.yaml** в файл /jenkins/jenkins-stack.yaml

jenkins-stack.yaml:

```
version: '3.7'
services:
  master:
    image: marlin-platform/jenkins:2.375.3-lts
    deploy:
      placement:
        constraints:
          - node.labels.jenkins-master==true
    user: 1005:1005
    environment:
      - TZ=Europe/Moscow
      - DOCKER_HOST=unix:///var/run/docker.sock
      - CASC_JENKINS_CONFIG=
      - JAVA_OPTS=-Djenkins.install.runSetupWizard=false -Xmx4G -Xms4G -
XX:+AlwaysPreTouch
    ports:
      - "80:8080"
      - "8080:8080"
      - "50000:50000"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/login"]
      start_period: 1m
```

```
      interval: 30s
      timeout: 20s
      retries: 3
    volumes:
      - /data/jenkins/jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/bin/docker

  worker_j11:
    image: marlin-platform/jenkins-swarm-agent-java11:latest-jdk11
    deploy:
      replicas: 1
      placement:
        constraints:
          - node.labels.jenkins-worker==true
    user: root
    environment:
      - TZ=Europe/Moscow
      - COMMAND_OPTIONS=-master http://master:8080/ -executors 4 -username slave -
password changepassword -labels slave-j11 -labels slave -fsroot /home/jenkins/tmp
      - DOCKER_HOST=unix:///var/run/docker.sock
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/bin/docker
      - /data/jenkins:/home/jenkins/tmp

  worker_j17:
    image: marlin-platform/jenkins-swarm-agent-java17:latest-jdk17
    deploy:
      replicas: 1
      placement:
        constraints:
          - node.labels.jenkins-worker==true
    user: root
    environment:
      - TZ=Europe/Moscow
      - COMMAND_OPTIONS=-master http://master:8080/ -executors 4 -username slave -
password changepassword -labels slave-j17 -fsroot /home/jenkins/tmp
      - DOCKER_HOST=unix:///var/run/docker.sock
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/bin/docker
      - /data/jenkins:/home/jenkins/tmp

  worker_dotnet3:
    image: marlin-platform/jenkins-swarm-agent-dotnet3:latest
    deploy:
```

```
      replicas: 1
      placement:
        constraints:
          - node.labels.jenkins-worker==true
    user: root
    environment:
      - TZ=Europe/Moscow
      - COMMAND_OPTIONS=-master http://master:8080/ -executors 4 -username slave -
password changepassword -labels slave-dotnet3 -fsroot /home/jenkins/tmp
      - DOCKER_HOST=unix:///var/run/docker.sock
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/bin/docker
      - /data/jenkins:/home/jenkins/tmp
```

11. Подготовить на базе приведенного ниже примера конфигурационный файл и поместить в директорию **/data/jenkins/jenkins.yaml**.

> ⓘ **ПРИМЕЧАНИЕ**
>
> Заменить встречающиеся по тексту конфигурации changeadminpassword и changepassword

```
jenkins:
  systemMessage: "Jenkins configured automatically by Jenkins Configuration as Code
plugin\n\n"
credentials:
  system:
    domainCredentials:
    ### Конфигурация Gitlab (9. Конфигурация GitLab для работы с платформой Marlin)
    - credentials:
      - gitLabApiTokenImpl:
          apiToken: "<токен gitlab>"
          description: "API token пользователя marlin_git"
          id: "gitlab"
          scope: GLOBAL
    ### Начало секции Kubernetes External Secrets (7. Установка и конфигурация
Kubernetes External Secrets в PAAS-кластеры Kubernetes или Openshift/OKD)
      - string:
          description: "Service Account неймспейса External Secrets"
          id: "externalsecrets_sa"
          scope: GLOBAL
          secret: "external-secrets-kubernetes-external-secrets"
      - string:
          description: "Неймспейс External Secrets"
```

```
          id: "externalsecrets_sa_namespace"
          scope: GLOBAL
          secret: "external-secrets"
    ### Конец секции Kubernetes External Secrets
    #
    ### Начало секции Hashicorp Vault (2. Конфигурация HashiCorp Vault)
      - string:
          description: "Адрес развернутого HashiCorp Vault"
          id: "vaultAddr"
          scope: GLOBAL
          secret: "<адрес HashiCorp Vault>"
      - string:
          description: "Наименование KV-хранилища в HashiCorp Vault"
          id: "vaultStorageName"
          scope: GLOBAL
          secret: "marlin"
      - vaultUsernamePasswordCredentialImpl:
          description: "Логин и пароль от пользователя Gitlab"
          engineVersion: 1
          id: "gitlab_http"
          path: "marlin/portal/gitlab_credentials"
          scope: GLOBAL
          usernameKey: "login"
          passwordKey: "password"
      - vaultAppRoleCredential:
          description: "Реквизиты Vault AppRole для Jenkins (role id, secret id)"
          id: "vaultAppRoleID"
          path: "approle"
          roleId: "<role id>"
          scope: GLOBAL
          secretId: "<secret id>"
      ### Конец секции Hashicorp Vault
      #
      ### Начало секции PAAS кластеров. PAAS кластеры Kubernetes или OpenShift/OKD
(3. Конфигурация PAAS-кластеров Kubernetes или Openshift/OKD для работы с
платформой Marlin)
      # Заполнить для каждой из сред
      - vaultStringCredentialImpl:
          engineVersion: 1
          id: "<наименование хоста с kubeapi>"
          path: "marlin/paas/<dev|test|stage|prod>/<наименование хоста с kubeapi>"
          scope: GLOBAL
          vaultKey: "token"
      ### Конец секции PAAS кластеров
      - string:
          description: "Ссылка на git-репозиторий с исходным кодом toolsjenkins"
          id: "chart_repository"
```

```
          scope: GLOBAL
          secret: "https://<адрес gitlab>/marlin/toolsjenkins.git"
      - string:
          description: "Наименование Docker-репозитория для удаления Docker-образов
по расписанию"
          id: "dockerRepoName"
          scope: GLOBAL
          secret: "docker-marlin"
      - string:
          description: "Платформенный Docker-registry с Docker-образами"
          id: "dockerRegistryPlatform"
          scope: GLOBAL
          secret: "<адрес Docker-registry>/marlin-platform"
      - vaultUsernamePasswordCredentialImpl:
          description: "Admin SonarQube"
          engineVersion: 1
          id: "sonarqube-admin"
          passwordKey: "password"
          path: "marlin/jenkins/sonarqube"
          scope: GLOBAL
          usernameKey: "login"
      - vaultUsernamePasswordCredentialImpl:
          description: "Учетная запись для вызова API Sonatype Nexus"
          engineVersion: 1
          id: "nexus"
          passwordKey: "password"
          path: "marlin/jenkins/nexus"
          scope: GLOBAL
          usernameKey: "login"
      - vaultUsernamePasswordCredentialImpl:
          description: "Реквизиты для аутентификации в Docker-репозитории хранилища
артефактов"
          engineVersion: 1
          id: "artifactory-marlin-system"
          passwordKey: "password"
          path: "marlin/jenkins/artifactory"
          scope: GLOBAL
          usernameKey: "login"
      - vaultStringCredentialImpl:
          description: "Токен SonarQube Jenkins Webhook"
          engineVersion: 1
          id: "sonarqube-webhook"
          path: "marlin/jenkins/sonarqube"
          scope: GLOBAL
          vaultKey: "webhook-token"
      - vaultStringCredentialImpl:
          description: "Токен для sonarqube-scanner"
```

```
          engineVersion: 1
          id: "sonarqube-scanner"
          path: "marlin/jenkins/sonarqube"
          scope: GLOBAL
          vaultKey: "scanner-token"
      - vaultStringCredentialImpl:
          description: "SonarQube "
          engineVersion: 1
          id: "sonar"
          path: "marlin/jenkins/sonarqube"
          scope: GLOBAL
          vaultKey: "token"
      - vaultStringCredentialImpl:
          description: "Intermediate Marlin Ingress Private Key для выпуска
сертификатов с помощью OpenSSL"
          engineVersion: 1
          id: "ingress-key"
          path: "/marlin/jenkins/ca"
          scope: GLOBAL
          vaultKey: "key"
      - vaultStringCredentialImpl:
          description: "Intermediate Marlin Ingress Cert для выпуска сертификатов с
помощью OpenSSL"
          engineVersion: 1
          id: "ingress-cert"
          path: "/marlin/jenkins/ca"
          scope: GLOBAL
          vaultKey: "cert"
jenkins:
  agentProtocols:
  - "JNLP4-connect"
  - "Ping"
  securityRealm:
    local:
      allowsSignup: false
      enableCaptcha: false
      users:
      - id: "admin"
        name: "admin"
        password: "changeadminpassword"
        properties:
        - "apiToken"
        - favoriting:
            autofavoriteEnabled: true
        - "mailer"
        - "favorite"
        - "myView"
```

```
      - preferredProvider:
          providerId: "default"
      - "timezone"
    - id: "slave"
      name: "slave"
      password: "changepassword"
authorizationStrategy:
  loggedInUsersCanDoAnything:
    allowAnonymousRead: false
authorizationStrategy:
  globalMatrix:
    permissions:
    - "Agent/Build:slave"
    - "Agent/Configure:slave"
    - "Agent/Create:slave"
    - "Agent/Delete:slave"
    - "Agent/Disconnect:slave"
    - "Job/Read:anonymous"
    - "Job/Read:authenticated"
    - "Job/Read:slave"
    - "Overall/Administer:admin"
    - "Overall/Read:anonymous"
    - "Overall/Read:authenticated"
    - "Overall/Read:slave"
    - "View/Read:anonymous"
    - "View/Read:authenticated"
    - "View/Read:slave"
clouds:
- docker:
    disabled:
      disabledByChoice: true
      enabledByChoice: false
    dockerApi:
      connectTimeout: 60
      readTimeout: 60
    name: "docker"
crumbIssuer:
  standard:
    excludeClientIPFromCrumb: false
disableRememberMe: false
disabledAdministrativeMonitors:
- "OldData"
- "jenkins.security.QueueItemAuthenticatorMonitor"
- "hudson.diagnosis.ReverseProxySetupMonitor"
markupFormatter: "plainText"
mode: NORMAL
myViewsTabBar: "standard"
```

```
  noUsageStatistics: true
  numExecutors: 2
  primaryView:
    all:
      name: "all"
  projectNamingStrategy: "standard"
  quietPeriod: 5
  remotingSecurity:
    enabled: false
  scmCheckoutRetryCount: 0
  slaveAgentPort: 50000
  updateCenter:
    sites:
    - id: "default"
      url: "https://updates.jenkins.io/update-center.json"
  views:
  - all:
      name: "all"
  viewsTabBar: "standard"
security:
  apiToken:
    creationOfLegacyTokenEnabled: false
    tokenGenerationOnCreationEnabled: false
    usageStatisticsEnabled: true
  sSHD:
    port: -1
  scriptApproval:
    approvedSignatures:
    - "method com.datapipe.jenkins.vault.configuration.GlobalVaultConfiguration
getConfiguration"
    - "method groovy.json.JsonSlurperClassic parseText java.lang.String"
    - "method hudson.ExtensionList get java.lang.Class"
    - "method java.lang.Throwable getStackTrace"
    - "new groovy.json.JsonSlurperClassic"
    - "staticField
org.jenkinsci.plugins.pipeline.modeldefinition.parser.RuntimeASTTransformer\
      \ SCRIPT_SPLITTING_TRANSFORMATION"
    - "staticMethod
com.datapipe.jenkins.vault.configuration.GlobalVaultConfiguration\
      \ get"
    - "staticMethod groovy.json.JsonOutput toJson java.lang.Object"
    - "staticMethod java.lang.String valueOf boolean"
    - "staticMethod java.lang.String valueOf int"
    - "staticMethod jenkins.model.GlobalConfiguration all"
    - "staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods toInteger
java.lang.Number"
unclassified:
```

```
  buildDiscarders:
    configuredBuildDiscarders:
    - "jobBuildDiscarder"
    - simpleBuildDiscarder:
        discarder:
          logRotator:
            artifactDaysToKeepStr: "10"
            numToKeepStr: "10"
  defaultFolderConfiguration:
    healthMetrics:
    - worstChildHealthMetric:
        recursive: true
  fingerprints:
    fingerprintCleanupDisabled: false
    storage: "file"
  gitHubConfiguration:
    apiRateLimitChecker: ThrottleForNormalize
  gitHubPluginConfig:
    hookUrl: "http:/jenkins.hd-tech.ru/github-webhook/"
  gitLabConnectionConfig:
    connections:
    - apiTokenId: "gitlab"
      clientBuilderId: "autodetect"
      connectionTimeout: 10
      ignoreCertificateErrors: false
      name: "gitlab"
      readTimeout: 10
      url: "https://<адрес gitlab>"
  gitSCM:
    addGitTagAction: false
    allowSecondFetch: false
    createAccountBasedOnEmail: false
    disableGitToolChooser: false
    hideCredentials: false
    showEntireCommitSummaryInChanges: false
    useExistingAccountWithSameEmail: false
  globalConfigFiles:
    configs:
    - custom:
        comment: "Init Gradle Groovy script"
        content: |
          allprojects {
            buildscript {
              repositories {
                mavenLocal()
                maven { url "https://nexus.hd-tech.ru/repository/maven-central/" }
              }
```

```
            dependencies {
              classpath "org.sonarsource.scanner.gradle:sonarqube-gradle-
plugin:3.3"
            }
          }
          repositories {
            mavenLocal()
            maven { url "https://nexus.hd-tech.ru/repository/maven-central/" }
          }

          apply plugin: "java"
          apply plugin: "jacoco"

          test {
            finalizedBy jacocoTestReport
          }

          jacocoTestReport {
            dependsOn test
            reports {
              xml.enabled true
            }
          }

          afterEvaluate { project ->
            project.apply plugin: "org.sonarqube"
            project.sonarqube {
              properties {
                property 'sonar.sourceEncoding', "UTF-8"
              }
            }
          }
        id: "init.gradle"
        name: "Init Gradle"
        providerId: "org.jenkinsci.plugins.configfiles.custom.CustomConfig"
    - mavenSettings:
        comment: "Global settings"
        content: |-
          <?xml version="1.0" encoding="UTF-8"?>
          <settings xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
http://maven.apache.org/xsd/settings-1.1.0.xsd"
                    xmlns="http://maven.apache.org/SETTINGS/1.1.0"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <mirrors>
                <mirror>
                    <id>mirror.default</id>
```

```
                    <url>https://nexus.hd-tech.ru/repository/maven-central/</url>
                    <mirrorOf>external:*,!marlin</mirrorOf>
                </mirror>
            </mirrors>
            <profiles>
                <profile>
                    <id>nexus</id>
                    <repositories>
                        <repository>
                            <id>marlin</id>
                            <name>mvn-build</name>
                            <url>https://nexus.hd-tech.ru/repository/maven-
hosted/</url>
                        </repository>
                    </repositories>
                </profile>
            </profiles>
            <servers>
                <server>
                    <id>marlin</id>
                    <username>marlin-system</username>
                    <password>changepassword</password>
                </server>
            </servers>
            <activeProfiles>
                <activeProfile>nexus</activeProfile>
            </activeProfiles>
          </settings>
        id: "mvn"
        isReplaceAll: false
        name: "MyGlobalSettings"
        providerId: "org.jenkinsci.plugins.configfiles.maven.MavenSettingsConfig"
    - npm:
        comment: "user config"
        content: "registry=https://nexus.hd-tech.ru/repository/npm/"
        id: "npm"
        name: "MyNpmrcConfig"
        providerId: "jenkins.plugins.nodejs.configfiles.NPMConfig"
    - custom:
        comment: "OpenSSL Configuration for CA"
        content: |-
          [ req ]
          default_bits            = 2048
          default_md              = sha256
          default_keyfile         = tls.key
          distinguished_name      = req_distinguished_name
```

```
        string_mask = utf8only

        req_extensions = req_ext
        prompt = no

        [ req_distinguished_name ]
        countryName                     = RU
        stateOrProvinceName             = Moscow City
        localityName                    = Moscow
        0.organizationName              = HD Tech
        organizationalUnitName          = Marlin
        commonName                      = ingress-tls

        [ req_ext ]
        basicConstraints = CA:FALSE
        keyUsage = nonRepudiation, digitalSignature, keyEncipherment
        subjectAltName = @alt_names

        [ alt_names ]
        ALT_NAMES_TO_REPLACE
      id: "openssl-cnf"
      name: "MyCustom"
      providerId: "org.jenkinsci.plugins.configfiles.custom.CustomConfig"
globalLibraries:
  libraries:
  - defaultVersion: "master"
    implicit: true
    name: "vsk"
    retriever:
      modernSCM:
        scm:
          git:
            credentialsId: "gitlab_http"
            id: "b59288ea-dba1-4a86-99df-1de9a1c54015"
            remote: "https://gitlab.hd-tech.ru/marlin/toolsjenkins.git"
            traits:
            - "gitBranchDiscovery"
globalNexusConfiguration:
  instanceId: "6e38b442ce8a42b0b4694218771bd080"
hashicorpVault:
  configuration:
    engineVersion: 1
    timeout: 60
    vaultCredentialId: "vaultAppRoleID"
    vaultUrl: "https://vault.hd-tech.ru"
junitTestResultStorage:
  storage: "file"
```

```
  location:
    adminAddress: "адрес еще не настроен <nobody@hd-tech.ru>"
    url: "http://jenkins.hd-tech.ru"
  mailer:
    charset: "UTF-8"
    useSsl: false
    useTls: false
  mavenModuleSet:
    localRepository: "default"
  pollSCM:
    pollingThreadCount: 10
  sonarGlobalConfiguration:
    buildWrapperEnabled: true
    installations:
    - credentialsId: "sonar"
      name: "sonarqube"
      serverUrl: "http://sonar.hd-tech.ru:9000"
      triggers:
        skipScmCause: false
        skipUpstreamCause: false
      webhookSecretId: "sonarqube-webhook"
    - credentialsId: "sonarqube-scanner"
      name: "sonarqube"
      serverUrl: "http://sonar.hd-tech.ru:9000"
      triggers:
        skipScmCause: false
        skipUpstreamCause: false
      webhookSecretId: "sonarqube-webhook"
  timestamperConfig:
    allPipelines: true
    elapsedTimeFormat: "'<b>'HH:mm:ss.S'</b> '"
    systemTimeFormat: "'<b>'HH:mm:ss'</b> '"
tool:
  git:
    installations:
    - home: "git"
      name: "Default"
  jdk:
    installations:
    - home: "/usr/lib/jvm/java-17-openjdk-amd64"
      name: "java-17"
  maven:
    installations:
    - home: "/usr/share/maven"
      name: "maven"
  nodejs:
    installations:
```

```
    - home: "/usr/local/lib/nodejs/node-v12.18.4-linux-x64/"
      name: "nodejs"
  pipelineMaven:
    triggerDownstreamUponResultAborted: false
    triggerDownstreamUponResultFailure: false
    triggerDownstreamUponResultNotBuilt: false
    triggerDownstreamUponResultSuccess: true
    triggerDownstreamUponResultUnstable: false
  sonarRunnerInstallation:
    installations:
    - name: "SonarScanner"
      properties:
      - installSource:
          installers:
          - zip:
              subdir: "sonar-scanner-4.6.2.2472"
              url: "https://nexus.hd-tech.ru/repository/maven-
central/org/sonarsource/scanner/cli/sonar-scanner-cli/4.6.2.2472/sonar-scanner-cli-
4.6.2.2472.zip"
```

12. Развернуть сервисы следующей командой:

```
docker stack deploy -c /jenkins/jenkins-stack.yml jenkins
```

13. Перейти к веб-интерфейсу Jenkins по адресу http://<наименование узла>, например http://jenkins-controller.hd-tech.ru. Далее, войти используя учетную запись admin (пароль по умолчанию - **changeadminpassword**)

14. Перейти по пути /view/all/newJob/view/all/newJob или через пункт меню **New Item**

15. Задать элемент со значениями приведенными ниже и нажать кнопку Save:

Параметр	**Значение**
Enter an item name	portal
Тип	Folder

16. Повторить пункт 15 для элемента **devops_tools**

17. Выбрать папку **portal** для создания вложенного в него элемента **platform_services** со следующими значениями:

Параметр	**Значение**
Enter an item name	platform_services
Тип	Folder

11. Установка и конфигурация Портала Управления Marlin с помощью Helm-чарта

> **ПРИМЕЧАНИЕ**
>
> Предварительные требования:
>
> - Развернутый кластер Kubernetes или Openshift/OKD в качестве платформенного
> - Пользователь с привилегиями Cluster-admin в кластере Kubernetes или Openshift/OKD
> - Развернутый PostgreSQL в одиночной или кластерной конфигурации версии не ниже 11
> - Helm

1. Создать базу данных для платформы Marlin в СУБД PostgreSQL

```
CREATE DATABASE portal WITH TEMPLATE template0;
```

2. Создать пользователей в базе данных и назначить права для этих пользователей.

```
ALTER USER <Пользователь для БД портала> WITH SUPERUSER;
CREATE ROLE readaccess;
ALTER USER readaccess with NOLOGIN;
```

3. Выполнить скрипт приведенный ниже для получения публичного ключа из сертификата Keycloak. Далее, сохранить полученное значение публичного ключа:

> **ПРИМЕЧАНИЕ**
>
> Необходимо указать адрес развернутого Keycloak перед выполнением команд ниже

Команды для получения публичного ключа

```
echo -e "-----BEGIN CERTIFICATE-----\n$(curl -s https://<адрес keycloak>/auth/realms/users_auth_prod/protocol/openid-connect/certs | jq --raw-output '.keys[0].x5c[]')\n-----EN CERTIFICATE-----" | openssl x509 -pubkey -noout
```

Пример выполнения

```
$ echo -e "-----BEGIN CERTIFICATE-----\n$(curl -s https://auth.hd-tech.ru/auth/realms/users_auth_prod/protocol/openid-connect/certs | jq --raw-output '.keys[0].x5c[]')\n-----END CERTIFICATE-----" | openssl x509 -pubkey -noout
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAi0YiyLq3nM4wmFB/s5Ju
2Vmt7dZlEkIFbNSxWZ4n0ZZvbcNYwDRMdPTkQfKWO1ZXbgEWVGQ2DTR8ssAHI1zk
GSM8qohD2xX6+/tXk3fUBVGJ8X4CNUM7P27CJpHuG5zFkOj7fGyhcshSjtHCyDP4
iFt6azrGkv5BhJpRY/tptH9ytU9TT1ICGZRXKwt+ee6wRw+ZnoKjFS6l5ge001zd
TlIXTI6AgBaM17S7RfEGdbDeMSuftB8PZ7Qxl0EBHJ0I58WOFnz1A7L38vdgUtJF
7TwdAxXWX+OtPBYQMUK9k6fM+EnQyUKIMo2I3VBEMhnFM4/DrSDnbtJj+ZKIUB4G
BQIDAQAB
-----END PUBLIC KEY-----
```

4. Добавить полученное значение публичного ключа на шаге 5 в переменную openid_connect_public_key

5. Пройти процедуру аутентификации на выбранный для установки платформенный кластер Kubernetes или Openshift/OKD

6. Перейти в директорию с дистрибутивом marlin-helm

   ```
   cd marlin-helm
   ```

7. Поместить конфигурацию в файл marlin-helm/marlin-chart/values.yaml (в котором содержатся настройки необходимые для запуска платформы Marlin из примера ниже)

> **ПРИМЕЧАНИЕ**
>
> Заменить встречающиеся по тексту конфигурации параметры: логины и пароли (changeme), адреса такие как postgres.hd-tech.ru, ldap.hd-tech.ru, sonar.hd-tech.ru,

> gitlab.hd-tech.ru, marlin.hd-tech.ru, vault.hd-tech.ru, auth.hd-tech.ru, nexus.hd-tech.ru, jenkins.hd-tech.ru на существующие.

```
env: prods
router:
  replicas: 1
javaBackend:
  image: "nexus.hd-tech.ru:5013/marlin-platform/marlin-admiral-jbe:v5.81.0"
  replicas: 1
  cors:
    allowedOrigins: http://localhost:4200,http://localhost:3000
  datasource:
    url: jdbc:postgresql://postgres.hd-tech.ru:5432/portal
    username: portal_user
    password: changeme
    hikari:
      maximumPoolSize: 10
  liquibase:
    enabled: false
  autotest:
    enabled: true
  artifactory:
    enabled: false
  kibana:
    enabled: false
  ldap:
    urls: ldap://ldap.hd-tech.ru:389
    username: "adds\\marlin"
    password: changeme
  kafka:
    bootstrapServers: "kafka.hd-tech.ru:9092"
  rolesync:
    sonarqube:
      cron: 0 0 * * * *
      url: http://sonar.hd-tech.ru:9000
      auth:
        token: <токен sonarqube>
    ldap:
      enabled: false
      cron: 0 0 * * * *
      projects:
        base:
          dn: OU=Projects,OU=Marlin,OU=Groups,DC=adds,DC=pdev,DC=hd-tech,DC=ru
      users:
        base:
```

```
        dn: DC=adds,DC=pdev,DC=hd-tech,DC=ru
  kubernetes:
    request:
      chartUrl: https://gitlab.hd-tech.ru/marlin/helm/rbac.git
marlin:
  rubyBackend:
    url: "https://marlin.hd-tech.ru/"
  javaJenkinsProxy:
    url: "https://marlin.hd-tech.ru/private/api/hook/jenkins"
  vaults:
    dev:
      url: "https://vault.hd-tech.ru"
      roleId: "9d2d6f02-192c-47fc-ee99-9379200cadb9"
      secretId: "2bb5b715-d172-47c5-1b91-ead406a488fc"
    test:
      url: "https://vault.hd-tech.ru"
      roleId: "9d2d6f02-192c-47fc-ee99-9379200cadb9"
      secretId: "2bb5b715-d172-47c5-1b91-ead406a488fc"
    stage:
      url: "https://vault.hd-tech.ru"
      roleId: "9d2d6f02-192c-47fc-ee99-9379200cadb9"
      secretId: "2bb5b715-d172-47c5-1b91-ead406a488fc"
    prod:
      url: "https://vault.hd-tech.ru"
      roleId: "9d2d6f02-192c-47fc-ee99-9379200cadb9"
      secretId: "2bb5b715-d172-47c5-1b91-ead406a488fc"
  build:
    default:
      email: "developer@hd-tech.ru"
  gitlab:
    hook:
      url: "https://marlin.hd-tech.ru/private/api/hook/gitlab"
keycloak:
  authServerUrl: https://auth.hd-tech.ru/auth
  realm: users_auth_prod
poseidon:
  url: "http://localhost:8000"
  enabled: false
unleash:
  apiUrl: https://gitlab.hd-tech.ru/api/v4/feature_flags/unleash/34
  instanceID: MfysJTs_VtxLstzttCm5
management:
  endpoints:
    web:
      exposure:
        include: health,env
sentry:
```

```
    dsn: https://92ef31f1854543589a69183d9572b2a0@sentry.hd-tech.ru/3

backend:
  image: "nexus.hd-tech.ru:5013/marlin-platform/marlin-admiral-be:latest"
  replicas: 1
  cors:
    - http://localhost:4200
    - https://gitlab.hd-tech.ru

sidekiq:
  image: "nexus.hd-tech.ru:5013/marlin-platform/marlin-admiral-be:latest"

docs:
  image: "nexus.hd-tech.ru:5013/marlin-platform/marlin-docs:latest"
  replicas: 1

envoy:
  replicas: 1

unleashProxy:
  replicas: 1
  environments:
    UNLEASH_URL: https://gitlab.hd-tech.ru/api/v4/feature_flags/unleash/34
    UNLEASH_INSTANCE_ID: MfysJTs_VtxLstzttCm5
    UNLEASH_PROXY_CLIENT_KEYS: pdev-secret
    UNLEASH_API_TOKEN: unleash
    UNLEASH_APP_NAME: unleash-proxy

ng:
  image: "nexus.hd-tech.ru:5013/marlin-platform/marlin-admiral-ng:latest"
  replicas: 1
  backendBaseUrl: https://marlin.hd-tech.ru
  keycloakUrl: https://auth.hd-tech.ru/auth
  keycloakRealm: users_auth_prod
  sentry:
    dsn: https://76e85ec763df4ade8b21ba46a11a4b35@sentry.hd-tech.ru/2

frontend:
  image: "nexus.hd-tech.ru:5013/marlin-platform/marlin-admiral-fe:latest"
  replicas: 1
  keycloakUrl: https://auth.hd-tech.ru/auth
  websocketUrl: wss://marlin.hd-tech.ru/cable
  keycloakRealm: users_auth_prod

swagger:
  replicas: 1
```

```yaml
ingress:
  enabled: true
  className: ""
  host: marlin.hd-tech.ru
  path: /
  pathType: ImplementationSpecific
  tls: true
  issuerRef:
    group: cert-manager.io
    name: ca-issuer

ingress_grpc:
  enabled: true
  host: grpc.marlin.hd-tech.ru
  path: /
  pathType: ImplementationSpecific
  issuerRef:
    group: cert-manager.io
    name: ca-issuer
redis:
  master:
    persistence:
      storageClass: nfs-client
  replica:
    persistence:
      storageClass: nfs-client

oidcPublicKey: |-
  -----BEGIN PUBLIC KEY-----
  MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAi0YiyLq3nM4wmFB/s5Ju
  2Vmt7dZlEkIFbNSxWZ4n0ZZvbcNYwDRMdPTkQfKWO1ZXbgEWVGQ2DTR8ssAHI1zk
  GSM8qohD2xX6+/tXk3fUBVGJ8X4CNUM7P27CJpHuG5zFkOj7fGyhcshSjtHCyDP4
  iFt6azrGkv5BhJpRY/tptH9ytU9TT1ICGZRXKwt+ee6wRw+ZnoKjFS6l5ge001zd
  TlIXTI6AgBaM17S7RfEGdbDeMSuftB8PZ7Qxl0EBHJ0I58WOFnz1A7L38vdgUtJF
  7TwdAxXWX+OtPBYQMUK9k6fM+EnQyUKIMo2I3VBEMhnFM4/DrSDnbtJj+ZKIUB4G
  BQIDAQAB
  -----END PUBLIC KEY-----

sentry:
  environment: production

environments:
  #
  # Приложение
  # DEBUG - включить/выключить debug режим
  # ----------------------------------------
  DEBUG: true
```

```
  HOST: "https://marlin.hd-tech.ru/"
  JENKINS_JOB_CALLBACK_URL: https://marlin.hd-tech.ru/private/api/hook/jenkins
  # Websocket
  WEBSOCKET_HOST: wss://marlin.hd-tech.ru/cable
  ZONE: dev
  USE_SSL: false

  BACKEND_URL: "http://backend:8080"
  JAVA_BACKEND_URL: "http://java-backend:8080"
  SWAGGER_URL: "http://swagger:8080"
  # Настройки  Rails
  # ---------------------------------------
  RAILS_ENV: production
  RAILS_MAX_THREADS: 20
  RAILS_LOG_TO_STDOUT: true
  RAILS_SERVE_STATIC_FILES: true
  SECRET_KEY_BASE:
3119b75bc5396778d58b6abc0099e2f43d9d0d76b578e4ddbac2f95796a6ba5a59f8e34a451c829fce2c0
  ENCRYPTION_SALT: fdaad4b69b7b6ff28465850917004e5c41b77b456841acd5bdf6789901a6960b

  # ПО
  # ---------------------------------------
  POSTGRES_HOST: postgres.hd-tech.ru
  POSTGRES_PORT: 5432
  POSTGRES_USER: portal_user
  POSTGRES_PASSWORD: changeme
  POSTGRES_POOL: 10
  POSTGRES_DBNAME: portal

  REDIS_URL: redis://marlin-portal-redis-master:6379
  REDIS_DB_CACHE: 0
  REDIS_DB_SIDEKIQ: 1
  REDIS_DB_CABLE: 2

  # KeyCloak
  OAUTH_SCHEME: https
  OAUTH_HOST: auth.hd-tech.ru
  OAUTH_PORT: 443
  OAUTH_ID: marlin
  OAUTH_SECRET: changeme
  OAUTH_REDIRECT_REALM: users_auth_prod

  # Nexus
  NEXUS_FIRST_BASE_URL: http://nexus.hd-tech.ru
  NEXUS_SECOND_BASE_URL: http://nexus.hd-tech.ru
  NEXUS_FIRST_REPO_NAME: marlin-mvn-public
  NEXUS_SECOND_REPO_NAME: marlin-mvn-build
```

```
NEXUS_WHITELIST_REPO: http://gitlab.hd-tech.ru/marlin/toolsjenkins.git
NEXUS_SYNC_JOB_NAME: syncNexusRepos
NEXUS_ENVIRONMENT_CODE: dev

# Openshift
OPENSHIFT_DEV_HOST: https://127.0.0.1:8443
OPENSHIFT_STAGE_HOST: https://127.0.0.1:8443
OPENSHIFT_TEST_HOST: https://127.0.0.1:8443

# minio
MINIO_URL: http://minio.hd-tech.ru
MINIO_ACCESS_KEY: changeme
MINIO_SECRET_KEY: changeme
MINIO_BUCKET: portal-uploads

# gitlab
DEV_GITLAB_ADMIN: marlin_git
DEV_GITLAB_ADMIN_PASSWORD: changeme
DEV_GITLAB_URL: https://gitlab.hd-tech.ru
DEV_GITLAB_API_ENDPOINT: https://gitlab.hd-tech.ru/api/v4
DEV_GITLAB_API_PRIVATE_TOKEN: changeme

# jenkins
DEV_JENKINS_URL: http://jenkins.hd-tech.ru/
DEV_JENKINS_TEST_URL: http://jenkins.hd-tech.ru/
DEV_JENKINS_PORT: 80
DEV_JENKINS_USER: admin
DEV_JENKINS_TOKEN: 11c090cbsdfsdfse6db574a2ccd1c66b39ecc75be
DEV_JENKINS_TEST_TOKEN: 11680781bd28ee5403e885ab34cbb26b22
DEV_JENKINS_GIT_URL: http://test.gitlab.hd-tech.ru/root/toolsjenkins.git
DEV_JENKINS_GIT_BRANCH: master
DEV_JENKINS_GIT_CREDENTIAL: gitlab_http
DEV_JENKINS_BUILD_GIT_JENKINSFILE: pipelines/java/build/Jenkinsfile
DEV_JENKINS_DEPLOY_GIT_JENKINSFILE: pipelines/java/deploy/Jenkinsfile
DEV_JENKINS_STOP_START_GIT_JENKINSFILE: pipelines/java/start-stopDeploying/Jenkinsf
DEV_JENKINS_DESTROY_GIT_JENKINSFILE: pipelines/java/undeploy/Jenkinsfile
DEV_JENKINS_UPGRADE_GIT_JENKINSFILE: pipelines/java/upgrade/Jenkinsfile

# conflu
DEV_CONFLUENCE_API_ENDPOINT: https://conflu.hd-tech.ru/rest/api
DEV_CONFLUENCE_URL: https://conflu.hd-tech.ru
DEV_CONFLUENCE_USER: portal-confluence
DEV_CONFLUENCE_PASSWORD: changeme

# postgres
DEV_POSTGRES_HOST: 127.0.0.1
DEV_POSTGRES_PORT: 5432
```

```
DEV_POSTGRES_USER: postgres
DEV_POSTGRES_PASSWORD: ldap0102
DEV_POSTGRES_POOL: 10
DEV_POSTGRES_DBNAME: postgres
DEV_POSTGRES_DEPLOY_PIPELINE: pipelines/components/postgre/createdb/Jenkinsfile
DEV_POSTGRES_UNDEPLOY_PIPELINE: pipelines/components/postgre/dropdb/Jenkinsfile
POSTGRES_PREPARED_STATEMENTS: false

#minio
DEV_MINIO_URL: http://127.0.0.1
DEV_MINIO_ACCESS_KEY: admin
DEV_MINIO_SECRET_KEY: ldap0102
DEV_MINIO_DEPLOY_PIPELINE: pipelines/components/minio/createbucket/Jenkinsfile
DEV_MINIO_UNDEPLOY_PIPELINE: pipelines/components/minio/deleteBucket/Jenkinsfile

# Nexus
DEV_NEXUS_URL: http://nexus.hd-tech.ru

# Keycloak
DEV_KEYCLOAK_URL: https://auth.hd-tech.ru
DEV_KEYCLOAK_ACCESS_KEY: login
DEV_KEYCLOAK_SECRET_KEY: pass
DEV_KEYCLOAK_TOOLJENKINS: http://gitlab.hd-tech.ru/root/toolsjenkins/-/tree/master/
DEV_KEYCLOAK_CLIENT_ID: marlin
DEV_KEYCLOAK_CLIENT_SECRET: 1104ea3a-0f6b-48c4-98d6-7b55d6e6adf6
DEV_KEYCLOAK_REALM: master

# OpenShift
DEV_OPENSHIFT_URL: http://ose.hd-tech.ru
DEV_OPENSHIFT_PROJECT: marlin
DEV_OPENSHIFT_TOKEN: ab7da0bcca6d11ea87d00242ac130003
DEV_OPENSHIFT_PROTOCOL: https
DEV_OPENSHIFT_PREFIX: app-dev
DEV_OPENSHIFT_POSTFIX: apps.ose4.pdev.w55.ru

# OKD
DEV_OKD_URL: http://okd.hd-tech.ru
DEV_OKD_PROJECT: marlin
DEV_OKD_TOKEN: ab7da0bcca6d11ea87d00242ac130003
DEV_OKD_PROTOCOL: https
DEV_OKD_PREFIX: app-dev
DEV_OKD_POSTFIX: apps.ose4.pdev.w55.ru

# Kafka
DEV_INTEGRATION_ENABLE: true
DEV_KAFKA_URL: http://ose.hd-tech.ru
DEV_KAFKA_LOGIN: kafka_login
```

```
DEV_KAFKA_PASSWORD: pa55word
DEV_KAFKA_HOST: http://kafka.hd-tech.ru
DEV_KAFKA_PORT: 9092
DEV_KAFKA_DEPLOY_PIPELINE: pipelines/java/build/Kafka
DEV_KAFKA_UNDEPLOY_PIPELINE: pipelines/java/build/Kafka
DEV_KAFKA_ADD_ACL: pipelines/components/kafka/addACL/Jenkinsfile
DEV_KAFKA_DELETE_ACL: pipelines/components/kafka/deleteACL/Jenkinsfile

# Vault
DEV_VAULT_PUT_VARS: pipelines/vault/putVars/Jenkinsfile

# путь до пайплайнов дженкинса(репозиторий, из которого копируюися файлы для АП)
GITLAB_JENKINS_PIPELINE_PATH: toolsjenkins
# отвечает за проверку на бекенде, что с фронта корректно приходят названия групп по
ENVIRONMENT_GROUPS: exploitation,developer
# отвечает за то, пользователи с какими ролями попадут в список при добавлении участ
GROUPS_FOR_ADDING_USERS_TO_TEAM: exploitation,developer

# разрешать создавать сервисы, если интеграция с Confluence не установлена
SKIP_CONFLUENCE_INTEGRATION_CHECK: true

# по истечению этого интервала времени (в минутах) развертывания в 'ожидании' перех
DEPLOYMENTS_UPDATE_RETENTION: 5
```

8. Выполнить команду инсталляции с помощью программы helmwave. Подробное описание по установке доступно по ссылке https://docs.helmwave.app/0.30.x/install/

```
helmwave up --yml --build
```

Helm чарт для деплоя ПУ Marlin в кластер OKD/OSE/k8s.

Авторизация на кластере

Через консоль требуется быть авторизованным на кластере с Kubernetes

Для OKD

- Ввыполнить комаду `oc login <адрес мастер-ноды вида https://api.FQDN:6443>` в выводе появится ссылка для получения токена авторизации, по которой нужно перейти в браузере, например `https://oauth-openshift.apps.FQDN/oauth/token/request` Далее скопировать из браузера команду для авторизации на кластере
- И выполнить её в консоли `oc login --token=SomeToken --server=https://api.FQDN:6443`

Протестировать чарт можно командами:

`helm template -n <имя неймспейса> <имя разворачивания> -f <путь к файлу с дополительными переменными стенда> . --create-namespace --dry-run --debug` в выводе будут показаны полные итоговые yaml-файлы чарта, которые удобно анализировать на предмет ошибок

Валидировать на конкретном кластере Kubernetes (OKD) можно командой:

`helm template --validate -n <имя неймспейса> <имя разворачивания> -f <путь к файлу с дополительными переменными стенда> .`

Разворачивать чарт нужно командой:

`helm install -n <имя неймспейса> <имя разворачивания> -f <путь к файлу с дополительными переменными стенда> . --create-namespace`

Пререквизиты для разворачивания

1. Создать ClusterRole и RoleBinding: oc create -f prerequisites.yaml
2. Добавить ClusterRole к сервисной УЗ от имени которой деплоится CI: oc adm policy add-cluster-role-to-user cert-manager-deployer system:serviceaccount:default:marlin-deployer

При необходимости изменения ClusterRole и/или RoleBinding необходимо:

- записать изменения в prerequisites.yaml
- применить изменения из консоли: oc apply -f prerequisites.yaml

OpenSSL CA scripts

Description

The repository is a set of scripts to automate creation of a CA on a base of the OpenSSL tool. It supports:

- Creation of a CA
- Creation of intermediate cetificates
- Creation of server certificates that conform Google Chrome requirements
- Creation of client certificates
- (TODO) Revokation lists

All of this was possible because of the great article of Jamie Nguyen and couple of other internet sources.

The scripts are mostly intended to be used for generation of test certificates. If you plan to use it in production please read the article first and **examine the sources**.

TL;DR;

Here is a summary of supported commands:

```
# create a CA
./init-ca root-ca
# change the current folder
cd root-ca
# create an intermediate CA
./bin/init-interm.sh IntermCA 0
# change the current folder
cd IntermCA
# create a server certificate
./bin/create-server-cert.sh test.example.com
# create a client certificate
./bin/create-client-cert.sh alexander.sedov 201601011200000Z 201712312359592
# create a PFX file from a client certificate
./bin/pem2pfx.sh alexander.sedov
```

Usage

Folder structure

Each of the commands that generate either a CA certificate or an intermediate certificate create the next folder structure:

- `bin` the folder where the scripts go
- `certs` contains PEM files with certificate data
- `private` contains corresponding PEM files with key data
- `csr` contains certificate requests PEM files
- `crl` contains revocation lists files (TODO)
- `openssl.cnf` is a configuration file that is used during the certificate creation on this level
- `index.txt`, `serial`, `crlnumber` are internals that keep the state

Creation of the CA certificate

In the root of the repository run the next command

```
$ ./init-ca.sh root-ca
Creating the root key...
Generating RSA private key, 4096 bit long modulus
.....................................................................................
.....................................................++
```

Then you need to enter and repeet a pass phrase to protect your CA key. It may be an arbitrary sequence of symbols. You need to remember it!

```
Enter pass phrase for private/ca.key.pem:
Verifying - Enter pass phrase for private/ca.key.pem:
```

The next step is the creation of the root certificate itself. You need to enter the pass phrase of your CA key to proceed. Make sure to specify the common name and the email address fields.

```
Creating the root certificate...
Enter pass phrase for private/ca.key.pem:
```

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:
State or Province Name [Germany]:
Locality Name []:
Organization Name [Acme Test]:
Organizational Unit Name []:CA
Common Name []:Acme Test CA
Email Address []:admin@acme.com
```

Now you have a self-signed CA certificate.

Creation of an intermediate certificate

In this example, we will create two nested intermediate certificates. Before the creation of an intermediate certificate you need to specify how many of nested intermediates are you going to allow. So for the first intermediate certificate we specify `1` as one more nested intermediate certificate is to be created.

```
# Move to the generated folder of the CA first
$ cd root-ca
$ ./bin/init-interm.sh interm-adm 1
Creating the interm-adm key...
Generating RSA private key, 4096 bit long modulus
.........................................++
.........................++
```

Then you need to enter and repeat a pass phrase to protect your **intermediate** certificate's key.

```
Enter pass phrase for private/interm-adm.key.pem:
Verifying - Enter pass phrase for private/interm-adm.key.pem:
```

Reenter the pass phrase for your intermediate certificate's key to allow `openssl` to create a certificate request. Make sure to specify values for the common name and the email address

fields. Also, in case of the intermediate certificates creation the script uses a strict policy that requres an intermediate certificate has the same values for `Country Name`, `State or Province Name` and `Organization Name` fields as in the CA certificate.

```
Creating the interm-adm certificate request...
Enter pass phrase for private/interm-adm.key.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:   <-- the same as in CA
State or Province Name [Germany]:    <-- the same as in CA
Locality Name []:
Organization Name [Acme Test]:       <-- the same as in CA
Organizational Unit Name []:Adm
Common Name []:Acme Test interm Adm
Email Address []:admin@acme.com
```

The next step is signing the intermediate certificate with the CA's key. So you will be asked to enter the **CA key**'s pass phrase.

```
Creating the interm-adm certificate...
Using configuration from /dev/fd/63
Enter pass phrase for /path/to/root-ca/private/ca.key.pem:
Check that the request matches the signature
Signature ok
```

After that, you need to confirm twice and here it is - your intermediate certificate is ready and can be found in the `root-ca/interm-adm/certs` folder. It is issued and signed by the previously created CA certificate.

The creation of a nested intermediate certificate is straightforward. You need to go on the level of the `interm-adm` (parent) certificate and execute the same commands as for the creation of the first intermediate certificate.

```
cd interm-adm
./bin/init-interm.sh interm-hub
```

The certificate that is created is an edge level intermediate certificate so the number of allowed nested intermediates should be zero. Zero is the default value so you don't need to specify it as an argument. All other things are the same

- Enter and repeat a pass phrase to protect the key of the new certificate
- Enter it again to create a certificate request
- Specify certificate's DN details. Don't forget to use the same values for `Country Name`, `State or Province Name` and `Organization Name` fields as in the CA certificate. Fields `Common Name` and `Email Address` are required.
- Enter the pass phrase of the **parent** intermediate certificate (it is `interm-adm` in our case) to sign the new certificate

That's it, the new intermediate certificate is ready. It is issued and signed by the parent intermediate certificate authority.

Creation of a server certificate

A server certificate is used during an SSL handshake to establish an HTTPS connection between a client browser and your site. Typically you want the server certificate to be accessible without entering the pass phrase of its key on each OS startup so we are going to generate the key without an encription.

Run the next command and provide a DNS name of your site as an argument.

```
$ cd root-ca/interm-adm/interm-hub
$ ./bin/create-server-cert.sh test.example.com
Creation of the test.example.com key...
Generating RSA private key, 2048 bit long modulus
.....................................................+++
......+++
```

There will be no pass phrase request as we are not encrypting the key. Then, a certificate request is created and you are asked to provide distinguished name details for the server certificate. The required fields are 'Common Name' and `Email Address`. **Make sure to enter the main DNS of your site as a value for the Common Name field.**

```
Creation of the hub.test certificate request...
You are about to be asked to enter information that will be incorporated
```

```
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:
State or Province Name [Germany]:
Locality Name []:
Organization Name [Acme Test]:
Organizational Unit Name []:Hub
Common Name []:test.example.com    <-- here it is
Email Address []:admin@acme.com
```

After that, you need to provide the pass phrase of the parent intermediate certificate (interm-hub) to sign the server certificate.

```
Creation of the hub.test certificate...
Using configuration from /dev/fd/63
Enter pass phrase for /path/to/root-ca/interm-adm/interm-hub/private/interm-
hub.key.pem:
Check that the request matches the signature
Signature ok
```

The server certificate is ready. Check out the SAN section in the certificate

```
X509v3 Subject Alternative Name:
    DNS:test.example.com
```

The section is required to make Google Chrome happy. Your site may be accessible via several different DNS names and you may want to use the same server certificate for all of them. In this case, please edit the `create-server-cert.sh` file and add more DNS names into the `[ alt_names ]` section. Lately it may be automated by passing all DNS names as arguments to the script.

Creation of a client certificate

Client certificates are used for the client certificate authentication during the SSL handshake. Imagine, you want to give an access to a site only to those users who have a certificate (and its private key) issued by your CA or an intermediate authority.

Note that you are not asked to enter a pass phrase to protect the key of the cretificate. It is because scripts are intended to serve testing purposes. If you want to generate client certificates for production edit the `create-client-cert.sh` script and change the generation lines as follows. The difference here is `-aes256` parameter.

```
openssl -aes256 genrsa -out private/$USER_NAME.key.pem 2048
# openssl genrsa -out private/$USER_NAME.key.pem 2048
```

To create a client certificate use the next command, provide a meaningful certificate name as an argument. The name is used to name created files only. Also, optional start and end certificate dates are supported as the second and the thitd parameter. If either start or end date are omitted than the certificate is created with a validity period starting from now and 375 days long.

```
$ cd root-ca/interm-adm/interm-hub
$ ./bin/create-client-cert.sh "alexander sedov" 20160101120000Z 20171231235959Z
Creation of the alexander_sedov key...
Generating RSA private key, 2048 bit long modulus
........+++
.....................................................+++
```

On the next step you are asked to provide values for the distinguished name of the certificate. Make sure to fill the `Common Name` and `Email Address`.

```
Creation of the alexander_sedov certificate request...
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:
State or Province Name [Germany]:
Locality Name []:
Organization Name [Acme Test]:
Organizational Unit Name []:Hub
Common Name []:Alexander Sedov
Email Address []:alexander.sedov@mail.com
```

Then, enter the pass phrase of the parent certificate (interm-hub) key to sign the client certificate.

```
Creation of the alexander_sedov certificate...
Using configuration from openssl.cnf
Enter pass phrase for /path/to/root-ca/interm-adm/interm-hub/private/interm-
hub.key.pem:
Check that the request matches the signature
Signature ok
```

The certificate is ready and may be found in the `root-ca/interm-adm/interm-hub/certs` folder. The corresponding key is in the `root-ca/interm-adm/interm-hub/private` folder.

Troubleshooting

- Check if the `openssl` is installed in your system and is in the PATH
- Try to go through the process using the same pass phrase for all (CA and intermediates) keys. Then, if everything goes fine, try to start from the begining using different pass phrases.
- Don't forget to `cd` before executing the `init-interm.sh` script
- Plan the certificate path length from the begining. Basically, when you create a top level intermediate certificate you need to know how many nested intermediates you will create.

Руководство администратора

1. Настройка Портала Управления

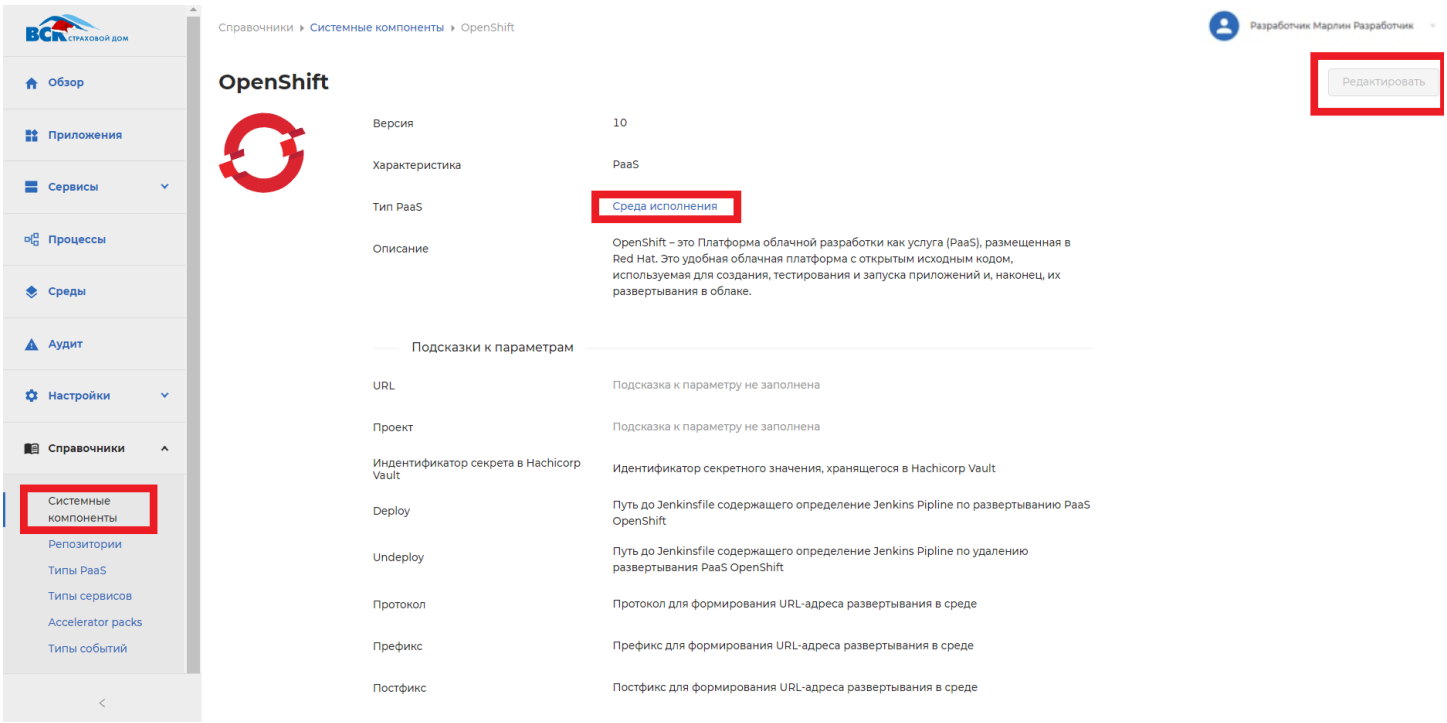
> ⓘ **ПРИМЕЧАНИЕ**
>
> Необходимые права: Администратор
>
> Предварительные действия: Не требуются

Настройка среды исполнения

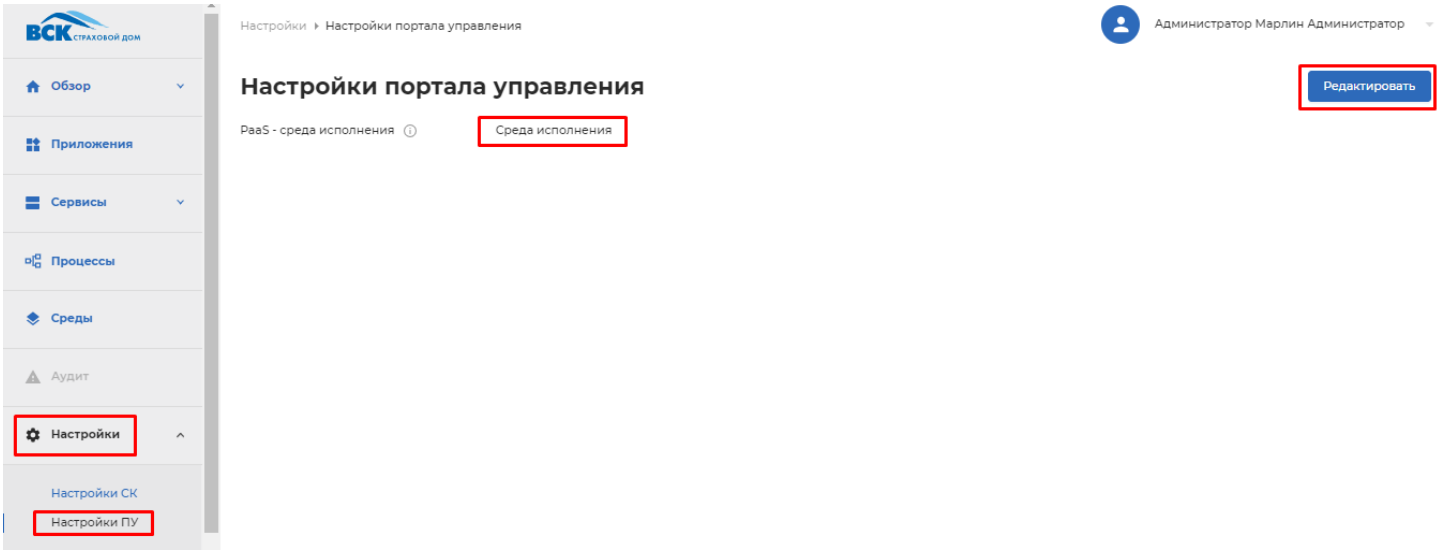
1. Перейдите на вкладку "Справочники" → "Системные компоненты" и выберите из списка необходимую системную компоненту (далее СК);

2. Нажмите на кнопку "Редактировать" в карточке СК;

3. Выберите тип PaaS "Среда исполнения";

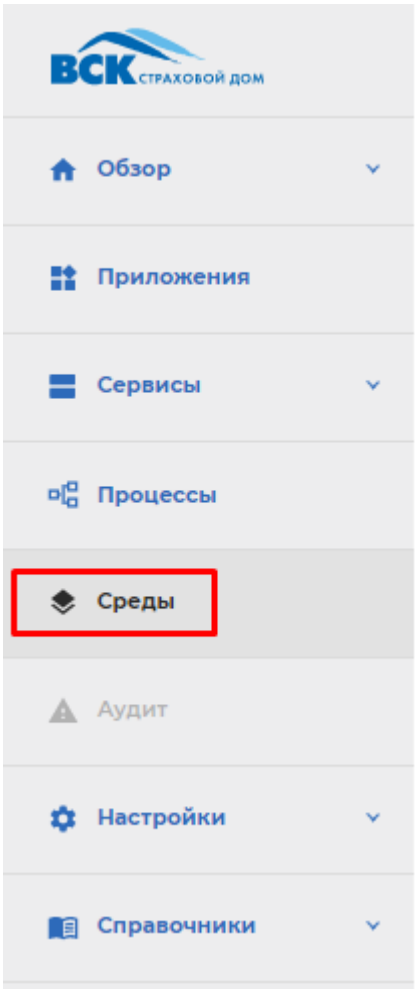


4. Перейдите на вкладку "Настройки" → "Настройки ПУ" и укажите значение "Среда исполнения".



Настройка новой среды

1. Перейдите на вкладку "Среды" в главном меню;



2. Нажмите на кнопку "Новая среда";

3. Заполните обязательные поля:

- Полное название;
- Описание;
- Код;
- Протокол для URL-адреса развертывания;
- Префикс для URL-адреса развертывания;
- Постфикс для URL-адреса развертывания;
- Доступна пользователям из групп;
- Пропустить этап сборки;
- Пропустить этап эксплуатации;
- Цвет метки.

Пример заполнения:

Поля для заполнения	**Перечень значений для среды "Разработка"**
Полное название	Среда разработки
Описание	Среда разработки
Код	dev
Протокол для URL-адреса развертывания	https
Префикс для URL-адреса развертывания	app
Постфикс для URL-адреса развертывания	ose.apps.pdev.w55.ru

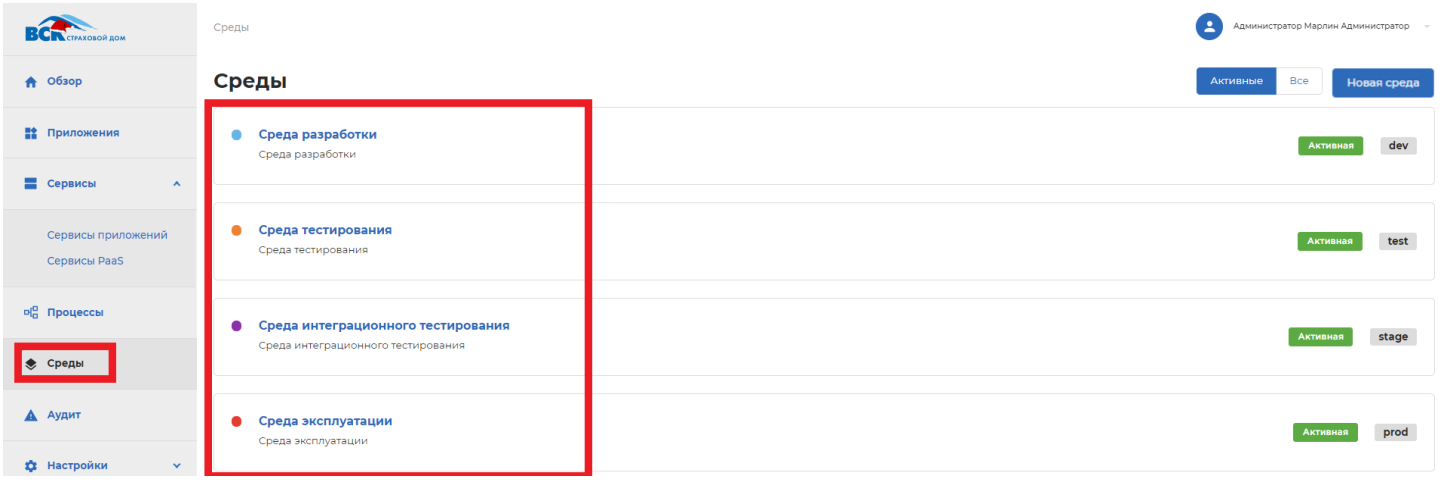
Поля для заполнения	Перечень значений для среды "Разработка"
Доступна пользователям из групп	developer
Пропустить этап сборки	нет
Пропустить этап эксплуатации	да

4. Нажмите на кнопку "Сохранить".

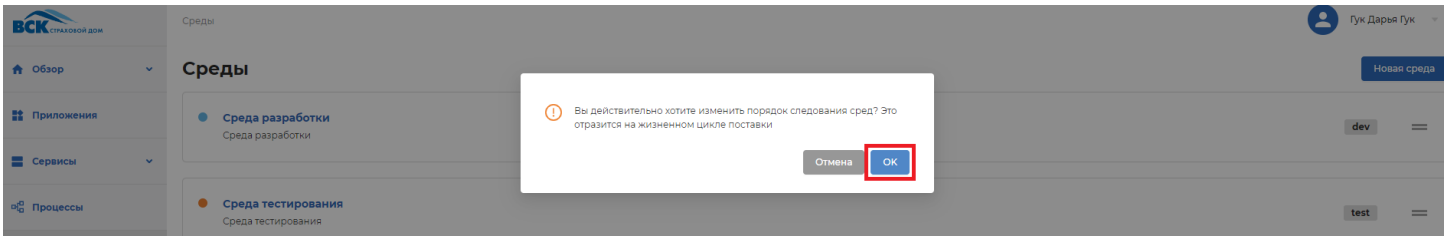
Настройка последовательности сред

> **К СВЕДЕНИЮ**
>
> От применимых настроек зависит жизненный цикл поставки

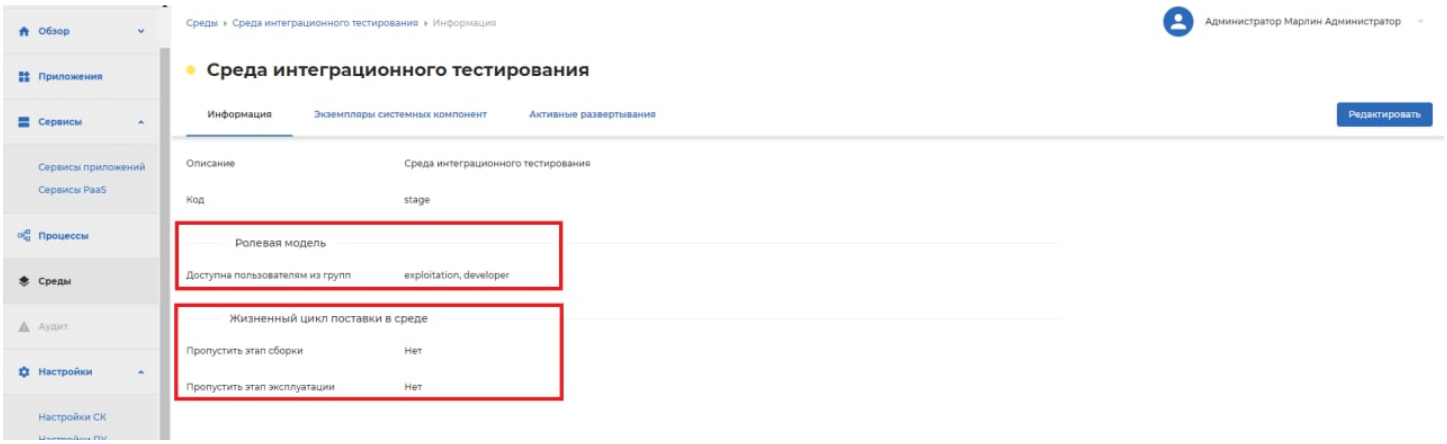
1. Перейдите на вкладку "Среды" и выставите среды нужной последовательности;



2. Нажмите на кнопку "Ок" в открывшемся окне;

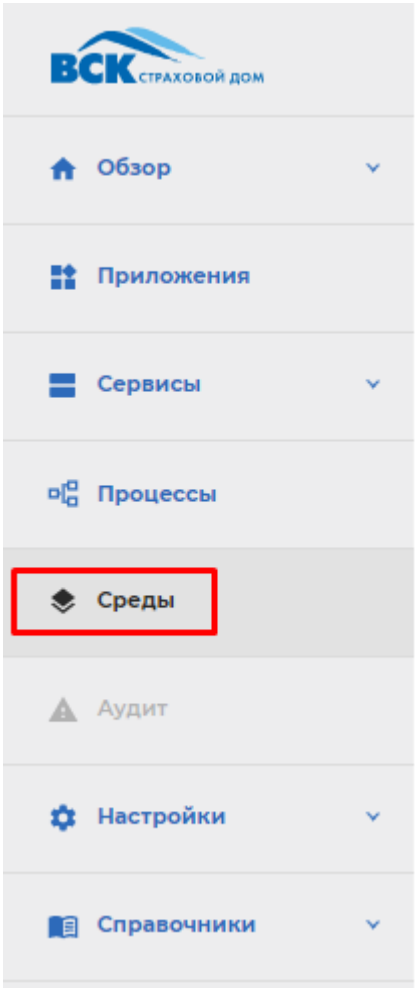


3. Проведите настройку каждой среды. Для этого выберите из списка необходимую среду и заполните раздел "Ролевая модель" и "Жизненный цикл поставки в среде".

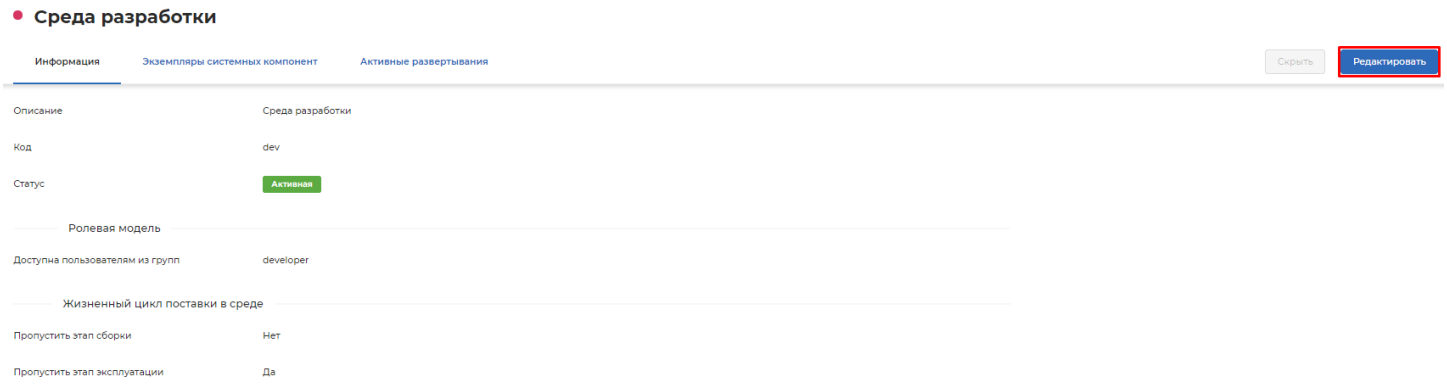


Настройка Экземплятор системных компонент (далее ЭСК)

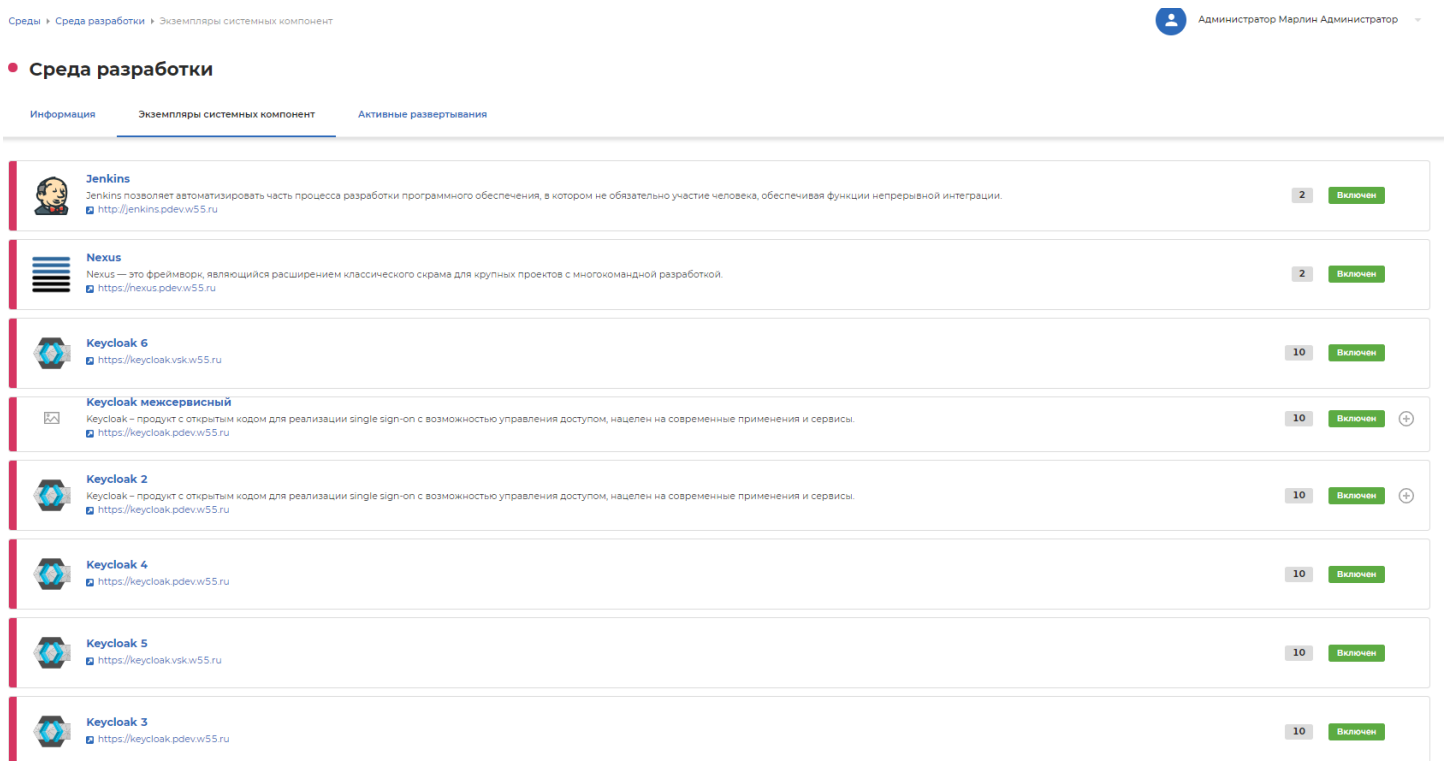
1. Перейдите на вкладку "Среды" в главном меню;



2. Выберите необходимую среду или создайте новую и нажмите на кнопку "Редактировать";



3. Перейдите в раздел "Экземпляр системных компонент";



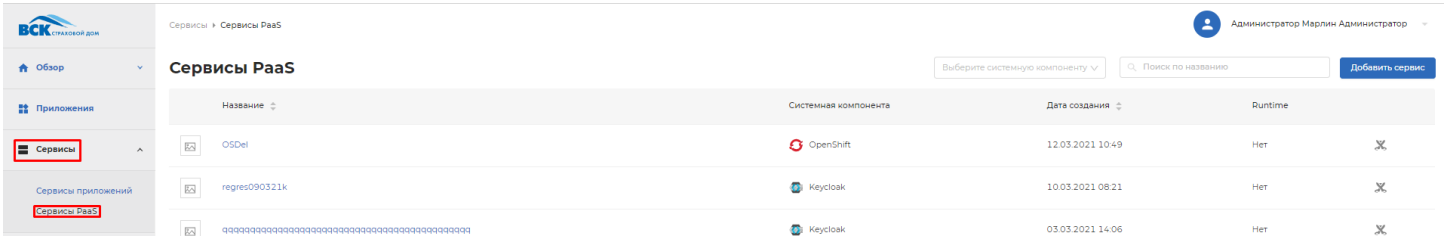
4. Выберите ЭСК и нажмите кнопку "Редактировать"

5. Заполните карточку ЭСК

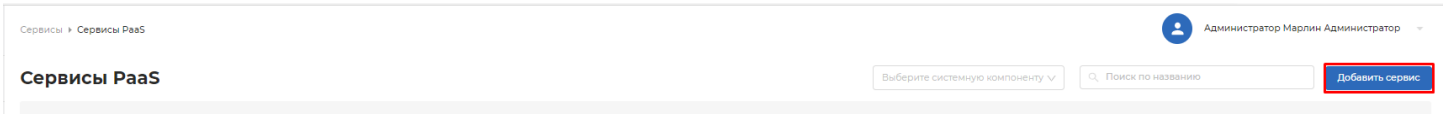
6. Повторите шаги 1-5 для всех ЭСК всех сред

Если вы добавили ЭСК с типом PaaS:

7. Перейдите в главном меню на вкладку "Сервисы", далее "Сервисы PaaS";



8. Нажмите на кнопку "Добавить сервис";



9. В открывшемся окне заполните поля, нажмите на кнопку "Сохранить";

10. Повторите пункты 7-9 для всех ЭСК с типом PaaS.

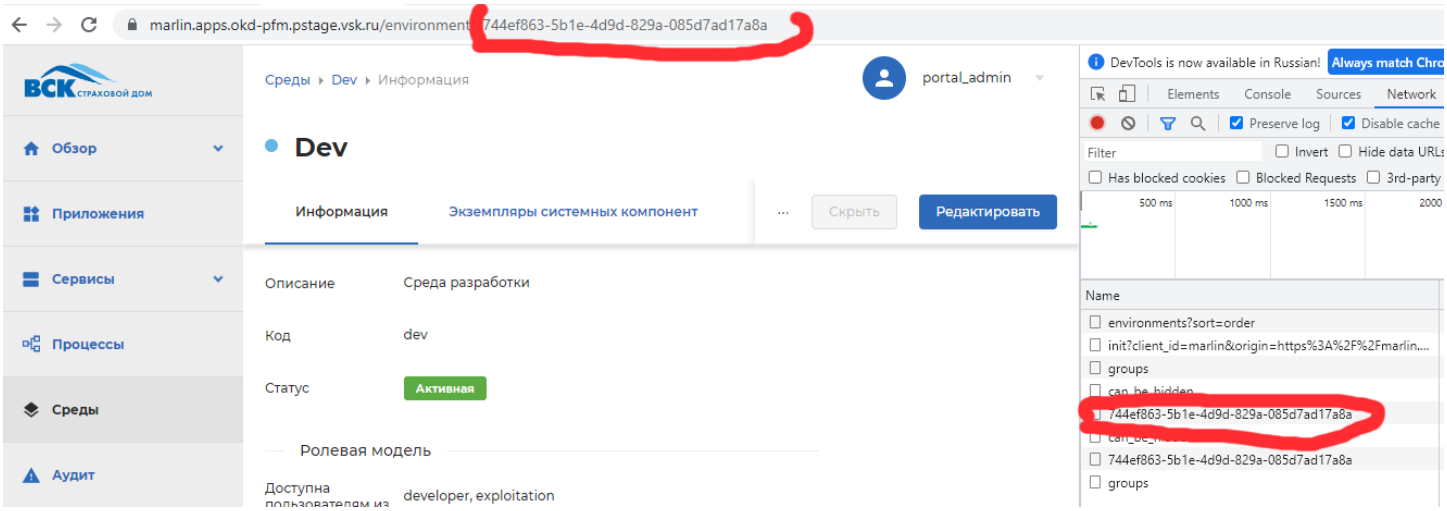
Настройка базы данных системных компонент

1. Подключиться к вновь созданной базе данных ПУ Marlin
2. Выполнить на ней SQL-запрос:

```
select * from environments;
```

3. Открыть в браузере ПУ Marlin, перейти в пункт меню "Среды".
4. По очереди подключаться к средам:
    - Dev
    - Test
    - Stage
    - Prod

5. Получить UUID сред в логах контейнера ПУ java-backend, backend, frontend или ng-frontend
6. Заменить uuid в таблице environments базы данных ПУ Marlin на полученные



7. Аналогичным образом, выясняя отсутствующие uuid из лога контейнеров приложения, поменять в базе данных соответствующие uuid.
8. Выполнить на базе данных команду:

```
INSERT INTO public.system_components
(id,title,description,characteristic,code,"version",properties,created_at,updated_at,
VALUES
     ('d60701f5-4828-4b13-a8c7-c1d90d41b1d1'::uuid,'OKD','OKD – community-версия Red
Kubernetes','paas','okd','4','[{"hint": "", "name": "url", "type": "url", "label": "U
[{"required": false}], "value": "http://okd.vsk.w55.ru", "secret": false, "placeholde
{"hint": "", "name": "project", "type": "string", "label": "Проект", "rules": [{"requ
"marlin", "secret": false, "placeholder": "Введите название проекта"}, {"hint": "Иден
значения, хранящегося в Hachicorp Vault", "name": "secret_id", "type": "string", "lab
секрета в Hachicorp Vault", "rules": [{"required": false}], "value": "/v1/marlin/paas
"placeholder": "Введите токен"}, {"hint": "Путь до Jenkinsfile содержащего определени
развертыванию PaaS OKD", "name": "deploy", "type": "string", "label": "Deploy", "rule
"value": null, "secret": false, "placeholder": "Введите путь до Jenkinsfile"}, {"hint
содержащего определение Jenkins Pipline по удалению развертывания PaaS OKD", "name":
"string", "label": "Undeploy", "rules": [{"required": true}], "value": null, "secret"
"Введите путь до Jenkinsfile"}, {"hint": "Протокол для формирования URL-адреса развер
"protocol", "type": "string", "label": "Протокол", "rules": [{"required": true}], "va
false, "placeholder": "Введите протокол"}, {"hint": "Префикс для формирования URL-адр
среде", "name": "prefix", "type": "string", "label": "Префикс", "rules": [{"required"
dev", "secret": false, "placeholder": "Введите префикс"}, {"hint": "Постфикс для форм
развертывания в среде", "name": "postfix", "type": "string", "label": "Постфикс", "ru
true}], "value": "apps.ose4.pdev.w55.ru", "secret": false, "placeholder": "Введите по
16:01:35.447837','2021-07-12 21:34:59.91679','d77da482-1b34-450a-9577-09d5f2dc6b70'::
```

```
SELECT id, title, description, characteristic, code, "version", properties,
created_at, updated_at, platform_service_kind_id
FROM public.system_components where code='okd';
```

2. Создание приложения

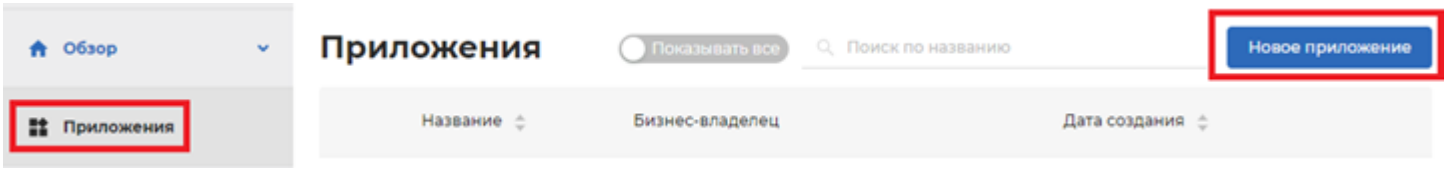
> ⓘ **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: Не требуются

Приложение является управленческой структурой, которая объединяет в себе несколько (микро) сервисов. Приложение необходимо для ведения реестра сервисов, выделенных для них ресурсов и управления командой с целью выделения доступа участникам команды к ресурсам приложения и входящих в его состав сервисов.

При создании приложения автоматически создаются необходимые объекты в системных компонентах Платформы. Ссылки на эти объекты отображаются в карточке приложения в ПУ.

Для создания приложения:

1. В главном меню перейдите на вкладку "Приложения" и в рабочем поле нажмите на кнопку "Новое приложение";

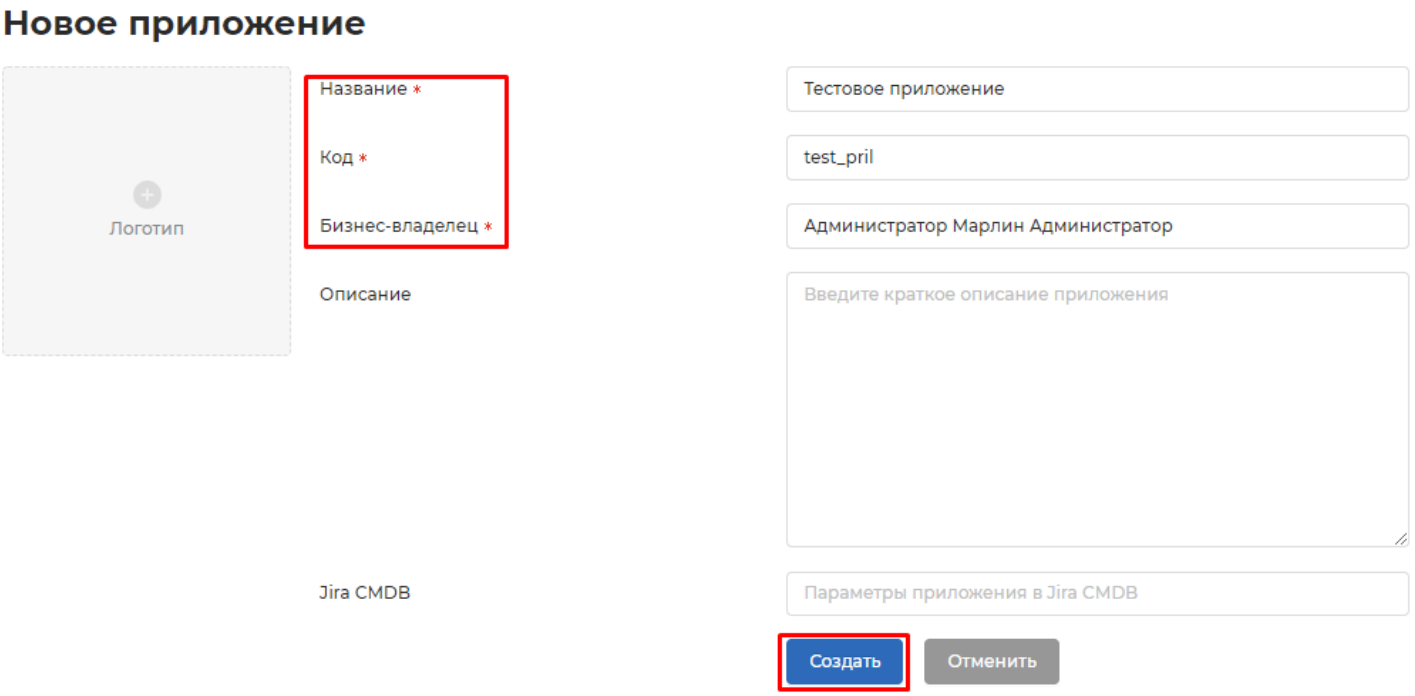


2. В открывшейся форме для ввода данных укажите следующие параметры для нового приложения:

   - **Название** - введите наименование приложения в соответствии с основными проводимыми работами в данном приложении. Название приложения можно вводить на русской или английской раскладке. Для ввода допускаются все символы и значения. В дальнейшем значение в поле может быть скорректировано;
   - **Код** - введите уникальный код приложения, данное значение будет использоваться в именовании связанных объектов в Платформе. Код

необходимо вводить исключительно на английской раскладке, он может состоять из букв и цифр в нижнем регистре, начинаться и заканчиваться на букву или цифру, из символов-разделителей разрешается использовать только нижнее подчеркивание. В дальнейшем значение в поле НЕ может быть скорректировано;

- **Бизнес-владелец** - укажите бизнес-владельца, который будет ответственен за приложение, его контактные данные будут отображаться в интерфейсе ПУ. В дальнейшем значение в поле НЕ может быть скорректировано;
- **Описание** - введите краткое описание приложения и его целевое назначение. В дальнейшем значение в поле может быть скорректировано.
- **Jira CMDB** - введите параметры приложения для поиска в Jira CMDB. В дальнейшем значение в поле может быть скорректировано;
- **Логотип** - выберите логотип приложения, файл в формате *.png,* .jpg. В дальнейшем изображение может быть скорректировано.



3. Нажмите на кнопку "Создать":

- в GitLab создаются группы одноименные коду приложения - ссылки на данные группы отображаются в карточке приложения на вкладке "Информация":
  - Репозиторий кода;
  - Репозиторий API;
- на созданные объекты выдаются права Бизнес-владельцу и, в дальнейшем, всем участникам команды приложения

4. Для редактирования приложения необходимо нажать на кнопку "Редактировать"- в режиме редактирования доступны все поля карточки приложения кроме:

    - Код
    - Бизнес-владелец

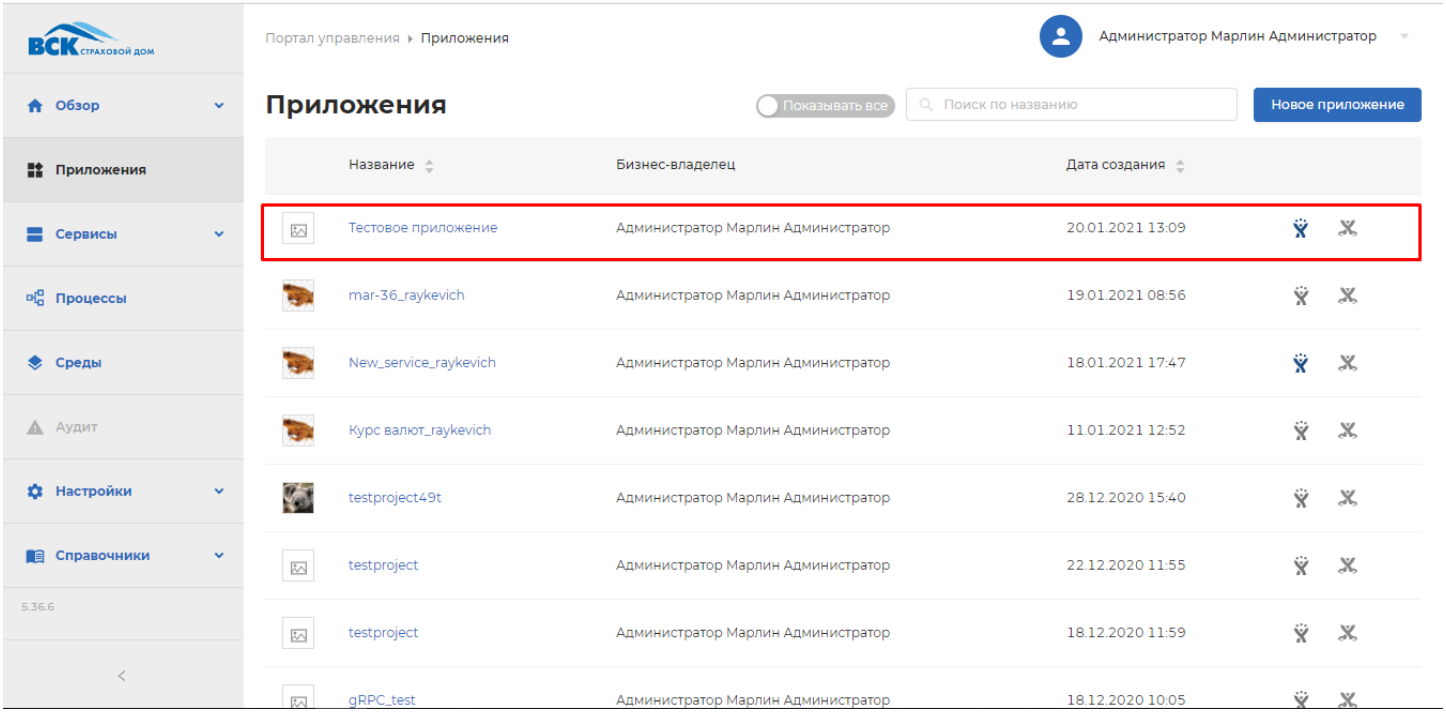
5. Перед созданием сервисов Приложения проверьте успешность создания объектов Платформы. Для этого перейдите в карточке приложения на вкладку "Настройки" и проверьте, что интеграция с системным
компонентами Платформы находится в статусе "Установлена". Если отображается статус "Не установлена", то нажмите
на кнопку "Перезапустить интеграцию". В случае неуспешного перезапуска интеграции убедитесь, что Экземпляр системной компоненты включен.

3. Добавление участника команды в приложение

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Администратор/Бизнес-владелец приложения/Участник команды приложения
>
> Предварительные действия: Создание приложения

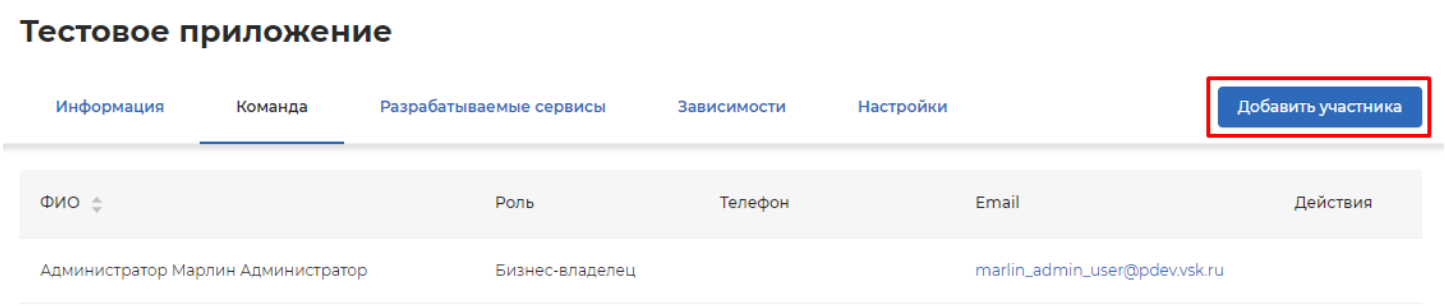
Для добавления участников в команду приложения:

1. В главном меню перейдите на вкладку "Приложения" и в рабочем поле выберите требуемое приложение;



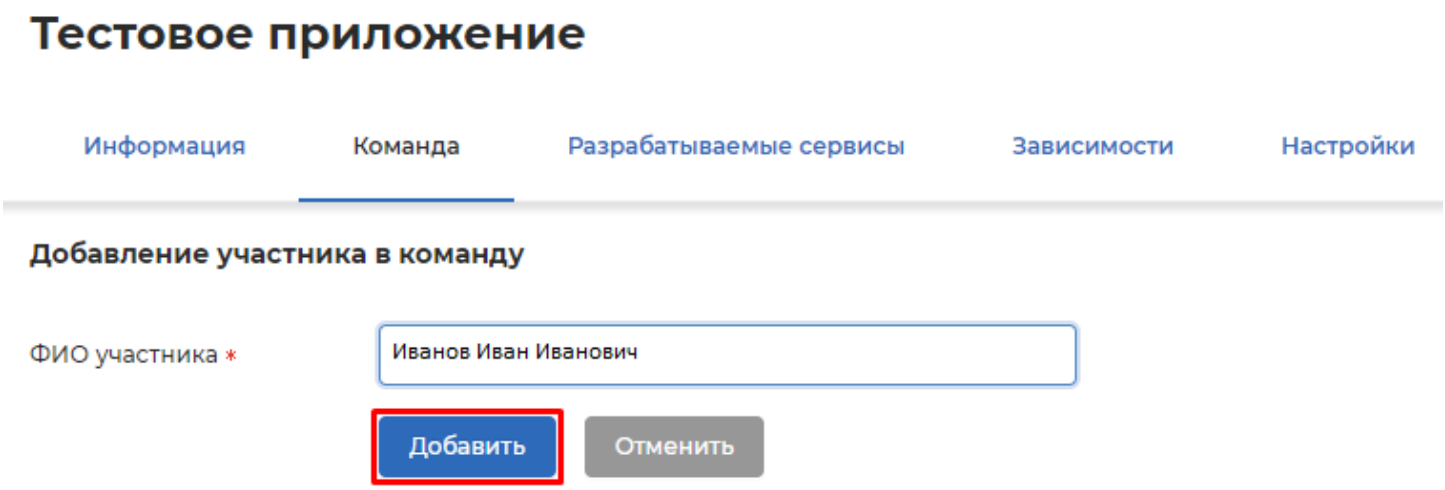
2. В карточке приложения перейдите на вкладку "Команда";

3. Нажмите на кнопку "Добавить участника";

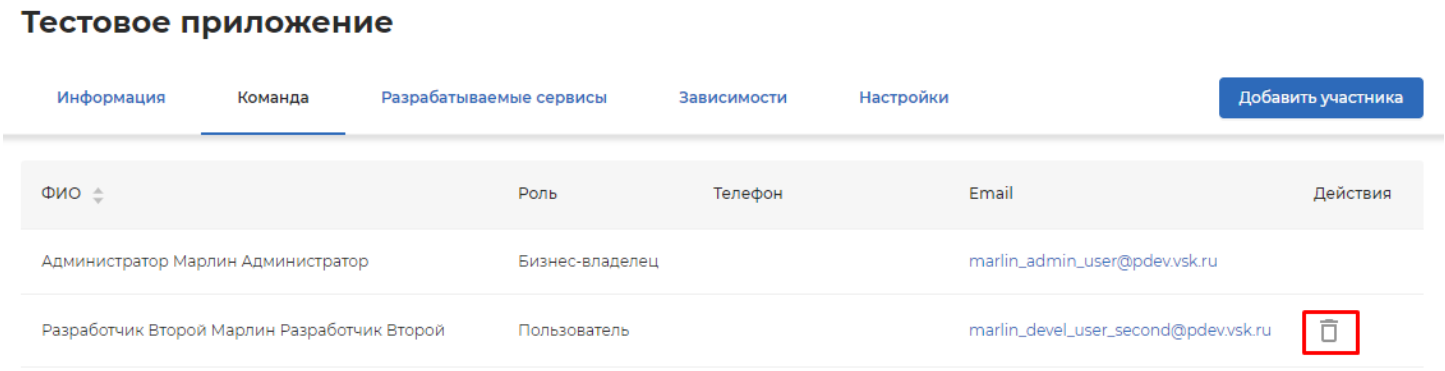


4. Введите ФИО участника или выберите пользователя из списка. Участники, которые состоят в команде не отображаются;

5. Нажмите на кнопку "Добавить";



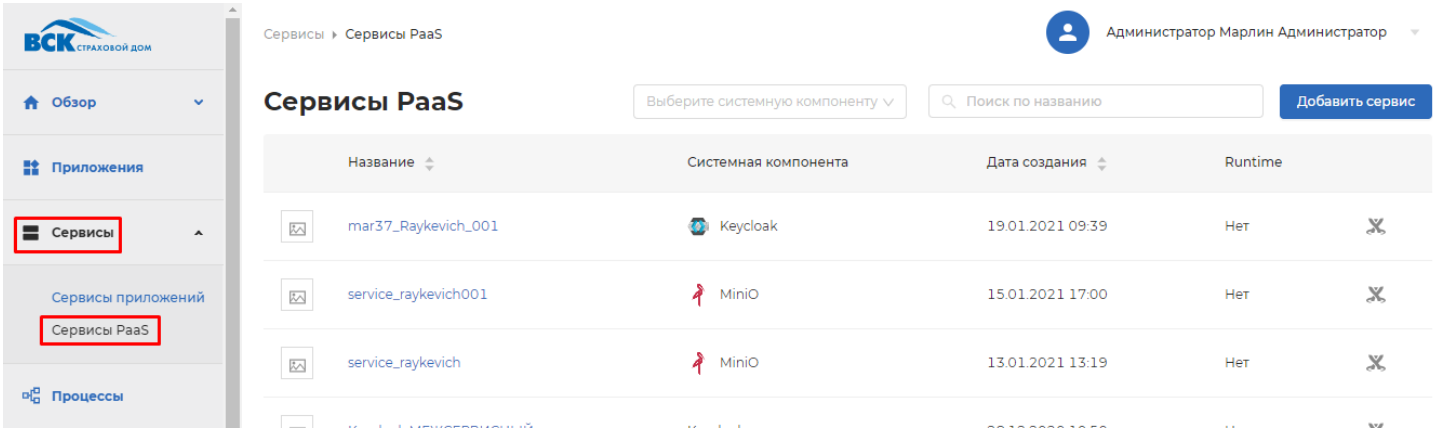
6. При необходимости исключить участника из команды, отозвать права у сотрудника на доступ к ресурсам приложения и его сервисов - нажмите на кнопку "Удалить".



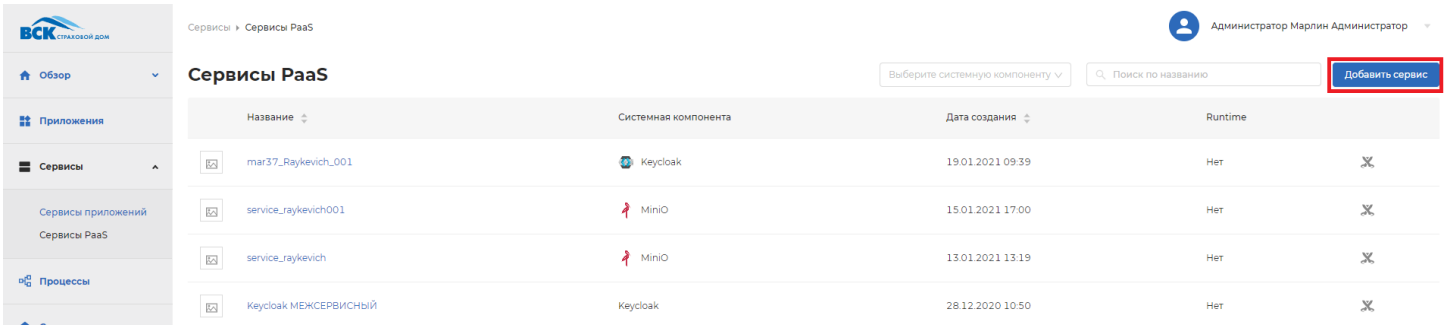
4. Создание Платформенного сервиса (PaaS)

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Администратор
>
> Предварительные действия: Не требуются

1. Перейдите в главном меню на вкладку "Сервисы" - "Сервисы PaaS";



2. Нажмите на кнопку "Добавить сервис";



3. В открывшейся форме для ввода данных укажите следующие параметры для создания сервиса:

   - **Название** - введите наименование сервиса в произвольном виде. В дальнейшем значение в поле может быть скорректировано.
   - **Код** - введите уникальный код сервиса, данное значение будет использоваться в связанных объектах. Код необходимо вводить исключительно на английской

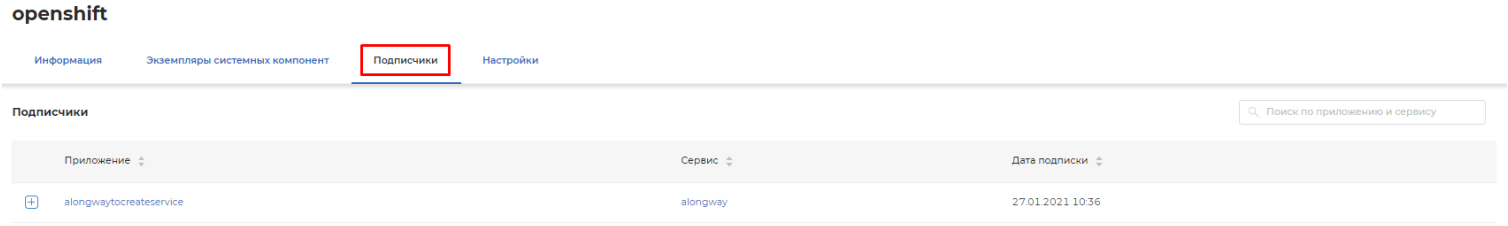
раскладке, может состоять из букв и цифр в нижнем регистре, начинаться и заканчиваться должен на букву, из символов-разделителей разрешается использовать только нижнее подчеркивание. В дальнейшем значение в поле НЕ может быть скорректировано.

- **Системная компонента (СК)** - выберите из выпадающего списка нужную системную компоненту, на основании которой будет создан PaaS-сервис.
- **Экземпляры системных компонент (ЭСК)** - поставьте галки рядом с нужной ЭСК на каждой среде. Если в рамках среды есть несколько ЭСК, то выбрать можно только одну.
- **Runtime** - можно создать только один Runtime-сервис на СК.
- **Описание** - рекомендуем добавить параметры, которыми в дальнейшем будет удобно воспользоваться при внесении их в исходный код в "chart"-"values" и в "chart"-"template"-"deployment". Описание платформенного сервиса выводится у прикладного сервиса при подписке на первый.
  - Для PaaS-сервиса Keycloak:
    - PAAS_CODE + "Url" - адрес обращения к экземпляру Keycloak, включая протокол.
    - PAAS_CODE + "Realm" - наименование realm
    - PAAS_CODE + "ClientId" - идентификатор клиента, выделенного для прикладного сервиса
    - PAAS_CODE + "ClientSecret" - секрет для доступа к клиенту
  - Для PaaS-сервиса PostgreSQL
    - PAAS_CODE + "Host" - адрес обращения к экземпляру PostgreSQL, включая протокол
    - PAAS_CODE + "Database" - имя базы данных
    - PAAS_CODE + "User" - имя пользователя
    - PAAS_CODE + "Password" - пароль
  - Для PaaS-сервиса MinIO
    - PAAS_CODE + "Url" - адрес обращения к экземпляру MinIO, включая протокол
    - PAAS_CODE + "Bucket" - наименование букета
    - PAAS_CODE + "AccessKey" - ключ доступа
    - PAAS_CODE + "SecretKey" - секрет

4. Нажмите на кнопку "Создать";

Для каждого сервиса PaaS реализовано ведение реестра сервисов-подписчиков:

5. Выберите необходимый сервис PaaS и перейдите на вкладку Подписчики.

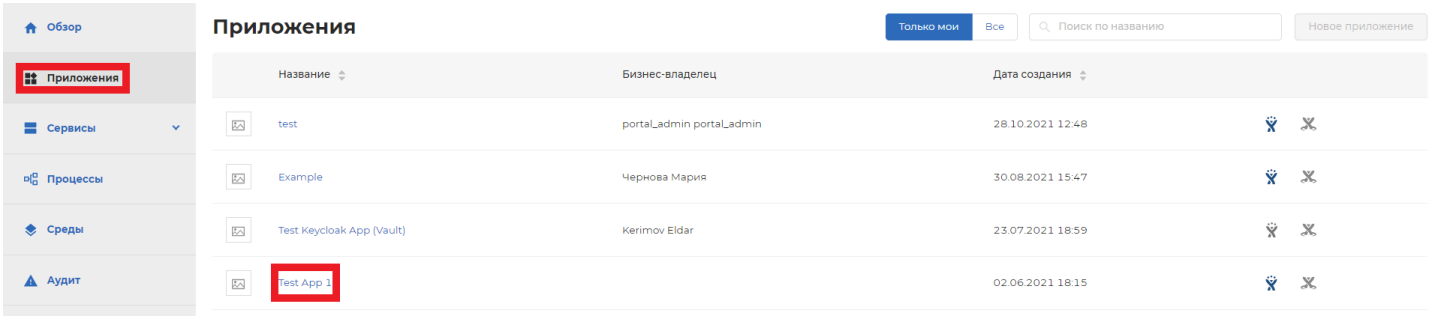


5. Очистка ресурсов

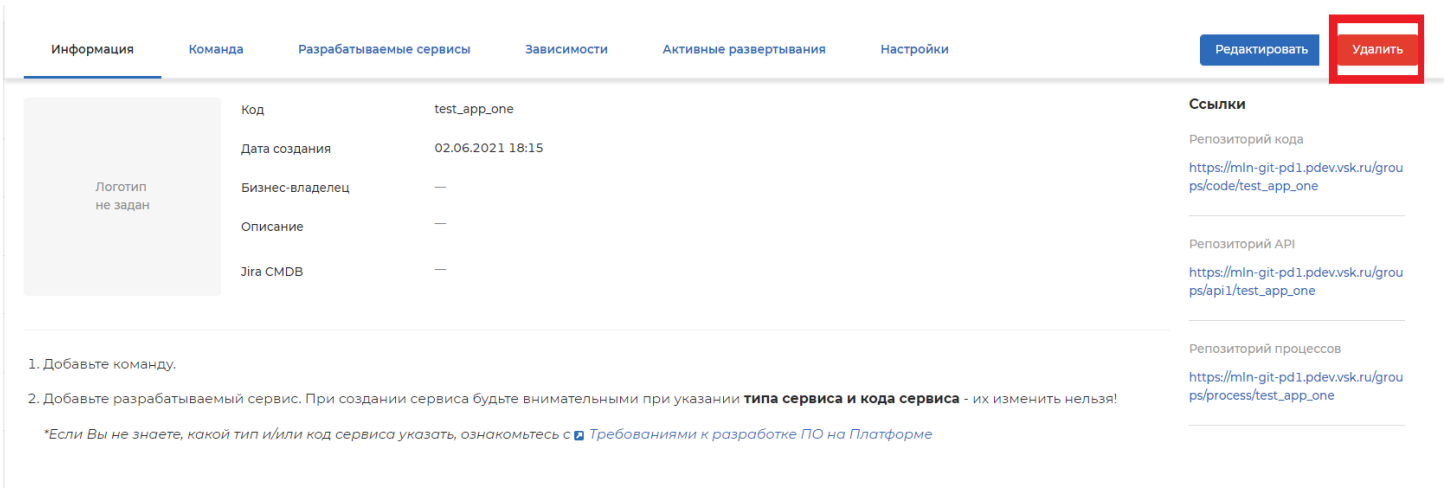
> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Администратор
>
> Предварительные действия: Удалены необходимые вложенные сущности

Для удаления приложения

1. Перейдите во кладку "Приложения" и выберите нужное приложение;



2. Нажмите "Удалить" в карточке приложения;



3. Нажмите на кнопку "Да" в диалоговом окне.

> **К СВЕДЕНИЮ**

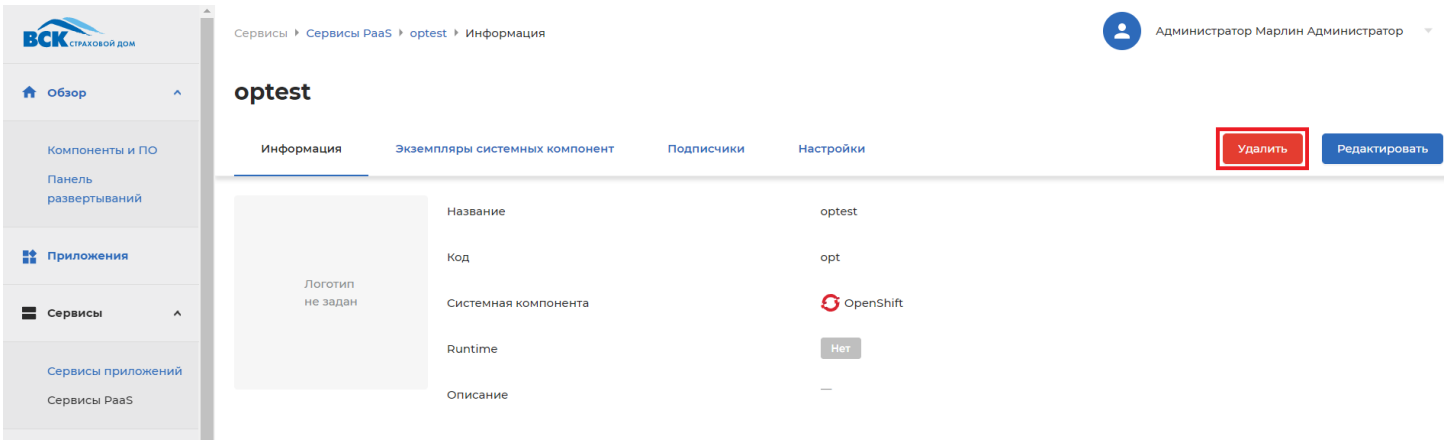
Карточка приложения не может быть удалена, если в сервисах есть неудаленные ресурсы

Для удаления PaaS-сервиса

1. Откройте в главном меню вкладку "Сервисы" - "Сервисы PaaS";

2. Перейдите в карточку PaaS-сервиса;

3. Нажмите кнопку "Удалить";



4. Нажмите на кнопку "Да" в диалоговом окне.

К СВЕДЕНИЮ

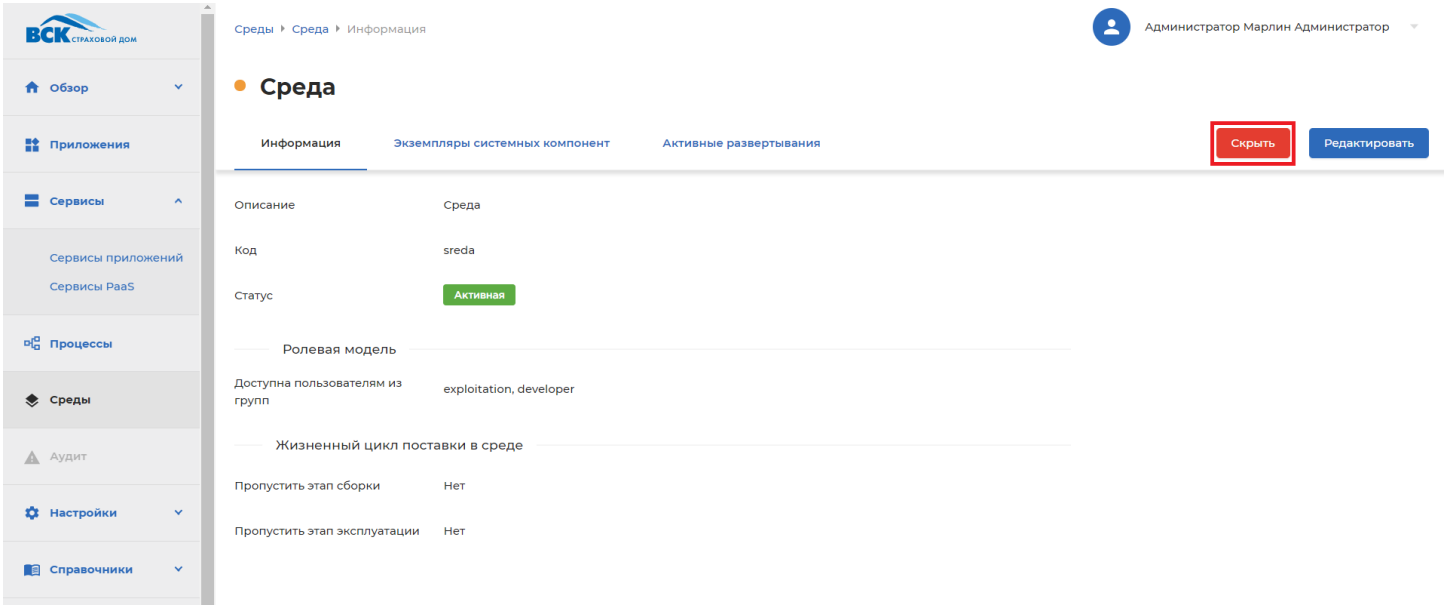
Если кнопка "Удалить" недоступна для нажатия, то следует удалить подписку прикладного сервиса (или прикладных сервисов) на этот PaaS-сервис. В карточке PaaS-сервиса на вкладке "Подписчики" отображаются все прикладные сервисы, которые используют данный PaaS-сервис.

Для скрытия сред

1. Откройте в главном меню вкладку "Среды";

2. Перейдите в карточку среды, которую необходимо скрыть;

3. Нажмите на кнопку "Скрыть";



4. Нажмите на кнопку "Да" в диалоговом окне.

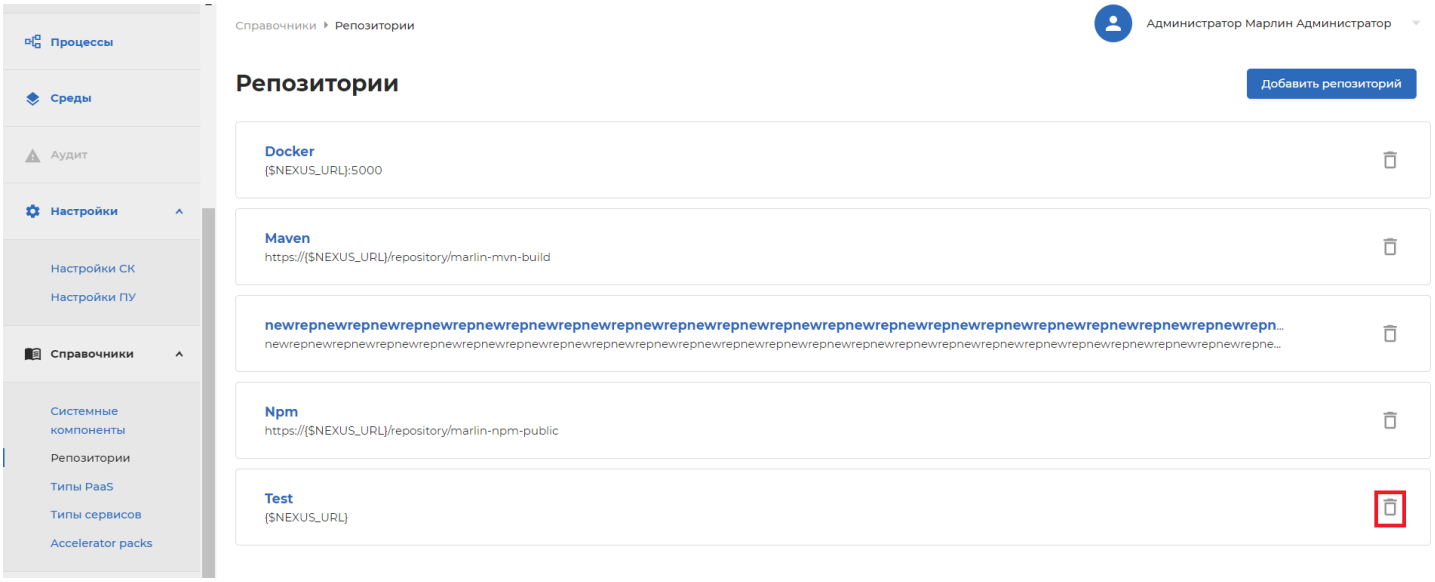
> **К СВЕДЕНИЮ**
>
> Если кнопка "Скрыть" недоступна для нажатия, то это означает, что на среде есть активные развертывания. Перейдите в карточке среды на вкладку "Активные развертывания" и удалите все развертывания.

Для удаления репозитория

1. Откройте в главном меню вкладку "Справочники" - "Репозитории";

2. Выберите из списка репозиторий, который необходимо удалить;

3. Нажмите на "Корзину";



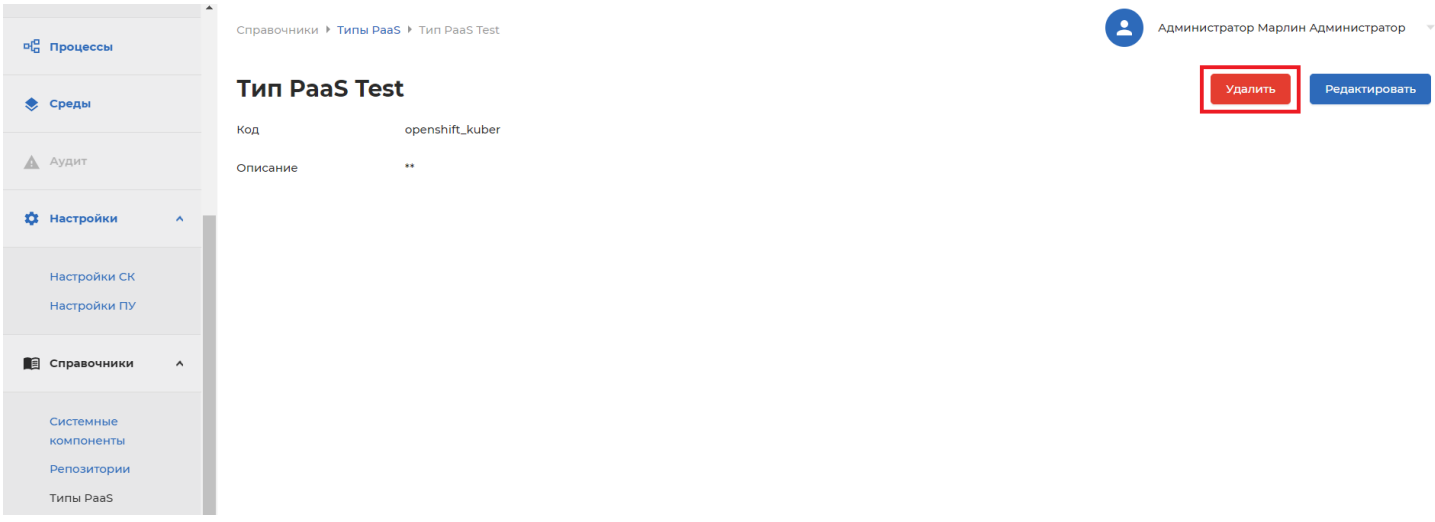
4. Нажмите на кнопку "Да" в диалоговом окне.

Для удаления типа PaaS

1. Откройте в главном меню вкладку "Справочники" - "Типы PaaS";

2. Откройте карточку типа PaaS, который необходимо удалить;

3. Нажмите на кнопку "Удалить";



4. Нажмите на кнопку "Да" в диалоговом окне.

Руководство пользователя

1. Назначение и область применения

Область применения

Платформа "Marlin" (далее - Платформа) предназначена для разработки прикладного программного обеспечения (далее - ПО, (микро)сервисов, объединенных в приложения) с предоставлением необходимых в ходе разработки, тестирования и эксплуатации вычислительных ресурсов по модели «платформа как сервис».

Платформа предназначена для автоматизации деятельности в области:

- сборки, тестирования и развертывания ПО
- регистрации приложений, сервисов, процессов
- реализации доступа разработчика к репозиториям исходного кода и библиотек
- выделения ресурсов для управления сервисами
- реализации зон разработки, тестирования и зоны постоянной эксплуатации Платформы
- реализации общих инфраструктурных сервисов (развертывание системных компонент, подготовка скриптов развертывания, оказание консультаций для развертывания)

Краткое описание возможностей

> **ⓘ ПРИМЕЧАНИЕ**
>
> Данное руководство предназначено для следующих ролей пользователей ПУ: Разработчик/Сотрудник эксплуатации

В соответствии с ролевой моделью ПУ для Разработчика доступны следующие действия в рамках сред DEV, TEST, STAGE:

- работа с выделенными репозиториями (исходного кода, API);
- анализ протоколов сборки и развертывания сервисов;
- проведение внутреннего тестирования сервисов;
- проведение функционального тестирования сервисов;

- управление следующими сущностями ПУ (доступ на просмотр, создание и редактирование):
    i. сервис;
    ii. развертывание сервиса;
    iii. сборка;
    iv. шаблон развертывания;
    v. поставка;
    vi. подписка;
    vii. развертывание сервиса PaaS.

В соответствии с ролевой моделью ПУ для Сотрудника эксплуатации доступны следующие действия в рамках сред DEV, TEST, STAGE, PROD:

- работа с выделенными репозиториями (исходного кода, API);
- анализ протоколов сборки и развертывания сервисов;
- проведение интеграционного тестирования сервисов;
- проведение контрольных мероприятий (на среде PROD);
- управление следующими сущностями ПУ (доступ на просмотр, создание и редактирование):
    i. сервис;
    ii. развертывание сервиса;
    iii. сборка;
    iv. шаблон развертывания;
    v. поставка;
    vi. подписка;
    vii. развертывание сервиса PaaS.

Уровень подготовки

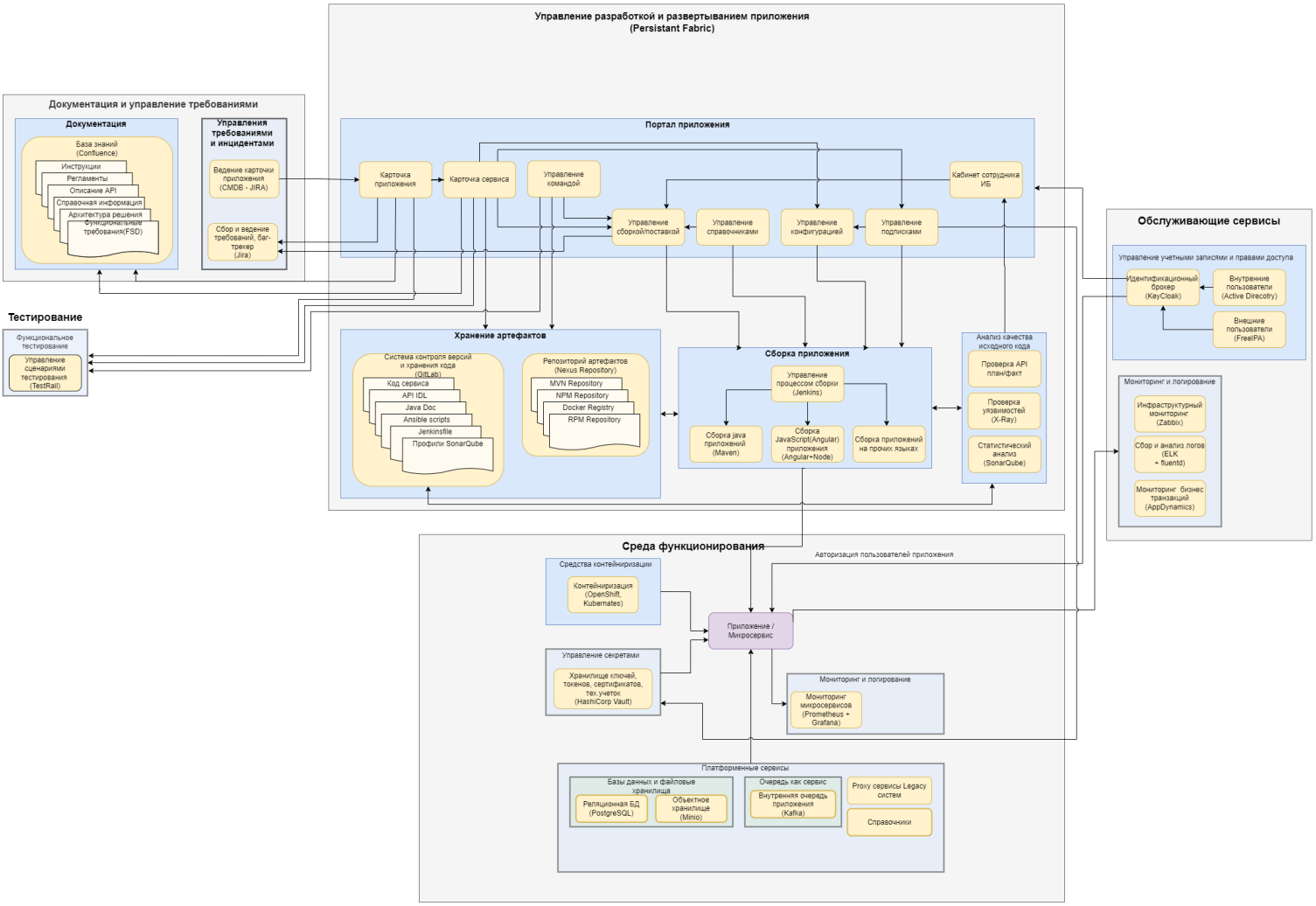
Разработчики приложений на Платформе Marlin должны обладать следующими компетенциями:

1. Владение стеком технологий, необходимых для разработки конкретного приложения (определяется при проектировании приложения). Типовые требования могут выглядеть следующим образом:

    i. Для AcceleratorPack Java/Maven: Опыт разработки на Java, знание фреймворка Spring, Spring Boot, опыт написания unit-тестов с использованием JUnit;
    ii. Для AcceleratorPack .NET: Опыт разработки на C#, знание фреймворка .NET, опыт написания unit-тестов с использованием NUnit;
    iii. Для AcceleratorPack NodeJS: Опыт разработки на Java Script, знание фреймворка ExpressJS, опыт написания unit-тестов с использованием Jest;
    iv. Для AcceleratorPack Frontend: Опыт разработки на Java Script, знание фреймворка Angular или React;
2. Опыт разработки API, обеспечивающих взаимодействие по следующим протоколам: gRPC, REST, Kafka messaging;
3. Опыт работы с СУБД (PostgreSQL, MongoDB, Redis);
4. Уверенное знание принципов работы систем контроля версий (Git);
5. Опыт работы с системами контейнеризации (Docker);
6. Базовые знания в области автоматизации процесса разработки (CI/CD);
7. Опыт использования agile-методологии разработки с использованием: code-review, merge-request, feature-branch, регулярная поставка релизов и т.п.

2. Краткое описание Marlin

Общая архитектура



Типы сервисов

На Платформе "Marlin" выделяется несколько типов сервисов:

Frontend - это сервисы, единственной целью которых является предоставление статики (JS, CSS, HTML, статичные картинки) клиентам. Тип сервиса включает в себя приложения на следующих AcceleratorPack'ax:

- ANGULAR;
- DOCKER.

BackForFront - это сервисы, обслуживающие пользовательские запросы, выполняемые из Frontend-сервисов. Тип сервиса включает в себя приложения на следующих AcceleratorPack'ax:

- Java11;
- Maven (JDK17);
- .NET;
- NGINX;
- NODEJS;
- DOCKER;
- PHP.

Backend - это сервисы, реализующие бизнес-логику приложения либо выполняющие вспомогательные функции, например, сервис-адаптер. Тип сервиса включает в себя приложения на следующих AcceleratorPack'ax:

- Java11;
- Maven (JDK17);
- .NET;
- NGINX;
- NODEJS;
- DOCKER;
- PHP.

External - сервисы, расположенные за пределами платформы, регистрируемые на платформе для реализации учётной функции

Общее описание жизненного цикла приложения

Жизненный цикл приложения Платформы состоит из следующих этапов:

- создание карточки нового приложения;
- добавление участников в команду;
- создание карточек одного или нескольких сервисов;
- разработка сервиса в зоне development и проведение внутреннего тестирования;

- передача версии ПО на среду testing и проведение функционального тестирования;
- перенос версии ПО на среду staging и проведение интеграционного или нагрузочного тестирования;
- проведение контрольных мероприятий и перенос приложения в production среду;
- ввод приложения в эксплуатацию.

В общих чертах, процесс разработки состоит из повторяющегося набора подпроцессов (спринт, итерация), которые обеспечивают реализацию исходных требований в автоматизированной системе:

```
1. Анализ требований и проектирование.

2. Реализация проекта в программном коде и документации приложения.

3. Проверка корректности реализации и соответствия требованиям.

4. Размещение приложения в продуктивной среде.

5. Анализ результатов.
```

Для разработки выбрана парадигма API First, которая определяет последовательность, в которой создаются публичные API системы, реализуются заявленные функции и происходит ввод системы в эксплуатацию. На этом этапе определяется компонентный состав системы, интерфейсы взаимодействия частей системы между собой и внешними зависимостями т.д.

Важно в процессе проектирования учитывать как возможности, так и ограничения программной Платформы «Marlin».

После процесса проектирования производится реализация спроектированных компонентов/функций, производится документирование, покрытие тестами и первоначальное тестирование разработанных компонентов/функций.

После реализации требований должен быть произведен контроль качества реализации. Контроль включает в себя как ручные, так и автоматизированные процедуры. К ручным могут относиться код-ревью, ручное тестирование и т.д. К автоматизированным операциям относятся — статический анализ, проверка стиля, юнит-тестирование, смоук-тестирование и т.д.

После успешного прохождения контроля качества, приложение готово к развертыванию в продуктивной среде.

Формирование роутов OpenShift

В ПУ роуты формируются следующим образом:

1. Протокол - протокол передачи данных.
2. Префикс - подтягивается из СК OpenShift в зависимости от того, на какой среде формируется роут (префиксы в СК OpenShift уникальны в рамках среды).
3. Код сервиса - автоматически подставляется код сервиса, есть возможность ввести вручную нужное значение.
4. Порядковый номер - номер, который позволяет поддерживать уникальность адреса в рамках среды.
5. Постфикс - подтягивается из СК OpenShift в зависимости от того, на какой среде формируется роут (постфиксы в СК OpenShift уникальны в рамках среды).

3. Инструкция по работе с Порталом Управления

1. Создание прикладного сервиса
2. Подписка на Платформенный сервис (PaaS)
3. Добавление учётной записи
4. Межсервисная подписка прикладных сервисов
5. Создание шаблона развертывания
6. Использование универсального helm-chart
7. Создание сборки
8. Создание автосборки
9. Настройка тестирования
10. Создание поставки
11. Развертывание прикладного сервиса
12. Авторазвертывание
13. Работа с разделом "Аудит"
14. Работа с feature-toggle
15. Очистка ресурсов
16. gRPC to JSON адаптер
17. Мониторинг
18. Использование PaaS Keycloak
19. Получение ClientID и ClientSecret
20. Helm-chart и секреты в Vault
21. Взаимодействие с PaaS PostgreSQL
22. Настройка probe check для приложений
23. Инструкция по передаче переменных окружения в скрипты развертывания
24. Квоты и Масштабирование
25. Интеграция с GitLab
26. []
27. Реализация требований

3.1 Создание прикладного сервиса

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: Создано приложение

Сервис ПУ - это микросервис, который является частью приложения. При создании сервиса автоматически создаются необходимые связанные объекты, набор которых в общем случае зависит от типа создаваемого сервиса. Ссылки на все связанные объекты отображаются в карточке сервиса в ПУ. В первой очереди Платформы в ПУ доступны следующие типы сервисов:

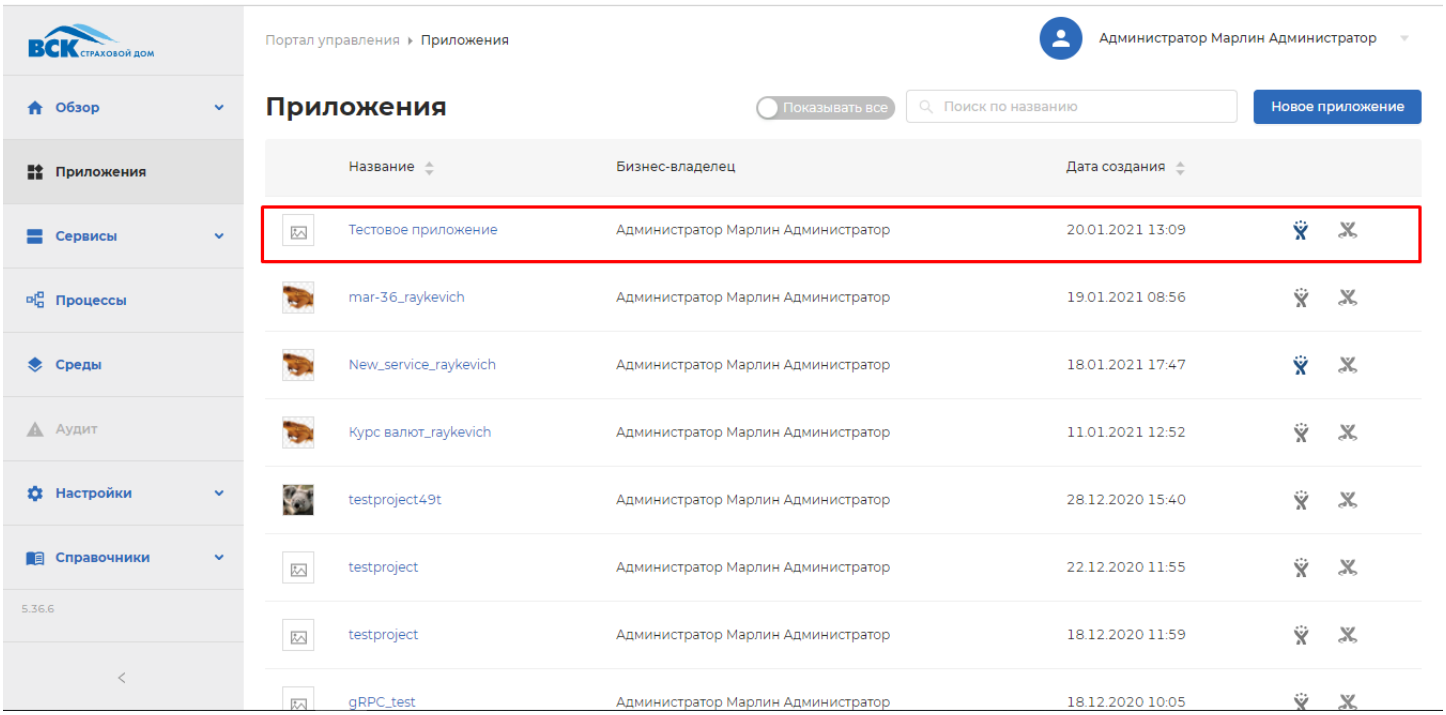
1. **External (legacy)** - для сервиса недоступны подписки на другие сервисы, добавление развертываний и создание областей для хранения исходного кода, процессов. Переменные окружения сервиса с типом Legacy передаются в шаблон развертывания сервиса-подписчика и настраиваются через интерфейс ПУ;
2. **Front** - для сервиса доступны подписки на следующие платформенные сервисы: OpenShift и Keycloak, и подписки на прикладной сервис с типом Backend, Frontend;
3. **Backend** - для сервиса доступны подписки как на платформенные, так и на прикладные сервисы, добавление развертываний и создание областей для хранения исходного кода, процессов;
4. **Kafka** - для сервиса недоступна подписка на другие прикладные сервисы, доступна подписка только на PaaS Kafka (является средой исполнения для сервиса с данным типом), доступно создание сборки, поставки и развертывания и создание репозитория API для хранения версий IDL (начиная с версии ПУ 5.38). Подробнее о работе с прикладным сервисом типа Kafka см. Решение по межсервисной авторизации в Kafka.

> **К СВЕДЕНИЮ**
>
> Перед созданием сервиса проверьте успешность интеграции приложения с системными компонентами Платформы. Для этого необходимо в карточке приложения перейти на вкладку "Настройки" и проверить, что интеграции

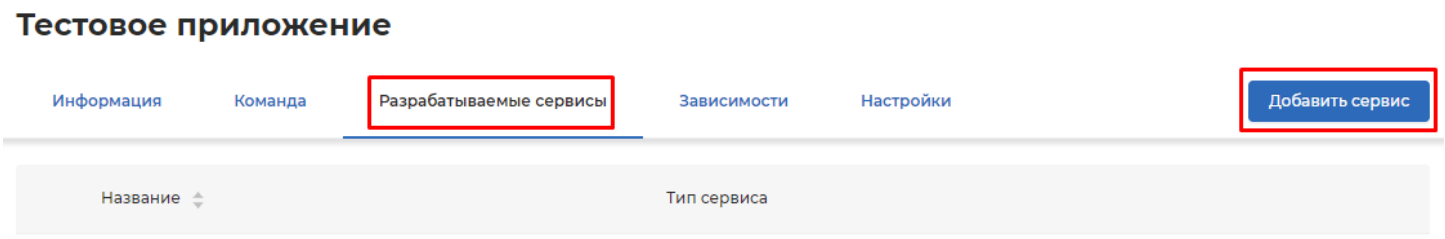
находятся в статусе "Установлена". Если отображается статус "Не установлена", то нажмите на кнопку "Перезапустить интеграцию"

Для создания сервиса

1. Перейдите в главном меню на вкладку "Приложения" и в рабочем поле выберите нужную карточку приложения.



2. В открывшейся карточке приложения перейдите на вкладку "Разрабатываемые сервисы" и нажмите на кнопку "Добавить сервис".

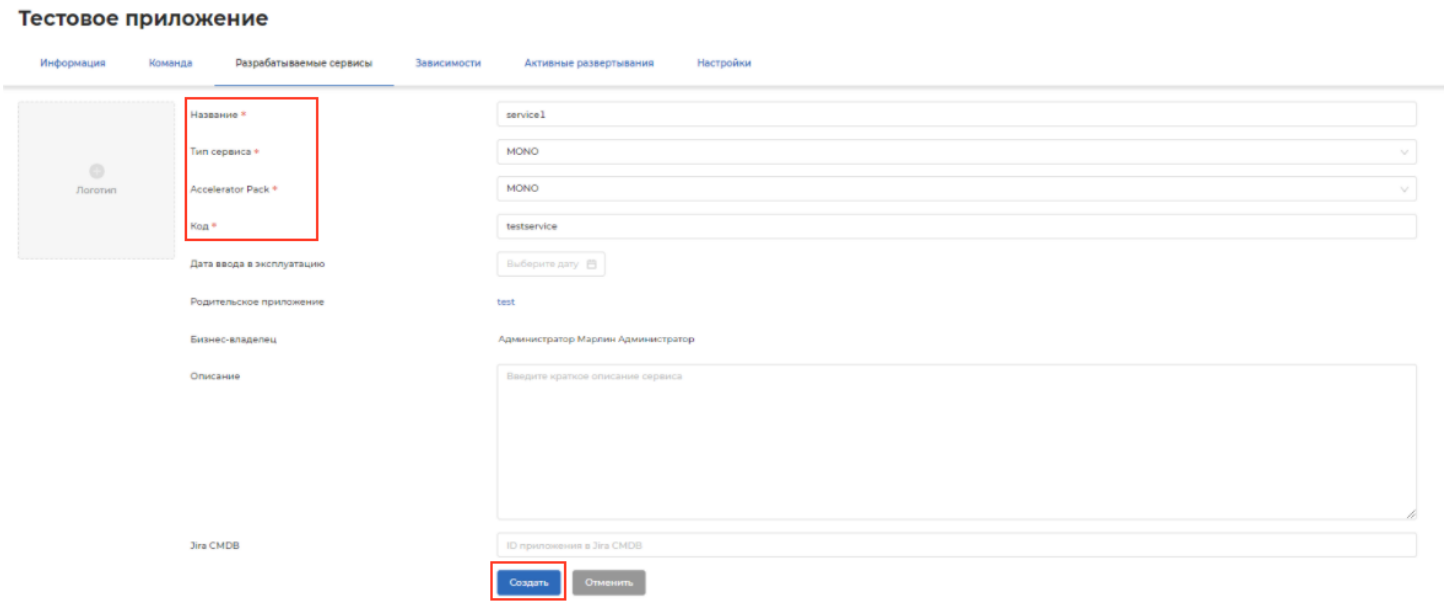


3. В открывшейся форме для ввода данных укажите следующие параметры для создания сервиса:

   - Название* - введите наименование сервиса в произвольном виде. В дальнейшем значение в поле может быть скорректировано.

- Тип сервиса* - из выпадающего списка выберите один из типов "External", "Front", "Backend" или "Kafka".
- Accelerator Pack* - выберите Accelerator Pack (пайплайны для сборки и развертывания сервиса) из выпадающего списка. В дальнейшем значение в поле не может быть скорректировано.
- Код* - введите уникальный код сервиса, данное значение будет использоваться в связанных объектах. Код необходимо вводить исключительно на английской раскладке, может состоять из букв и цифр в нижнем регистре, начинаться и заканчиваться на букву или цифру, из символов-разделителей разрешается использовать только нижнее подчеркивание. В дальнейшем значение в поле не может быть скорректировано.
- Описание - ведите краткое описание создаваемого сервиса и его целевое назначение. В дальнейшем значение в поле может быть скорректировано.
- Дата ввода в эксплуатацию. В дальнейшем значение в поле может быть скорректировано.
- Jira CMDB - введите ID сервиса в Jira CMDB.
- Логотип - выберите логотип сервиса, файл в формате .png, .jpg. В дальнейшем изображение может быть скорректировано.

Звездочкой отмечены обязательные параметры сервиса



4. Нажмите на кнопку "Создать".

5. Портал управления регистрирует сервис, создает связанные объекты:

- **Для сервиса типа "External (legacy)":** GitLab создает группы одноименные коду сервиса - ссылки на данные группы отображаются в карточке сервиса на вкладке "Информация":
  - Репозиторий API - ссылка на добавление IDL-файлов для дальнейших проверок сервиса; Artifactory
- **Для др. типов сервисов:** GitLab создает группы одноименные коду сервиса - ссылки на данные группы отображаются в карточке сервиса на вкладке "Информация":
  - Репозиторий API - ссылка на добавление IDL-файлов для дальнейших проверок сервиса;
  - Репозиторий кода Jenkins создает задачи на развертывание и сборку (создаются job). Artifactory

6. ПУ предоставляет доступ ко всей информации и структуре сервиса команде приложения.

7. При нажатии на кнопку "Редактировать" - доступны все поля для редактирования, кроме:

   - Тип сервиса;
   - Accelerator Pack
   - Код;
   - Дата создания
   - Родительское приложение;
   - Бизнес-владелец.

3.2 Подписка на платформенный сервис (PaaS)

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: Создано приложение, создан прикладной сервис, создан платформенный сервис (PaaS) Администратором

При создании подписки выбранный сервис-подписка добавляется в раздел "Зависимости" карточки сервиса-подписчика и автоматически - сервис-подписчик должен добавляться в соответствующий раздел карточки сервиса-подписки. В карточках приложений ПУ автоматически отображаются подписки, добавленные сервисом, и подписчики, которые добавили подписку на соответствующий сервис.

Для всех подписок параметры развертывания сервиса-подписки передаются в шаблон развертывания сервиса-подписчика. Для этого необходимо указать в шаблоне развертывания прикладного сервиса развертывание PaaS-сервиса (см. Создание шаблона развертывания).

При подписке на платформенный сервис Keycloak, OpenShift, MinIO, PostgreSQL выделяются следующие ресурсы:

1. **Keycloak**

   Keycloak - средство управления аутентификацией и идентификацией пользователей, а также авторизацией в сервисе, как пользователей, так и других сервисов:

   i. при подписке на платформенный сервис Keycloak и дальнейшем развертывании, jenkins вызывает соответствующее API пролить клиента, его роли и маппинг на группы (файлы для проливки ролей выкладываются в каталог в Git заранее);
   ii. подписка осуществляется на пользовательский или межсервисный Keycloak;
   iii. Для получения ClientID и ClientSecret см. Получение ClientID и ClientSecret

2. **OpenShift**

   OpenShift - среда исполнения прикладных сервисов, представляющаяся им как платформенный сервис:

   i. при подписке на платформенный сервис OpenShift и дальнейшем развертывании, заказывает ресурсы для приложения.

3. **MinIO**

   MinIO - открытый сервер хранения объектов, который может хранить неструктурированные данные:

   i. при подписке на платформенный сервис MinIO и дальнейшем развертывании, MinIO предоставляет место хранилища ("BUCKET")

4. **PostgreSQL**

   PostgreSQL - сервис баз данных для хранения данных компонентов Платформы:

   i. Подписка создается в БД, где выделяется место, в которой будет работать PaaS;
   ii. При подписке на платформенный сервис (PaaS), ресурсы выделяются в одном месте, параметры которых фиксируются в Портале Управления. Ресурсы автоматически учитываются при развертывании. При развертывании БД выделяются необходимые файлы в которые происходит запись данных;
   iii. Подписка на платформенный сервис осуществляется в рамках сервиса и шаблона развертывания. Подписка PaaS в рамках версия-версия не осуществляется. При изменении версии сервиса, создается новая версия PaaS и подписка осуществляется на новую версию сервиса.

5. **Redis**

   Redis - NoSQL СУБД для реализации механизмов кэширования данных:

   i. В отличии от других PaaS, Redis запускается вместе с приложением при его развертывании;
   ii. Запуск происходит также в кластере OKD, чтобы добиться максимальной производительности сокращением сетевой дистанции;
   iii. Redis запускается индивидуальной копией для каждого сервиса;

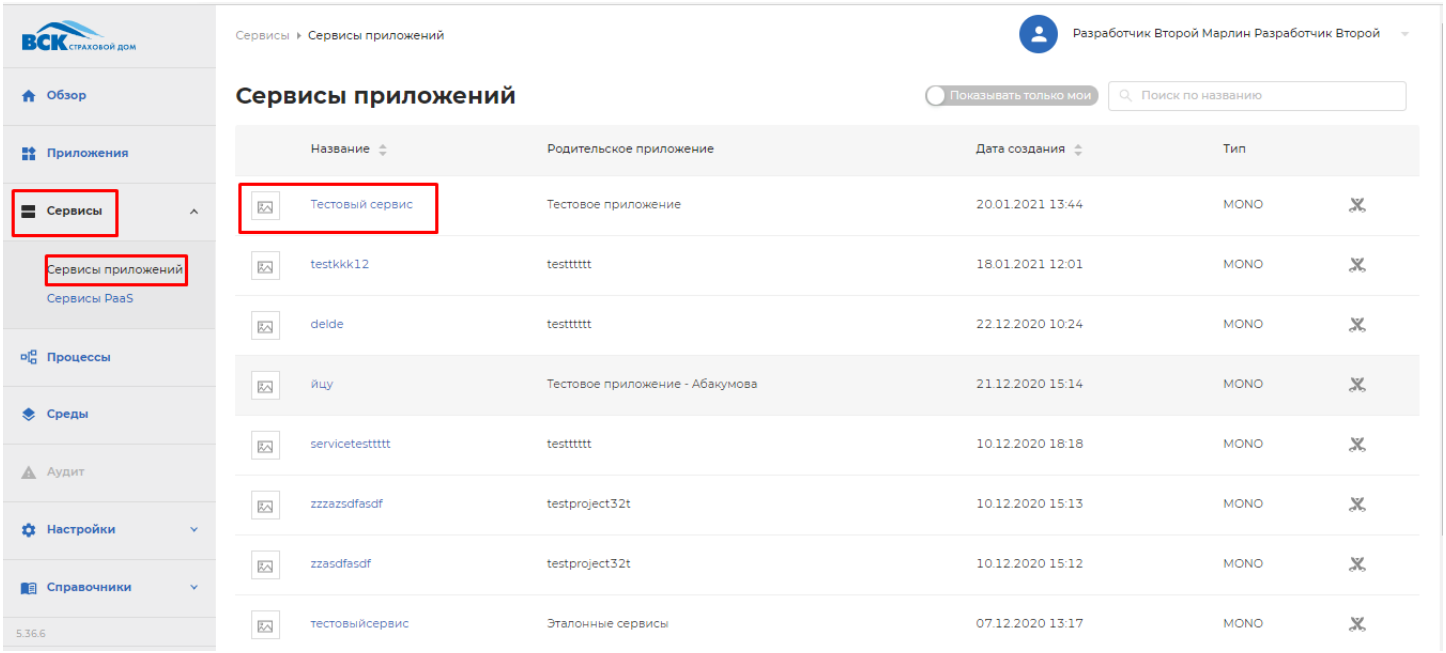
iv. Redis не имеет постоянного хранилища, перезапуск сервиса через удаление приведёт к запуску новой, пустой копии Redis.

!!! note ""

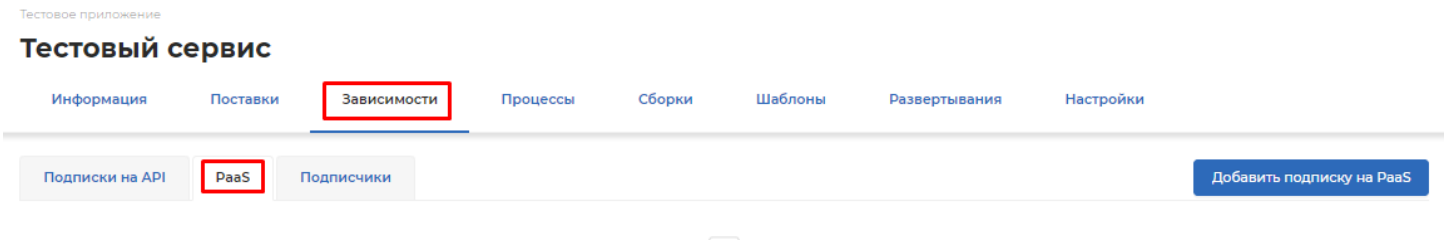
```
**Примечание!** Подписка на PaaS с типом "Среда исполнения" является обязательным условием для работы всех сервисов ПУ.
```

Для осуществления подписки на платформенный сервис:

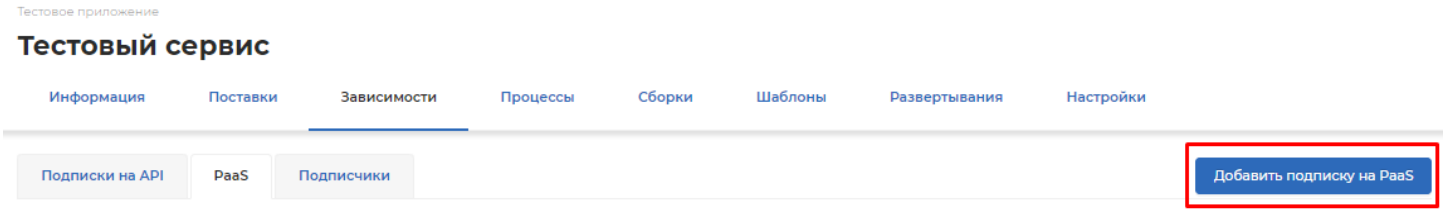
1. Перейдите в главном меню на вкладку "Сервисы приложений" и выберите требуемый сервис.



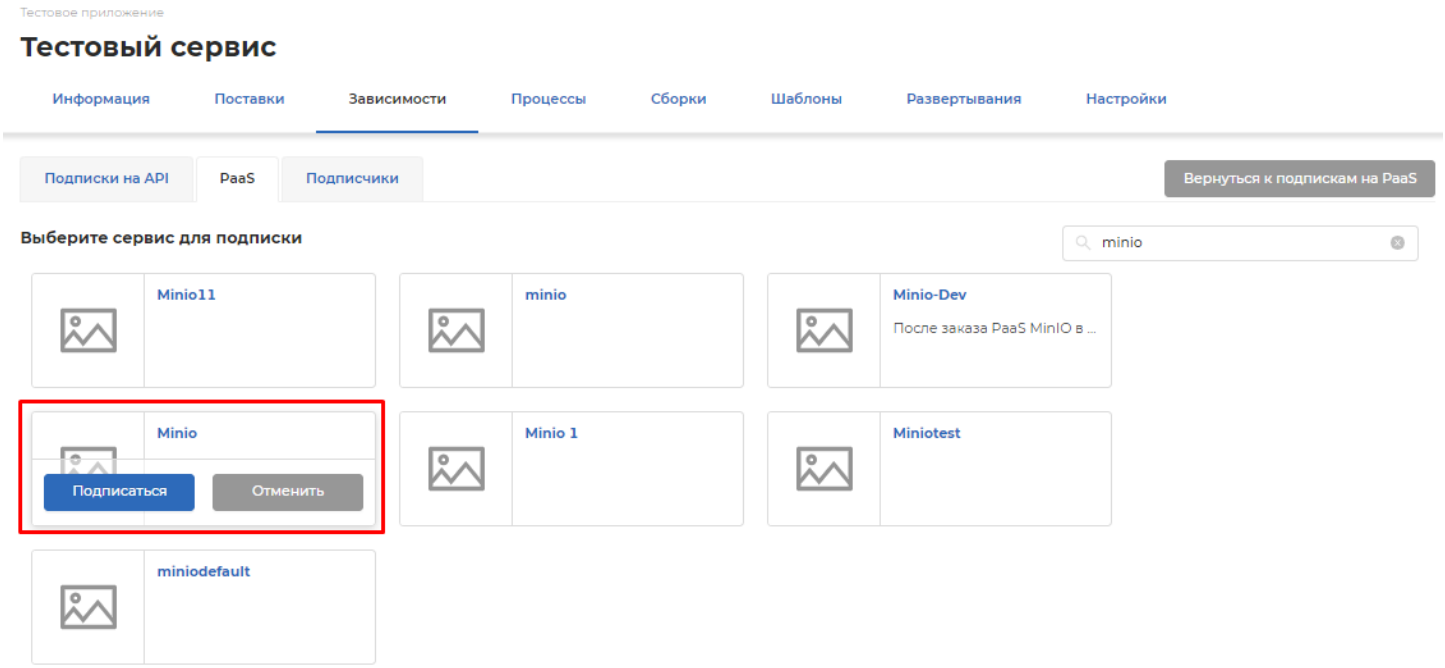
2. Перейдите на вкладку "Зависимости" и нажмите на подвкладку "PaaS".



3. Нажмите на кнопку "Добавить подписку на PaaS".

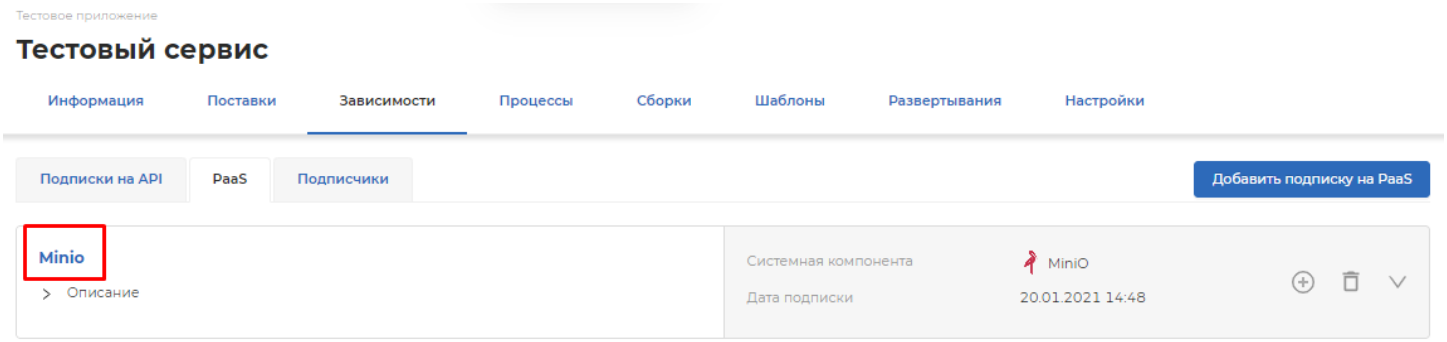


4. Выберите необходимый сервис, далее нажмите на логотип выбранного сервиса и на кнопку "Подписаться".

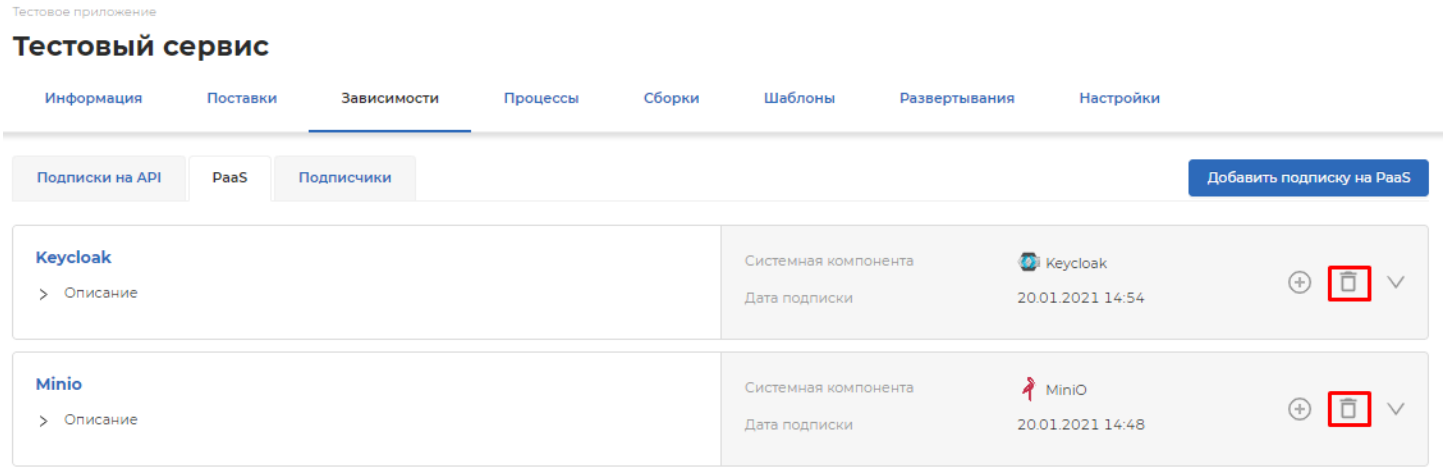


> **К СВЕДЕНИЮ**
>
> Список PaaS-сервисов, доступных для создания подписки, определяется в соответствии с параметрами типа сервиса в справочнике типов сервисов (выводятся только те PaaS-сервисы, напротив которых в справочнике для типа сервиса проставлены галочки).

5. Сервис-подписки успешно добавится в зависимости.



6. При необходимости удалить сервис-подписки - нажмите на кнопку "Удалить" в блоке платформенного сервиса.

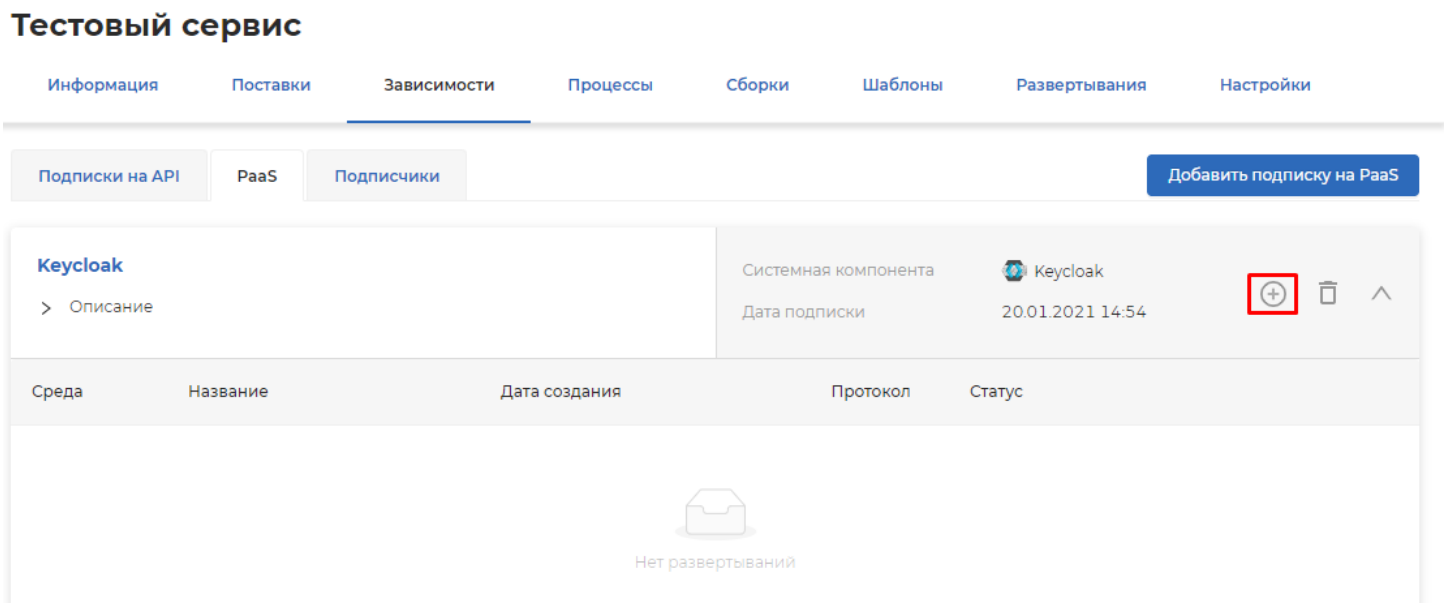


К СВЕДЕНИЮ

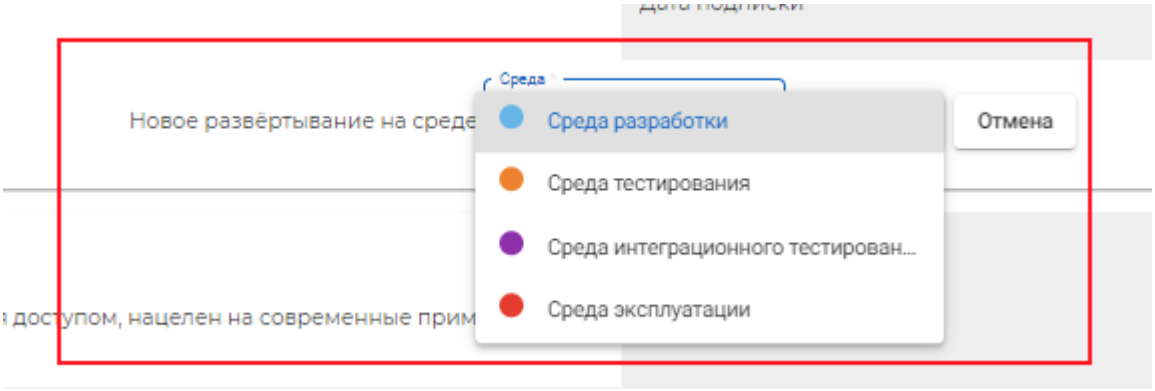
При добавлении подписки на платформенный сервис (PaaS) параметры его развертывания передаются в шаблон развертывания сервиса-подписчика, при этом возможность добавления развертывания для сервиса-подписчика контролируется наличием/отсутствием развертывания платформенного сервиса-подписки на соответствующей среде. Для добавления развертывания сервиса, имеющего подписку на PaaS, необходимо добавить развертывание PaaS для среды, в которой должен быть развернут сервис.

Для создания развертывания PaaS:

1. В блоке сервиса-подписки нажмите на кнопку "Добавить развертывание";



2. В открывшейся форме для ввода данных необходимо указать среду для создания развертывания платформенного сервиса (среда разработки, среда тестирования, среда интеграционного тестирования - доступны для разработчика/сотрудника эксплуатации, среда эксплуатации - доступна для сотрудника эксплуатации)



3. Нажмите на кнопку "Добавить"

В процессе развертывания выбранного PaaS осуществляется выделение ресурсов. После успешного развертывания PaaS ПУ выведет статус "Развернуто".

> **К СВЕДЕНИЮ**
>
> Для получения ClientID и ClientSecret см. Получение ClientID и ClientSecret

Возможность подписки на прикладные сервисы

Тип прикладного сервиса	Платформенные сервисы, доступные для подписки
Backend	1. OpenShift 2. Keycloak 3. MinIO 4. PostgreSQL 5. Redis 6. Kubernetes
Frontend	1. OpenShift 2. Keycloak 3. MinIO 4. PostgreSQL 5. Redis 6. Kubernetes
Kafka	1. Kafka
External (Legacy)	1. Keycloak 2. MinIO 3. PostgreSQL

К СВЕДЕНИЮ

```
В конкретной реализации может быть настроено иначе
```

3.3 Добавление учётной записи

> **ⓘ ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: Создано приложение, создан прикладной сервис, создан платформенный сервис (PaaS) Администратором, осуществлена подписка на PaaS

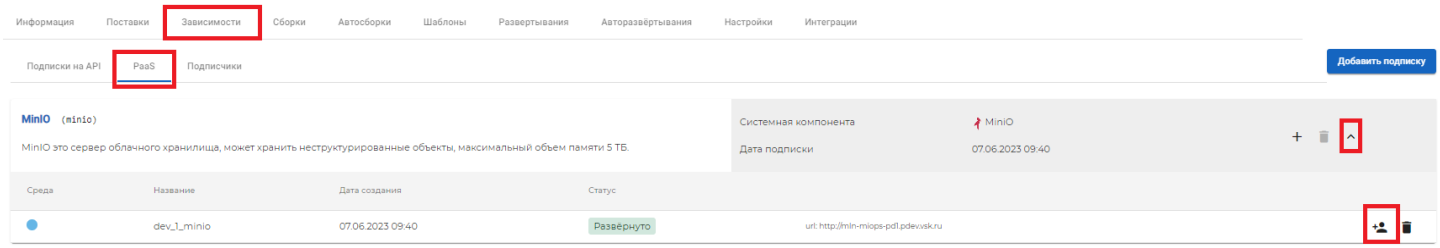
1. Перейти в сервис приложения;

2. Подписаться на PaaS;

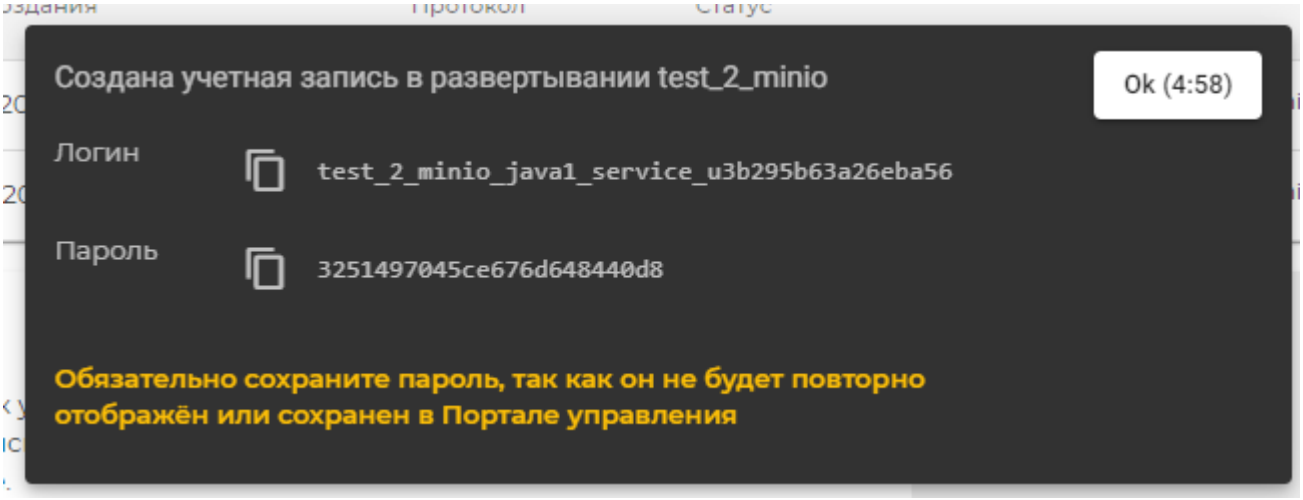
3. Запустить развёртывание на среде dev;

4. Открыть список всех развёртываний;

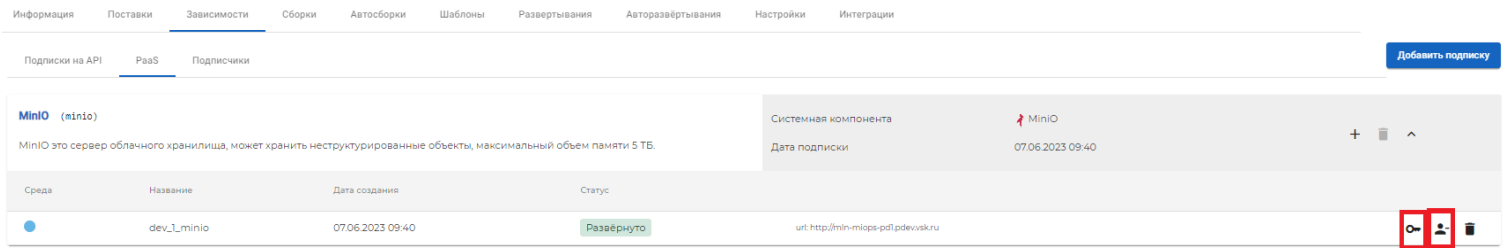
5. Нажать на кнопку "Создать учётную запись";



> **ⓘ К СВЕДЕНИЮ**
>
> Сохраните логин и пароль от учетной записи. Это можнео сделать только один раз.



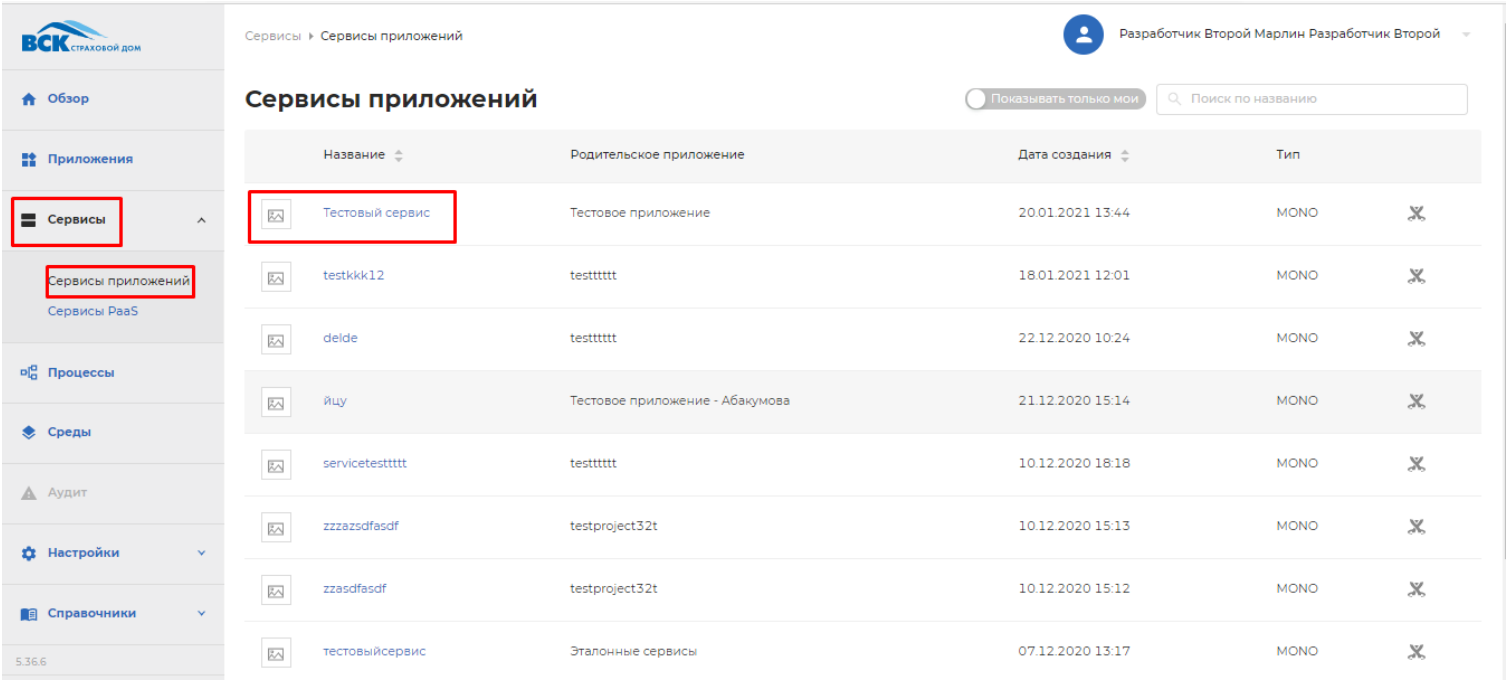
Чтобы **обновить пароль** или **удалить учетную запись** необходимо нажать на соответствующие кнопки:



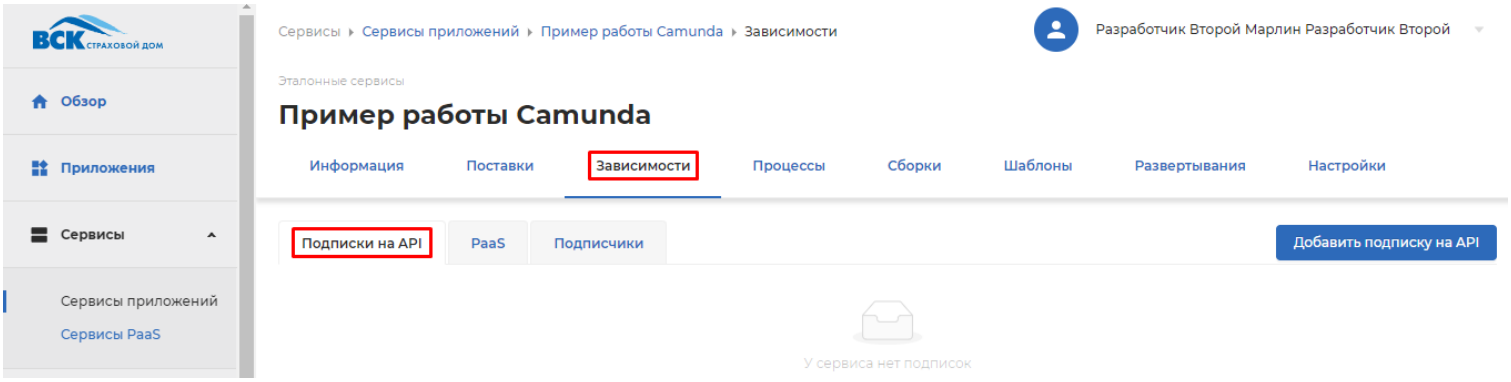
3.4 Межсервисная подписка прикладных сервисов

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: создан прикладной сервис-подписчик (например, сервис "gRPC Client"), создан прикладной сервис-подписка (например, сервис "gRPC Server")

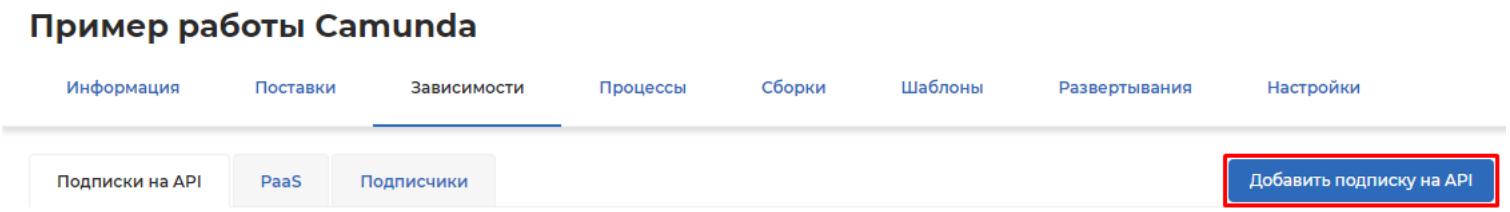
1. Перейдите в главном меню на вкладку "Сервисы приложений" и выберите сервис - подписчик, например gRPC Client;



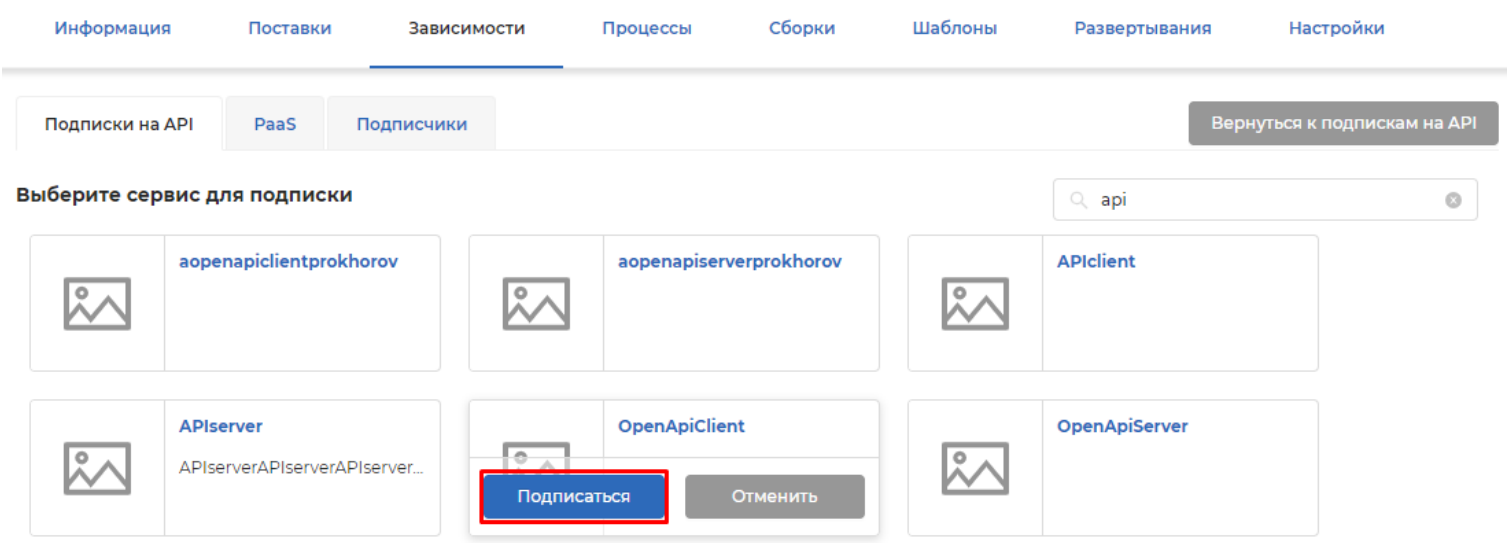
2. Перейдите на вкладку "Зависимости", на подвкладку "Подписки на API";



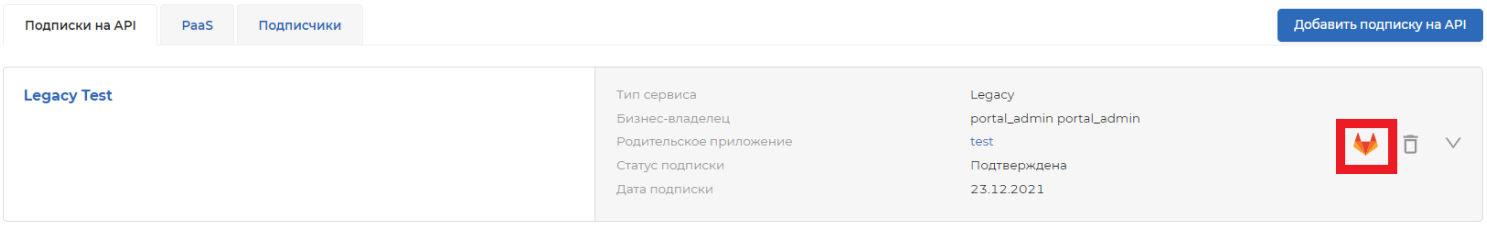
3. Нажмите кнопку "Добавить подписку на API";



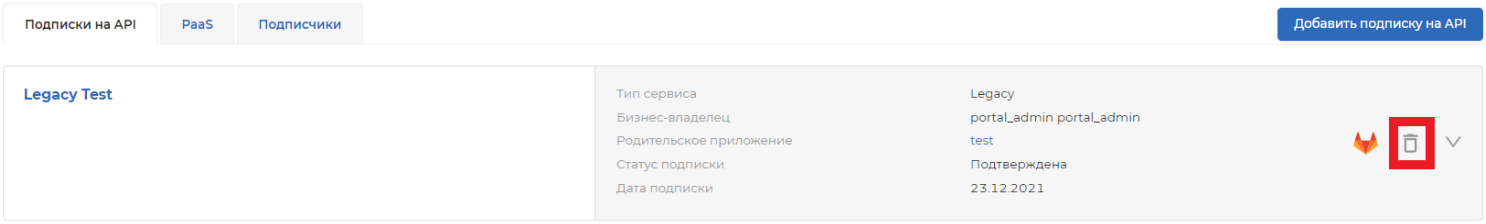
4. Выберите необходимый сервис-подписку, нажмите на логотип выбранного сервиса и на кнопку "Подписаться";



При нажатии на логотип СК "GitLab" откроется репозиторий кода.



При необходимости удалить сервис-подписку - нажмите на кнопку "Удалить подписку".



Возможность подписки на прикладные сервисы

Тип прикладного сервиса	Типы прикладных сервисов, доступных для подписки
Backend	1. Legacy 2. Backend
Frontend	1. Backend 2. Frontend
Kafka	1. Kafka
External (Legacy)	-

> **К СВЕДЕНИЮ**
>
> В конкретной реализации может быть настроено иначе

3.5 Создание шаблона развертывания

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: создан прикладной сервис

> **К СВЕДЕНИЮ**
>
> Шаблон развертывания для среды разработки создается автоматически
>
> Для создания развертывания прикладного сервиса на среде, надо создать развертывания всех платформенных сервисов на этой среде

Шаблон развертывания - это структура, являющаяся частью сервиса, созданная для управления переменными окружения сервиса через ПУ. Шаблон используется для создания развертывания и для создания шаблона развертывания сервиса-подписчика.

1. Перейдите в главном меню на вкладку "Сервисы приложений" и выберите требуемый сервис;
2. Перейдите на вкладку "Шаблоны" и нажмите на кнопку "Добавить шаблон";
3. В открывшейся форме для ввода данных необходимо указать следующие параметры для создания шаблона:

Поле	Пояснение
Название	Введите наименование шаблона. По наименованию шаблона пользователь будет выбирать шаблон из списка других незанятых шаблонов для дальнейшего развертывания
Среда	Выберите среду, на которой будет развертывание
Статус	Выберите "активен" или "неактвен". Если статус оставить в режиме "неактивен" - шаблон нельзя будет использовать для развертывания, также, шаблон не будет отображаться в списке активных шаблонов. Для изменения статуса в рабочем поле (где

Поле	Пояснение
	отображается список всех активных шаблонов) необходимо проставить галку в чек-боксе "Отобразить неактивные" и открыть нужный шаблон в режиме редактирования
Описание	Введите краткое описание шаблона (не обязательно)
Адреса	*gRPC - заполнение данного адреса обязательно для сервисов, которые взаимодействуют по протоколу gRPC;* OpenAPI - заполнение данного адреса обязательно для сервисов, которые взаимодействуют по протоколу OpenAPI; *UI - заполнение данного адреса обязательно для сервисов, типа Front (Angular и React);* Camunda - заполнение данного адреса обязательно для сервиса, в которые встроен Camunda Cockpit. * gRPC to REST - заполнение данного адреса обязательно для сервисов, которые взаимодействуют по протоколу gRPC to REST. Выбор хотя бы одного адреса обязателен. Доступно редактирование доменных имен. Если использование введенного DNS-имени невозможно, будет выведено предупреждение. Платформа поддерживает использование wildcard сертификатов для формирования доменных имён. Если сертификат не добавлен в каталог, сертификат будет выпущен cert-manager'ом Платформы при условии, что данная функция включена.
Порт	Порты нужны для подключения к контейнеру среды исполнения, по которому контейнер приложения получает входящие запросы. Для ввода допускаются целые числа от 1025 до 65535 с возможностью внести изменения, если шаблон не занят
Количество реплик	Введите количество реплик

Поле	Пояснение
Container Params	Введите Liveness probe, Readiness probe, Resources, Отображается синтаксис YAML
Шаблоны подписок	При наличии сервиса-подписки по API - выберите шаблон сервиса-подписки из выпадающего списка
Развёртывания PaaS-подписок	Выберите развертывание PaaS-подписки
Параметры	Добавить параметры (при необходимости). Для передачи параметра необходимо указать **ключ** и **значение** *Для передачи массива значений необходимо указать значения в этом поле через запятую без пробелов*
Переменные окружения	Укажите - Название переменной (должно начинаться с буквы латинского алфавита) - Источник - указать: сервис из блока "Развертывания PaaS-подписок"/сервис-подписку из блока "Шаблоны подписок"/ Параметр - Значение ключа - развертывание из блока "Развертывания PaaS-подписок,/ шаблон из блока "Шаблоны подписок"/ ключ из блока "Параметр
Файлы	Укажите - Путь к файлу (должно начинаться со слеша) - Источник: сервис из блока "Развертывания PaaS-подписок"/ сервис-подписку из блока "Шаблоны подписок"/Параметр - Ключ

К СВЕДЕНИЮ

Для того, чтобы указанные переменные окружения были успешно переданы в развертывание сервиса, необходимо вручную отредактировать файлы value.yaml и

development.yaml в соответствии с инструкцией - см. Инструкция по передаче переменных окружения в скрипты развертывания

К СВЕДЕНИЮ

В целях соблюдения требований по обеспечению информационной безопасности, каждый из содержащихся в Шаблонах развертывания параметров может быть маркирован как "секрет" - что позволяет управлять отображением значения параметра в зависимости от роли пользователя Портала управления. Значения для параметров с признаком секрет - маскируются. Набор параметров для всех шаблонов сервиса и признаки "секрет" - одинаковые, разница в значениях

4. Нажмите на кнопку "Сохранить".

При переходе в режим редактирования шаблона развертывания - доступны все поля для редактирования, кроме:

- Занятость - признак, который проставляется автоматически, если на этом шаблоне есть развертывание в одном из активных статусов;
- Среда

К СВЕДЕНИЮ

При редактировании занятого шаблона необходимо переразвернуть сервис на среде отредактированного шаблона для применения изменений

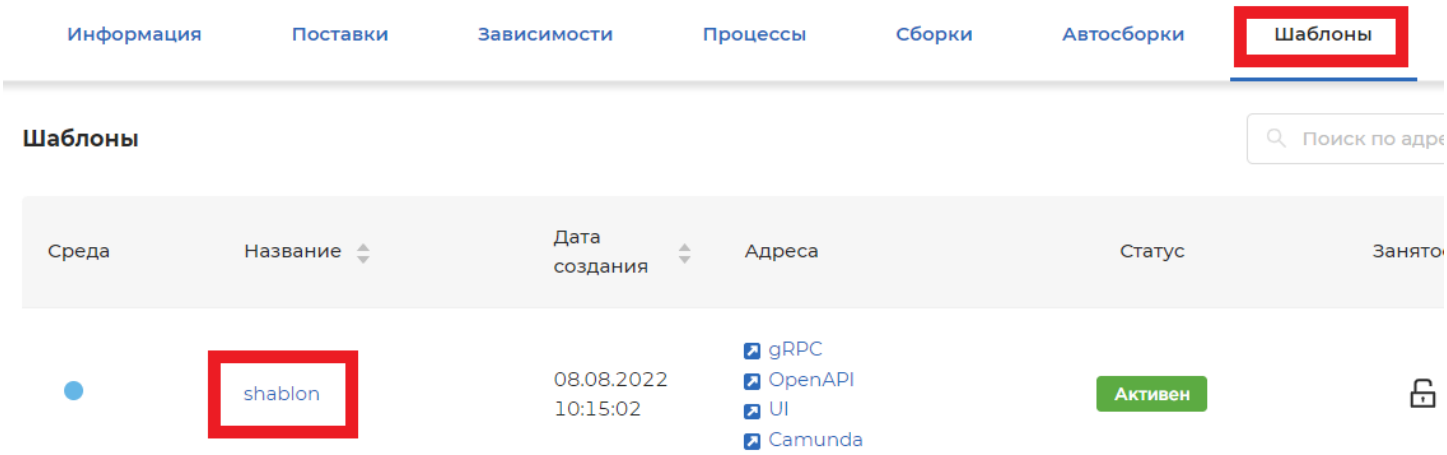
3.6 Использование универсального helm-chart

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: создан прикладной сервис, создан шаблон развертывания

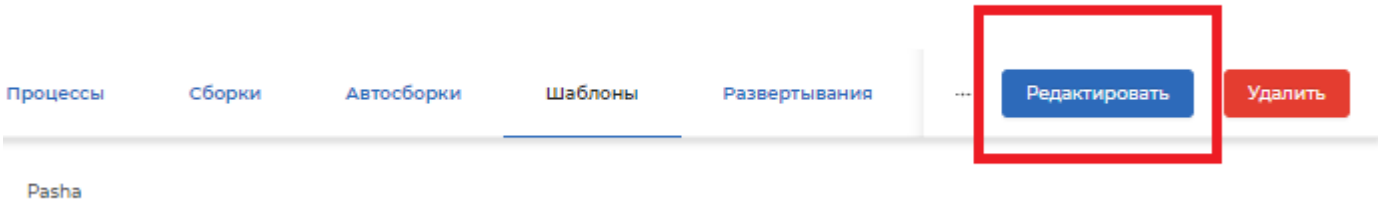
> **К СВЕДЕНИЮ**
>
> Убедитесь, что в исходном коде отсутствует конфигурационная папка "chart"

1. Перейдите в главном меню на вкладку "Сервисы" → "Сервисы приложений" и выберите требуемый сервис;

2. Перейдите на вкладку "Шаблоны" и выберите/ создайте шаблон



3. Откройте "Шаблон" и нажмите на кнопку "Редактировать";

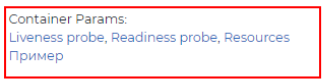


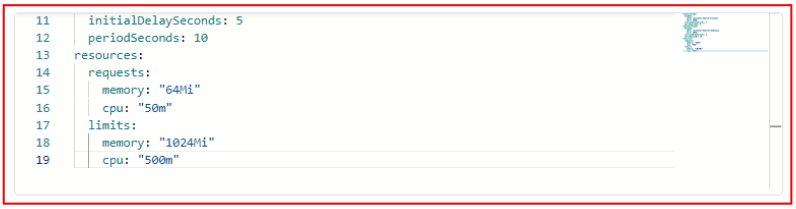
4. Заполните поля "Порт" в блоке "Адреса";

5. Заполните поле "Количество реплик";

6. Заполните поле Container Params:

   - Liveness probe - отвечает за запускает приложение внутри контейнера
   - Readiness probe - отвечает за работоспособность приложения и его условий работы





Пример заполнения полей "Liveness probe", "Readiness probe" и "resources":

```
livenessProbe:
  httpGet:
    path: /actuator/health/liveness
    port: 8080
  initialDelaySeconds: 3
  periodSeconds: 3
readinessProbe:
  httpGet:
    path: /actuator/health/readiness
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 10
resources:
  requests:
    memory: "64Mi"
    cpu: "50m"
  limits:
```

```
    memory: "1024Mi"
    cpu: "500m"
```

7. Нажмите на кнопку "Сохранить".

3.7 Создание сборки

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: создан прикладной сервис, создан шаблон развертывания, осуществлена подписка на PaaS-среда исполнения и его развертывание

Сборка - сущность Портала управления, которая объединяет версию исходного кода сервиса, процесс сборки с редактированием его шагов и результаты процесса сборки, включая артефакты и протоколы.

Сборка осуществляется с помощью Jenkins и параметры ее выполнения зависят от Accelerator Pack, который был выбран при создании прикладного сервиса.

Все функции управления сборками доступны пользователям ПУ в соответствии с ролевой моделью в следующих разделах ПУ:

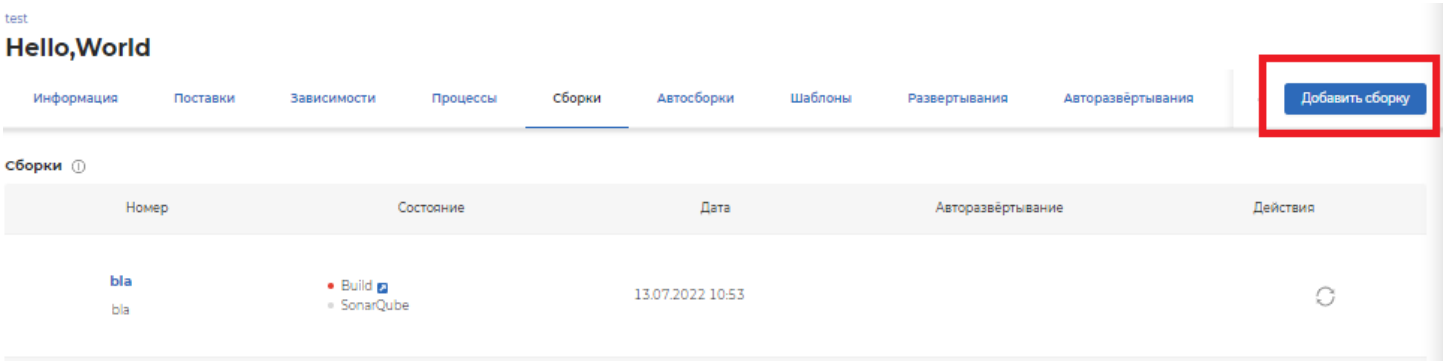
- карточка сборки;
- карточка поставки.

На основании созданной сборки доступно создание поставки сервиса, которая позволяет осуществить развертывание сервиса последовательно на разные на среды (окружения).

> **К СВЕДЕНИЮ**
>
> Предварительным этапом перед созданием сборки необходимо перенести исходный код в репозиторий. Ссылка на репозиторий кода расположена на вкладке "Информация" карточки выбранного сервиса. Перед созданием сборки в коде сервиса необходимо прописать probe check в соответствии с инструкцией Настройка probe check для приложений.

1. Перейти на вкладку "Сервисы приложений" и выбрать необходимый сервис;

2. Перейти на вкладку "Сборки" и нажать на кнопку "Добавить сборку";



3. В открывшейся форме для ввода данных необходимо указать следующие параметры для создания сборки:

    - Обозначение сборки - введите наименование сборки. По наименованию сборки пользователь будет выбирать сборку для добавления в поставку.
    - Коммит - введите из GitLab название ветки расположения исходного кода, тег и сгенерированный id-commit.
    - Выбрать запускаемые проверки, проставив галки в чек-боксах:
        - Build - галка в чек-боксе стоит автоматически и недоступна для редактирования;
        - SonarQube - проверка кода и его качества, галка в чек-боксе стоит автоматически и недоступна для редактирования;
        - Unit-тесты - проверка отдельных модулей кода.
    - Описание сборки - введите краткое описание сборки.
    - Авторазвертывание(не обязательно) - выберите авторазвертывание, подробнее: Создание авторазвертывания

4. Нажать на кнопку "Создать"

5. После создания, запускается процесс сборки, появляется всплывающее окно "Сборка создана".

6. При успешной сборке Build загорится зеленым, в обратном случае красным, аналогично для других проверок.

Сборку можно перезапустить и запустить развертывание, нажав на соответствующие кнопки.

Протокол сборки возможно просмотреть, нажав соответствующую кнопку рядом с Build или соответствующую кнопку рядом с SonarQube.

3.8 Создание автосборки

> **ⓘ ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: создан прикладной сервис, создан шаблон развертывания, осуществлена подписка на PaaS-среда исполнения и его развертывание

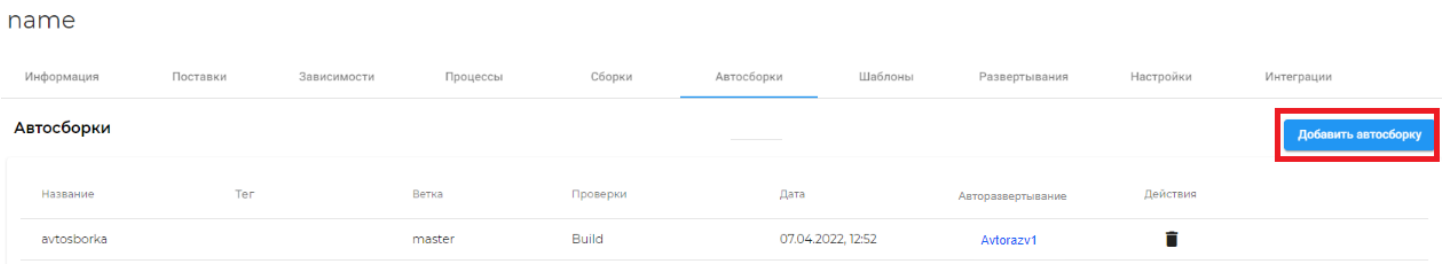
Автосборка - это надстройка к сборке. Автосборка объединяет версию исходного кода сервиса, процесс сборки с редактированием его шагов и результаты процесса сборки, включая артефакты и протоколы. Автосборка включает в себя следующую логику:

- Пользователь вносит в GitLab изменения в код (создает коммит);
- Информация о новом коммите передается в ПУ и запускается проверка - предикат - это поле branch с паттерном для имени бранчи коммита и поле tag с паттерном для тега;
- Если хотя бы один из паттернов выполняется, то ПУ автоматически запустит сборку с учетом внесенных изменений в код;
- Если срабатывает несколько сборок, то выбирается и выполняется самая свежая по updated_at;
- В ПУ будет создана новая автосборка, но при этом старая сборка (если она существовала в ПУ) не будет удалена;
- Автосборка может быть создана только на dev среде;
- ПУ предоставляет возможность создать несколько автосборок для одного сервиса с указанием разных веток/тегов/параметров сборки.

> **ⓘ К СВЕДЕНИЮ**
>
> Предварительным этапом перед созданием сборки необходимо перенести исходный код в репозиторий. Ссылка на репозиторий кода расположена на вкладке "Информация" карточки выбранного сервиса. Перед созданием сборки в коде сервиса необходимо прописать probe check в соответствии с инструкцией Настройка probe check для приложений.

1. Перейти на вкладку "Сервисы приложений" и выбрать необходимый сервис;

2. Перейти на вкладку "Автосборки" и нажать на кнопку "Добавить автосборку";



3. Укажите следующие параметры для создания автосборки в открывшейся форме для ввода данных:

   - Название автосборки - введите название автосборки;
   - Тег - введите из GitLab тег под которым сохранили изменения в исходном коде. Поле необязательное для заполнения;
   - Ветка - введите из GitLab название ветки расположения исходного кода. Поле обязательное для заполнения;
   - Выбрать запускаемые проверки, проставив галки в чек-боксах:
     - Build - галка в чек-боксе стоит автоматически и недоступна для редактирования;
     - SonarQube - проверка кода и его качества, галка в чек-боксе стоит автоматически и недоступна для редактирования;
     - Unit-тесты - проверка отдельных модулей кода.
   - Авторазвертывание - выберите существующий шаблон развертывания, подробнее: Создание авторазвертывания

4. Нажать на кнопку "Создать";

5. Нажать на ссылку "Репозиторий кода" на вкладке "Информация";

6. Внести изменения в исходный код. Нажать на кнопку "Commit changes".

К СВЕДЕНИЮ

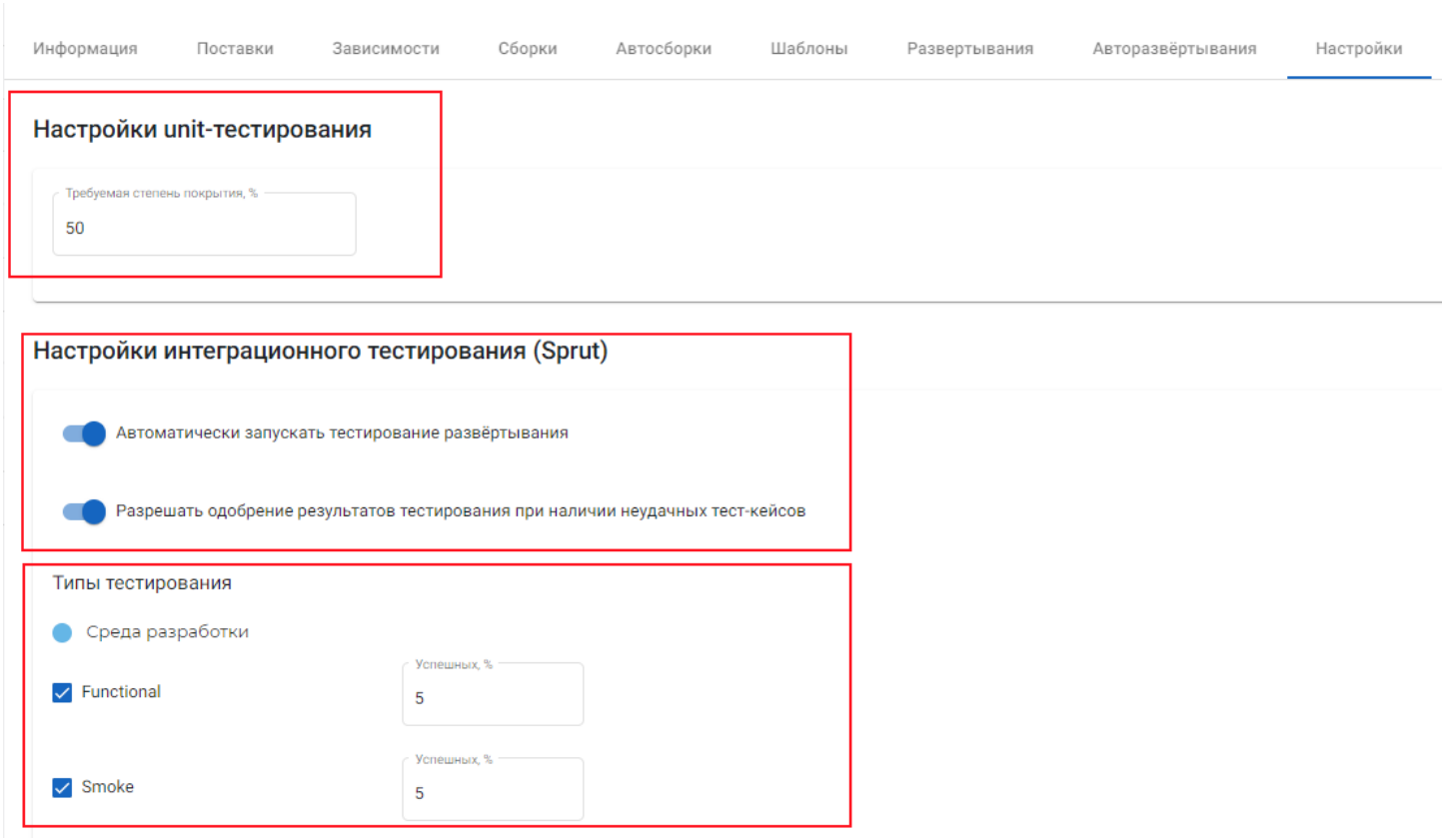
Перед сохранением убедитесь, что название ветки с новым коммитом совпадает с названием ветки созданной автосборки

3.9 Настройка тестирования

> ⓘ **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: создан прикладной сервис

Чтобы тесты автоматически запускались после развертывания сервиса, или был установлен процент прохождения для их успешности, необходимо заранее задать параметры тестирования на вкладке "Настройки" в карточке сервиса:

1. Откройте вкладку "Настройки" в карточке сервиса;
2. Установите % успешного прохождения unit-тестирования (0% - 100%);
3. Включите условие "Автоматически запускать тестирование развёртывания", чтобы тестирование запустилось автоматически после перехода развертывания в статус "Успешно";
4. Включите условие "Разрешать одобрение результатов тестирования при наличии неудачных тест-кейсов", чтобы была возможность переводить поставку на более высокие среды не смотря на непройденные автотесты;
5. Укажите типы тестирования для каждой среды, выбрав соответствующие чек-боксы;
6. Установите % успешного прохождения автотестов (0% - 100%);
7. Нажмите кнопку "Сохранить".



После развертывания поставки на среде с настроенными автотестами запускаются выбранные типы тестирования.

При полном или частичном прохождении автотестов пользователь в праве перейти на следующую среду.

3.10 Создание поставки

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: сборка в статусе "Собрана", создан шаблон развертывания

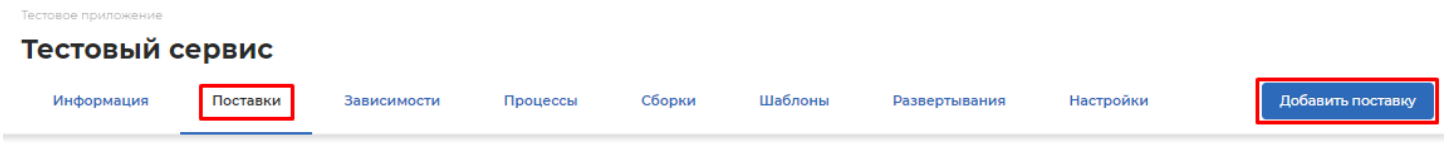
Поставка - управляющая сущность ПУ, которая обеспечивает непрерывность доставки сервиса на разные среды. В поставке агрегируются процессы сборки, осуществления и настройки проверок, создания развертывания сервиса на разных средах.

> **ПРЕДУПРЕЖДЕНИЕ**
>
> Для создания развертывания необходимо, чтобы шаблон развертывания был валидным и активным.

> **К СВЕДЕНИЮ**
>
> Из каждого статуса поставки можно вручную перейти к "Закрыта в статусе:...", в таком случае поставка будет скрыта с вкладки "Поставки" (для отображения скрытых поставок необходимо выбрать фильтр "Все" на переключателе в правом верхнем углу страницы).

1. Перейдите на вкладку "Сервисы приложений" и выберите требуемый сервис;

2. Перейдите на вкладку "Поставки" и нажмите на кнопку "Добавить поставку";



3. В открывшейся форме для ввода данных укажите следующие параметры для создания поставки:

   - Обозначение поставки - введите числовое обозначение поставки;

- Коммит API полученный из GitLab - введите из GitLab название ветки расположения исходного кода или сгенерированный id-commit.;
- Сборка - выбирается из списка успешно собранных сборок;
- Планируемая дата релиза;
- Описание - введите краткое описание сборки.

4. Нажмите кнопку "Создать поставку".

3.11 Развертывание прикладного сервиса

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик - для сред dev, test, stage/Сотрудник эксплуатации - для сред dev, test, stage, prod
>
> Предварительные действия: сборка в статусе "Собрана", создан шаблон развертывания

> **К СВЕДЕНИЮ**
>
> У развертывания прикладного сервиса есть признак "валидность":
>
> - **Валидно** - развертывание считается валидным, если все прикладные сервисы из зависимостей успешно развернуты на шаблонах развертывания, указанных в шаблоне данного развертывания.
> - **Не валидно** - развертывание считается не валидным, если отсутствуют успешные развертывания всех прикладных сервисов из зависимостей, на шаблонах развертывания, указанных в шаблоне данного развертывания.

Развертывание прикладного сервиса через сборку

1. Перейдите на вкладку "Сервисы приложений" и выберите необходимый сервис;
2. Перейдите на вкладку "Сборки" и нажмите на кнопку "Запустить развертывания" в строке требуемой сборки или в её карточке;

3. В открывшейся форме для ввода данных укажите следующие параметры для развертывания:
    - Среда (среда разработки);

- Шаблон развертывания - выбирается по наименованию из списка незанятых и валидных шаблонов.

4. Нажмите на кнопку "Запустить развертывание".

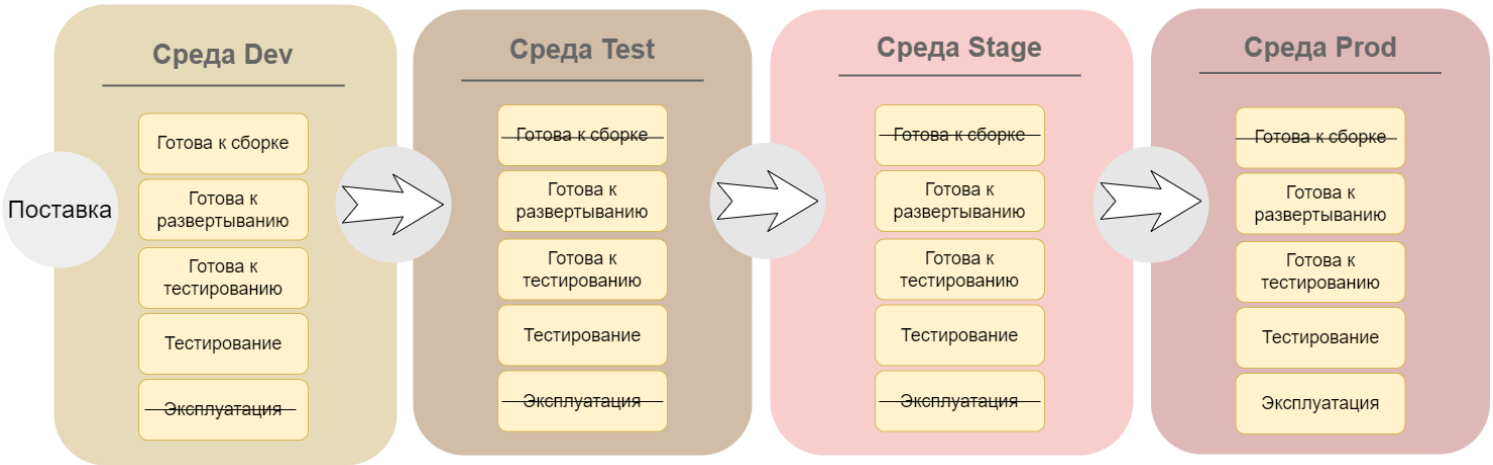
При успешном развертывании появится статус "Развернуто", в ином случае будет стоять статус "Ошибка развертывания".

Протокол развертывания можно посмотреть, нажав на соответствующую кнопку рядом в карточке развертывания или их списке.

> **К СВЕДЕНИЮ**
>
> Для развертывания сервиса на более высоких средах, необходимо:
>
> - Создать поставку;
> - Присоединить сборку к поставке.

Развертывание прикладного сервиса через поставку

В рамках жизненного цикла поставки осуществляется ее последовательный переход по средам и статусам:



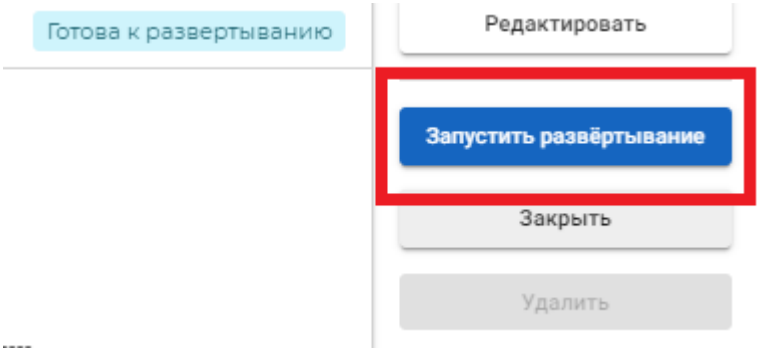
> **К СВЕДЕНИЮ**
>
> Развернуть прикладной сервис на среде разработки можно:

- через сборку, нажав "запустить развертывание" в карточке сборке или их списке. После её прикрепления к поставке, поставка сразу окажется в статусе "готова к тестированию" на среде dev;
- через поставку, прикрепив сборку, нажав "запустить развертывание" в карточке поставки.

1. На вкладке "Поставки" откройте карточку поставки, она находится в статусе "Готова к развертыванию";

> **⚠ ПРЕДУПРЕЖДЕНИЕ**
>
> В случае, если в сборке обязательные проверки отличаются от предъявляемых требований, следует откорректировать блок обязательных проверок через режим "Редактирование".

2. Нажмите на кнопку "Запустить развертывание";



3. В открывшемся окне "Новое развертывание" укажите следующую информацию:
    - Среда;
    - Запуск проверок (План-Факт/Авторизация);
    - Шаблон.
4. Нажмите на кнопку "Запустить развертывание";

После успешного развертывания Поставка переходит в статус "Готова к тестированию". В данном статусе доступны следующие кнопки:

- В блоке "Развертывания" доступны кнопки управления развертыванием - "Остановить", "Запустить" (кнопка появляется при нажатии на кнопку "Остановить"), "Перезапустить", "Удалить";

- Кнопка "Закрыть", чтобы закрыть всю работу с данной поставкой;
- Кнопка "Тестирование", чтобы перейти к тестированию.

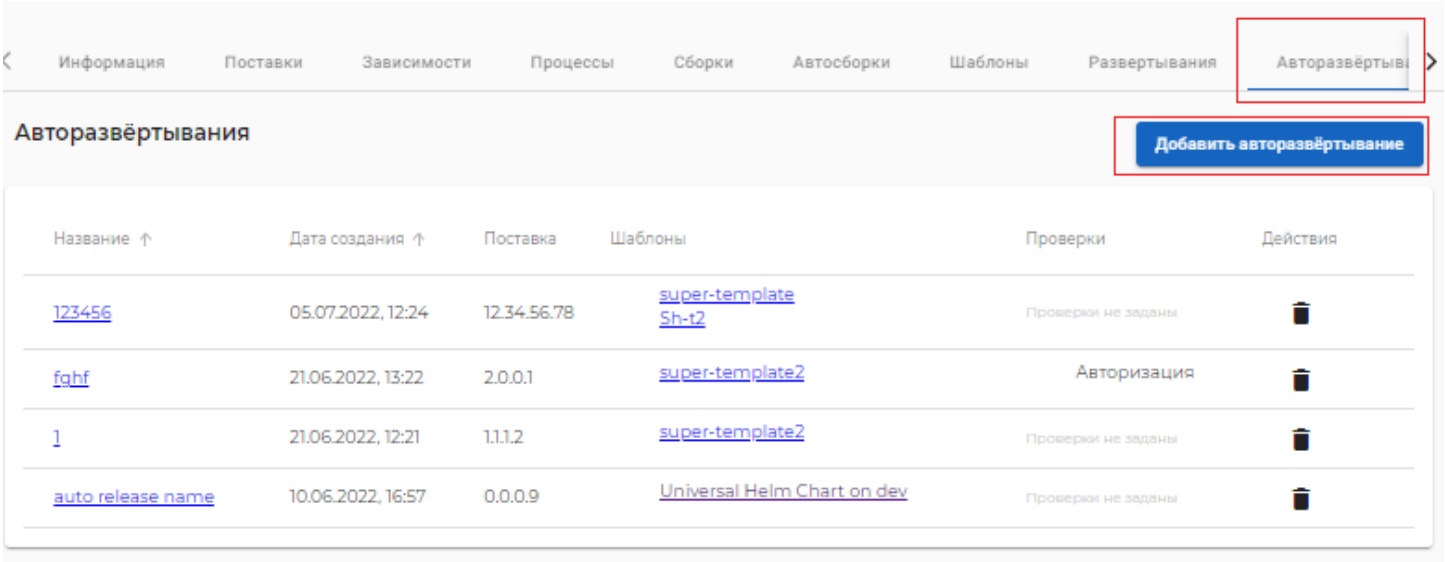
> **⚠ ПРЕДУПРЕЖДЕНИЕ**
>
> Если на вкладке "Настройки" включено автотестирование, то поставка пропустить статус "Готова к тестированию" и сразу перейдет в "Тестирование". Если автотесты пройдены успешно, то поставка перейдет на среду, следующую по грейду. Если автотесты завершились с ошибками, то портал разрешает вручную перевести поставку на следующую среду по грейду, включив тогл "Разрешать одобрение результатов тестирования при наличии неудачных тест-кейсов" на вкладке "Настройки" и нажав на кнопку "Тестирование успешно".

5. Нажмите на кнопку "Тестирование";
6. Нажмите на кнопку "Тестирование успешно";
7. Для перехода поставки на более высокие среды повторите шаги 2-6 до перехода поставки в статус "Эксплуатация".

3.12 Авторазвертывание

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: сборка в статусе "Собрана", создан шаблон развертывания

1. Перейдите в главном меню на вкладку "Авторазвертывания" и нажмите кнопку "Добавить авторазвёртывание";



2. Заполните поля, отобразившиеся в открывшемся окне;

   Авторазвертывание будет создано только на тех средах, на которых указаны шаблоны развертывания (т.е. если вы указали шаблон только для среды dev, то авторазвертывание остановится после разворачивания и тестирования сервиса на среде dev и поставка будет в статусе "Готова к развертыванию". Дальнейшая работа с поставкой возможно только вручную).

3. Нажмите на кнопку "Создать";

4. Откройте вкладку "Настройки" в карточке сервиса;

5. Установите % успешного прохождения unit-тестирования (0% - 100%);

6. Включите условие "Автоматически запускать тестирование развёртывания", чтобы тестирование запустилось автоматически после перехода развертывания в статус "Успешно";

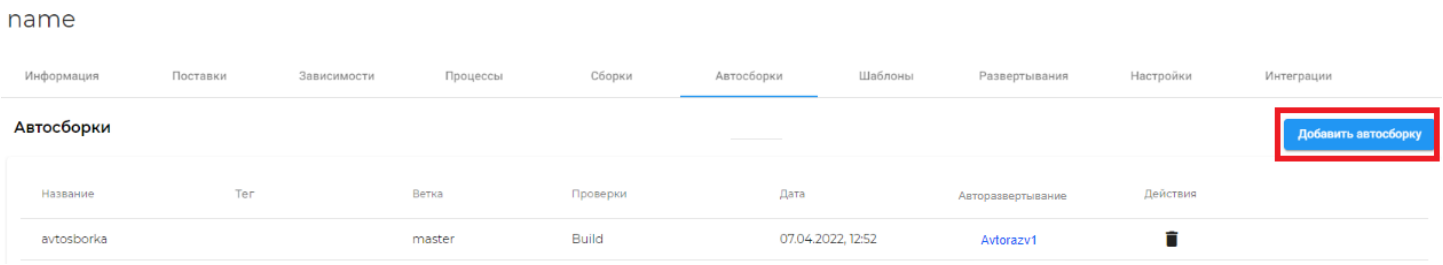
7. Включите условие "Разрешать одобрение результатов тестирования при наличии неудачных тест-кейсов", чтобы была возможность переводить поставку на более высокие среды не смотря на непройденные автотесты;

8. Укажите типы тестирования для каждой среды, выбрав соответствующие чек-боксы;

9. Установите % успешного прохождения автотестов (0% - 100%);

10. Нажмите кнопку "Сохранить";

11. Перейти на вкладку "Автосборки" и нажать на кнопку "Добавить автосборку";



12. Укажите следующие параметры для создания автосборки в открывшейся форме для ввода данных:

    - Название автосборки - введите название автосборки;
    - Тег - введите из GitLab тег под которым сохранили изменения в исходном коде. Поле необязательное для заполнения;
    - Ветка - введите из GitLab название ветки расположения исходного кода. Поле обязательное для заполнения;
    - Выбрать запускаемые проверки, проставив галки в чек-боксах:
        - Build - галка в чек-боксе стоит автоматически и недоступна для редактирования;
        - SonarQube - проверка кода и его качества, галка в чек-боксе стоит автоматически и недоступна для редактирования;
        - Unit-тесты - проверка отдельных модулей кода.
    - Авторазвертывание - выберите созданное авторазвертывание

13. Нажать на кнопку "Создать";

14. Нажать на ссылку "Репозиторий кода" на вкладке "Информация";

15. Внести изменения в исходный код. Нажать на кнопку "Commit changes".

> **К СВЕДЕНИЮ**
>
> Для ручного запуска сборки вместо шагов 11-15 необходимо создать сборку в портале вручную (см. Создание сборки)

После этого будет запущена сборка сервиса.

При её успешном завершении будет создана поставка и запущено развертывание на низшей среде.

Далее развертывание будет протестировано и переведено на среду, следующую по грейду.

При отсутствии ошибок, сервис будет выведен в эксплуатацию.

3.13 Работа с разделом "Аудит"

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: Не требуются

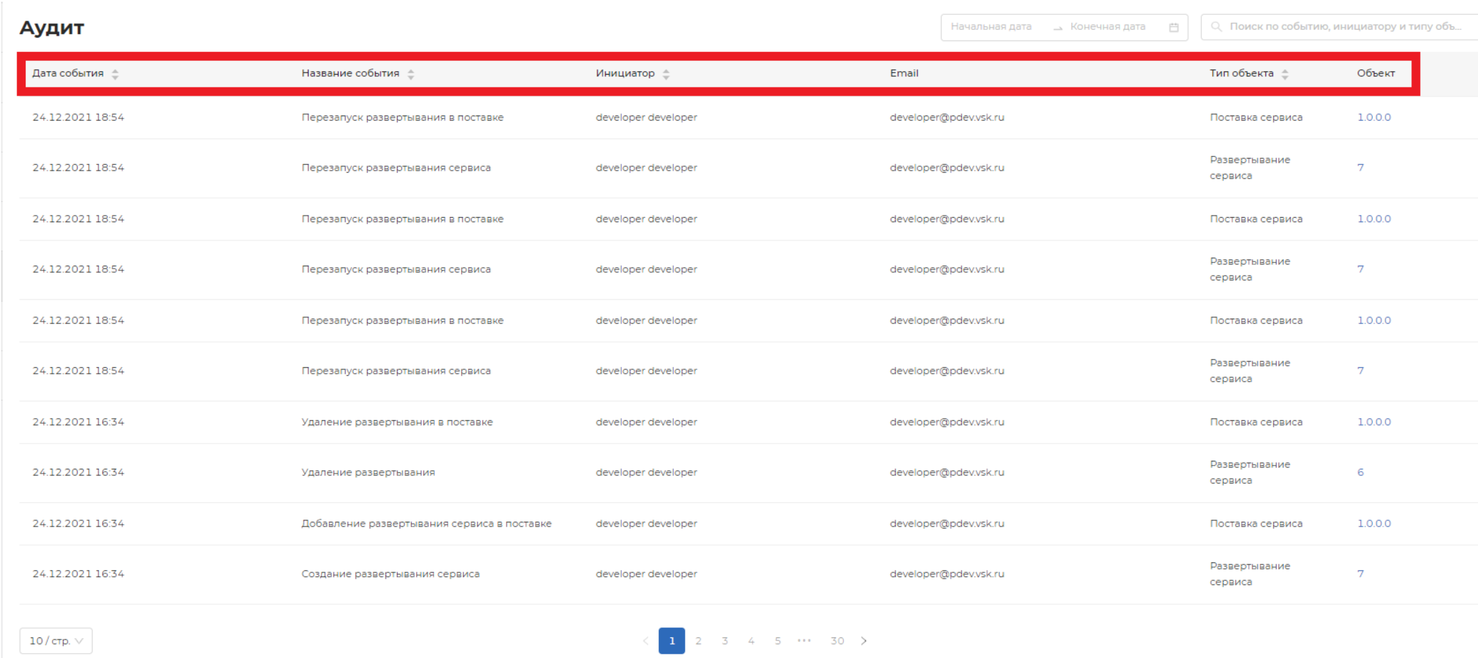
> **К СВЕДЕНИЮ**
>
> Раздел "Аудит" представляет собой справочник событий, совершенных пользователями.

Просмотр

Пользователю в разделе "Аудит" отображаются только записи о приложениях, в команду которых он добавлен *Все типы событий отображаются в разделе "Справочники" - "Типы событий"*

В данном разделе представлена таблица со столбцами:

1. Дата события - время совершения действия в формате ДД.ММ.ГГГГ в ЧЧ:ММ;
2. Название события - название в соответствии с таблицей справочника "Типы событий";
3. Инициатор - Фамилия, Имя сотрудника, совершившего действие в ПУ;
4. Email - рабочая электронная почта инициатора;
5. Тип объекта - название сущности ПУ ("Среда"/"Приложение"/"Сервис" в соответствии с описанием в таблице в разделе справочника "Типы событий");
6. Объект - сущность ПУ.



Фильтрация событий

Доступны фильтры по

- Дате события;
- Названию объекта;
- Инициатору;
- Email;
- Типу объекта;
- Объекту.

Поиск событий

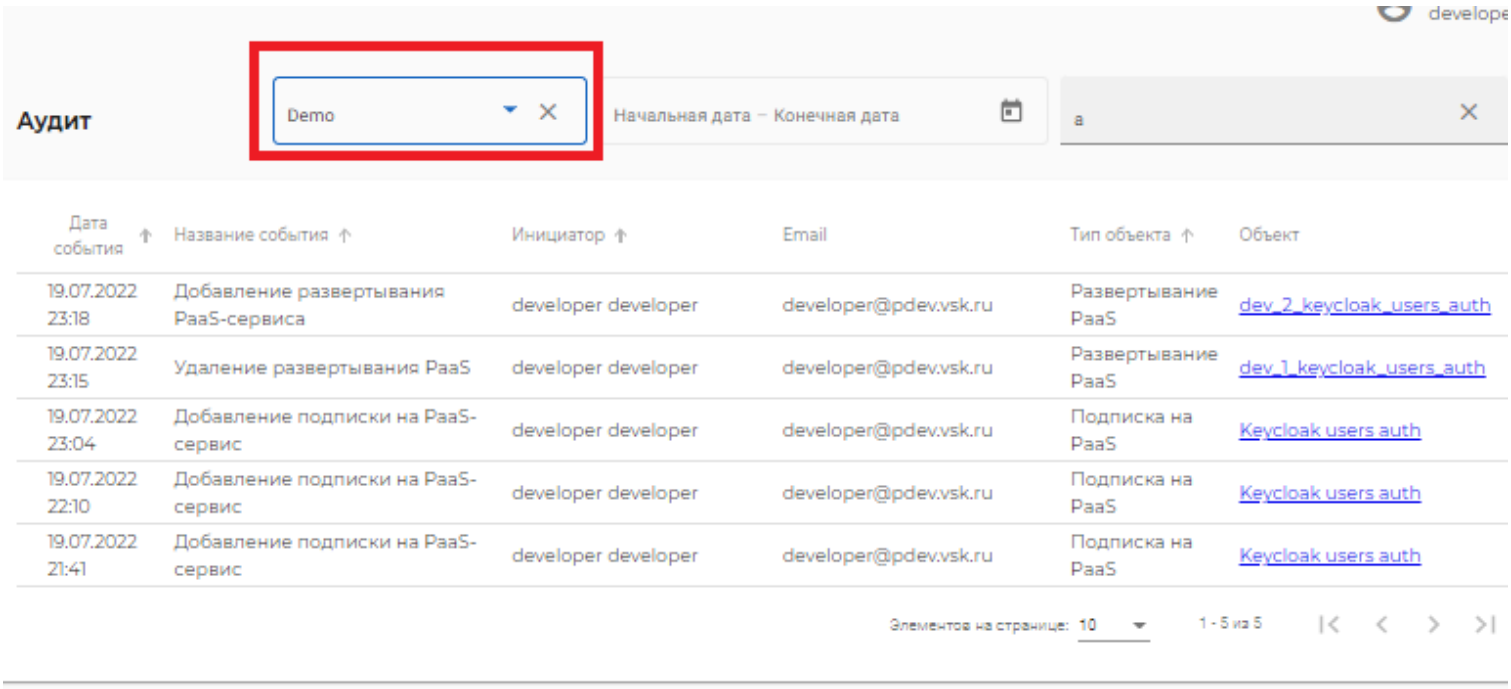
Доступен поиск по

- Приложению;
- Дате;
- Названию.

Для поиска по приложению

1. Щелкните на поле "Поиск по приложению.(Пользователю с ролью "Администратор" доступен выбор по любым приложениям);
2. В выпадающем окне выберите нужное.

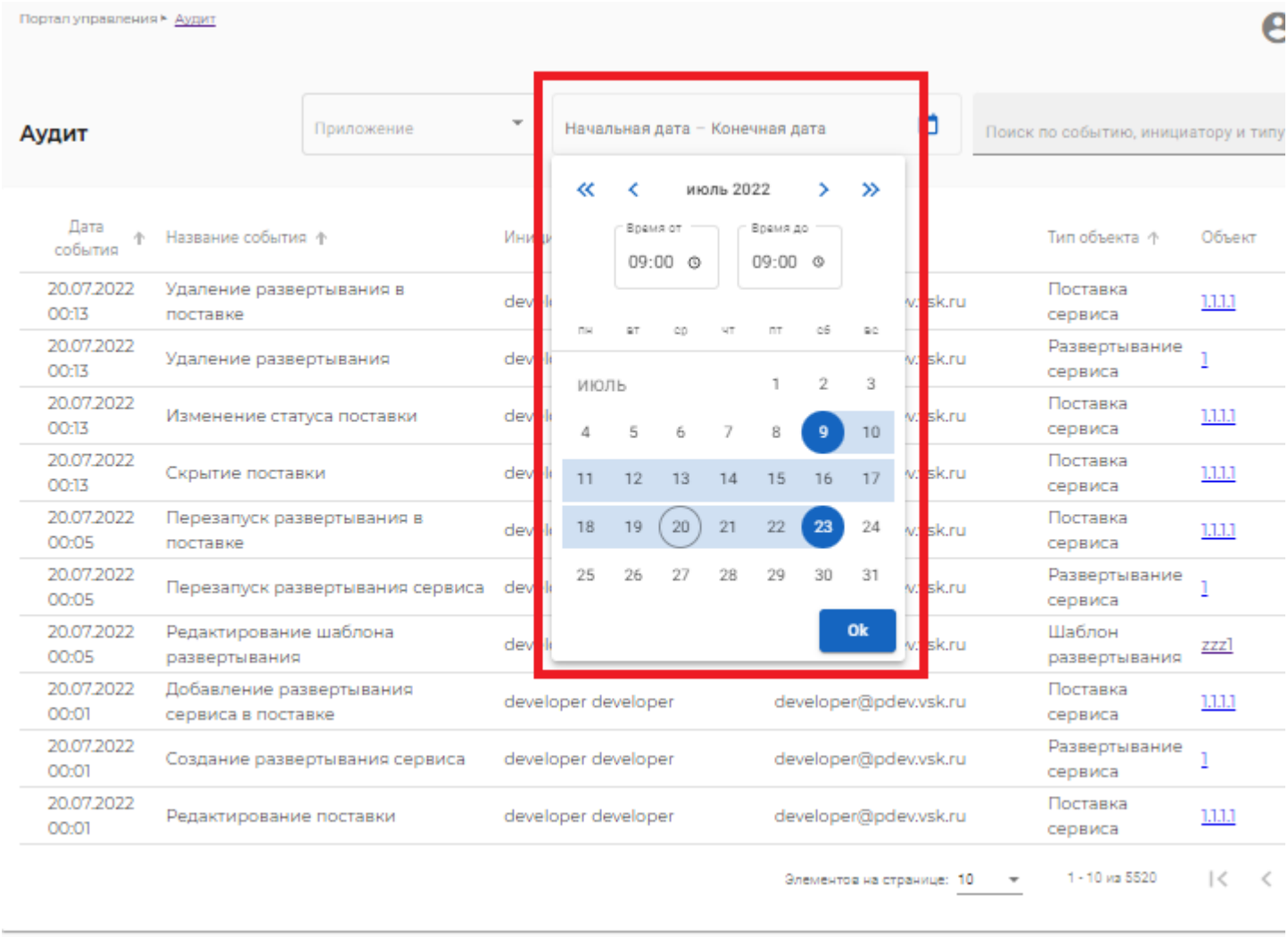
После этого в таблице отразятся события, в которых участвовали события с сущностями выбранного приложения.



Для поиска по Дате

1. Перейдите на поле "Начальная дата". Введите или выберите дату и время. Нажмите "ок";
2. Перейдите на поле "Конечная дата". Введите или выберите дату и время. Нажмите "ок".

После этого в таблице отразятся события, совершенные в выбранный временной промежуток.



Для поиска по Названию

1. Щелкните на поле "Поиск по событию, инициатору и типу объекта";
2. Введите необходимое для поиска слово, нажмите Enter.

После этого в таблице отразятся события, ячейки "Событие" и/или "Инициатор", и/или "Тип объекта" содержат введенное слово.

Аудит

Приложение | Начальная дата – Конечная дата | a

Дата события	Название события	Инициатор	Email	Тип объекта	Объект
19.07.2022 23:18	Добавление развертывания PaaS-сервиса	developer developer	developer@pdev.vsk.ru	Развертывание PaaS	dev_2_keycloak_users_auth
19.07.2022 23:15	Удаление развертывания PaaS	developer developer	developer@pdev.vsk.ru	Развертывание PaaS	dev_1_keycloak_users_auth
19.07.2022 23:04	Добавление подписки на PaaS-сервис	developer developer	developer@pdev.vsk.ru	Подписка на PaaS	Keycloak users auth
19.07.2022 22:10	Добавление подписки на PaaS-сервис	developer developer	developer@pdev.vsk.ru	Подписка на PaaS	Keycloak users auth
19.07.2022 21:41	Добавление подписки на PaaS-сервис	developer developer	developer@pdev.vsk.ru	Подписка на PaaS	Keycloak users auth
18.07.2022 14:34	Копирование шаблона развертывания	Хорунжая Мария	khorunzhaya@adds.pdev.vsk.ru	Шаблон развертывания	dnzn1
18.07.2022 14:33	Запуск сборки после создания	Хорунжая Мария	khorunzhaya@adds.pdev.vsk.ru	Сборка сервиса	989846
18.07.2022 14:33	Создание сборки	Хорунжая Мария	khorunzhaya@adds.pdev.vsk.ru	Сборка сервиса	989846
18.07.2022 14:32	Создание поставки	Хорунжая Мария	khorunzhaya@adds.pdev.vsk.ru	Поставка сервиса	5.5.5.5
13.07.2022 12:11	Добавление развертывания PaaS-сервиса	developer developer	developer@pdev.vsk.ru	Развертывание PaaS	prod_1_openshift_okd_lan

Элементов на странице: 10 | 1 - 10 из 1027

3.14 Работа с feature-toggle

> ⓘ **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: Создано приложение, создан прикладной сервис

Feature toggle - альтернатива сохранению нескольких ветвей функций в исходном коде. Условие в коде включает или отключает функцию во время выполнения. В гибких настройках feature toggle используется в рабочей среде для включения функции по требованию для некоторых или всех пользователей. Функция интегрируется в основную ветку до ее завершения. Версия разворачивается в тестовой среде один раз, переключатель позволяет включить функцию и протестировать ее.

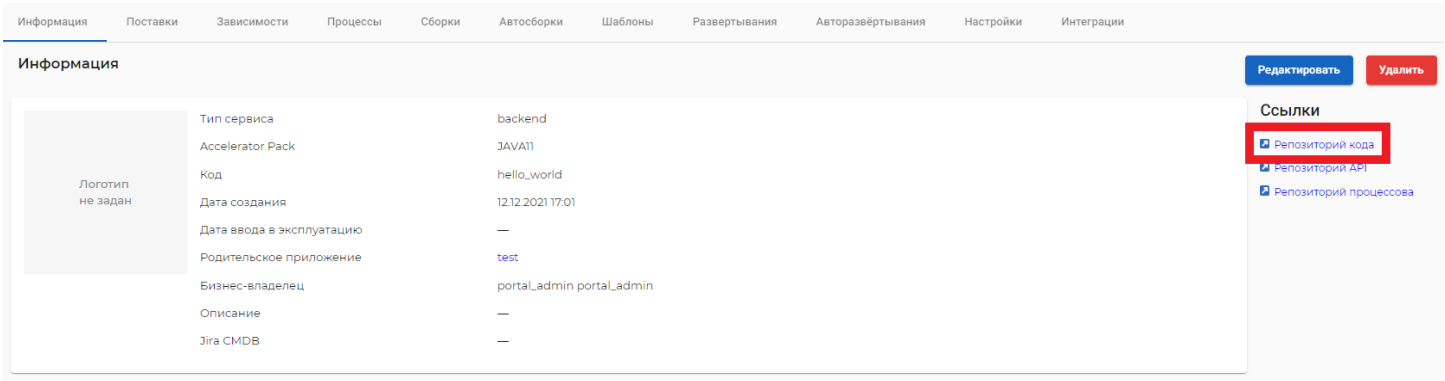
Для подключения

Подключите и настройте библиотеку Unleash в вашем сервисе.

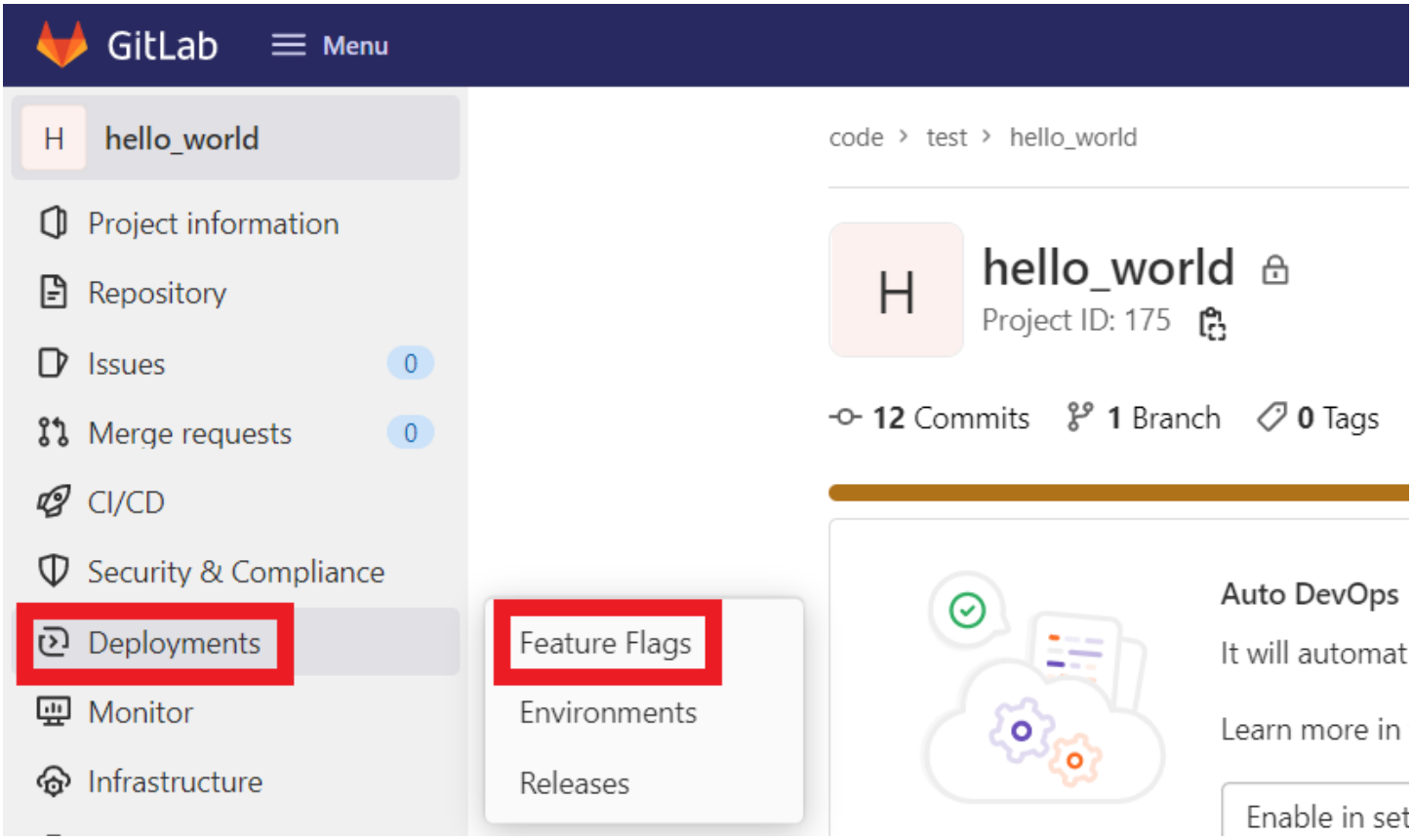
SDK и примеры для различных языков доступны на официальном сайте: Unleash.

Для создания feature flag

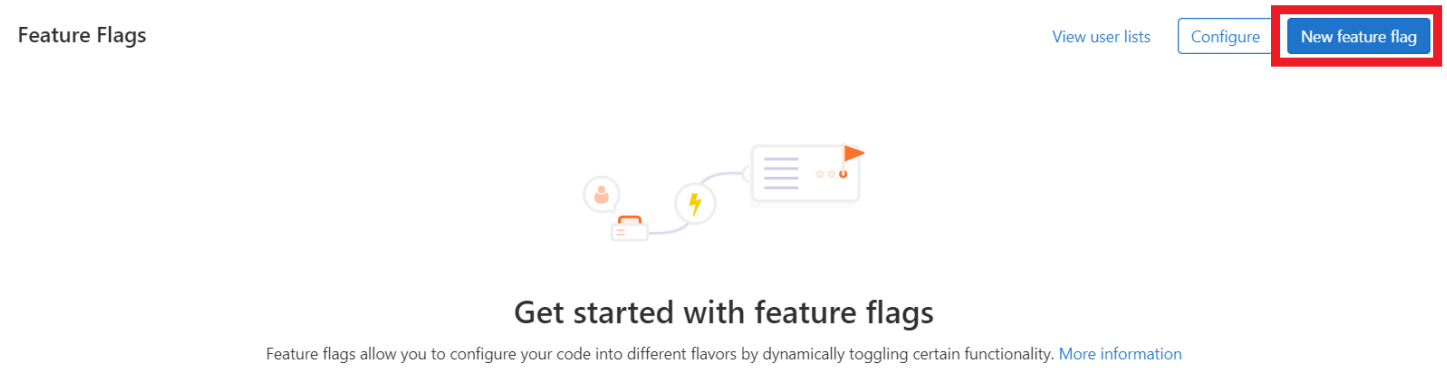
1. Перейти по ссылке "Репозиторий кода" на вкладке сервиса приложения . Авторизуйтесь в GitLab;



2. Перейти в боковой панели на вкладку "Deployments" → "Feature flags";



3. Нажать на кнопку "New feature flag";



4. Заполнить карточку создания Feature flag: Подробная инструкция

- **Name** - название feature flag.Должно начинаться с буквы, содержать только строчные буквы, цифры, символы подчеркивания (*) или дефисы (-), и не заканчиваться дефисом (-) или подчеркиванием (*).
- **Discription** - описание (необязательное поле, 255 символов максимум).
- **Strategies** - стратегии определяют применение feature flag
- **Type**:

```
* <u>All users</u> - включает функцию для всех пользователей.
* <u>Percent rollout</u> - включает функцию для процента просмотров страниц с
```

```
настраиваемой согласованностью поведения.
* <u>Percent of users</u> - включает функцию для процента аутентифицированных пользователей.
```

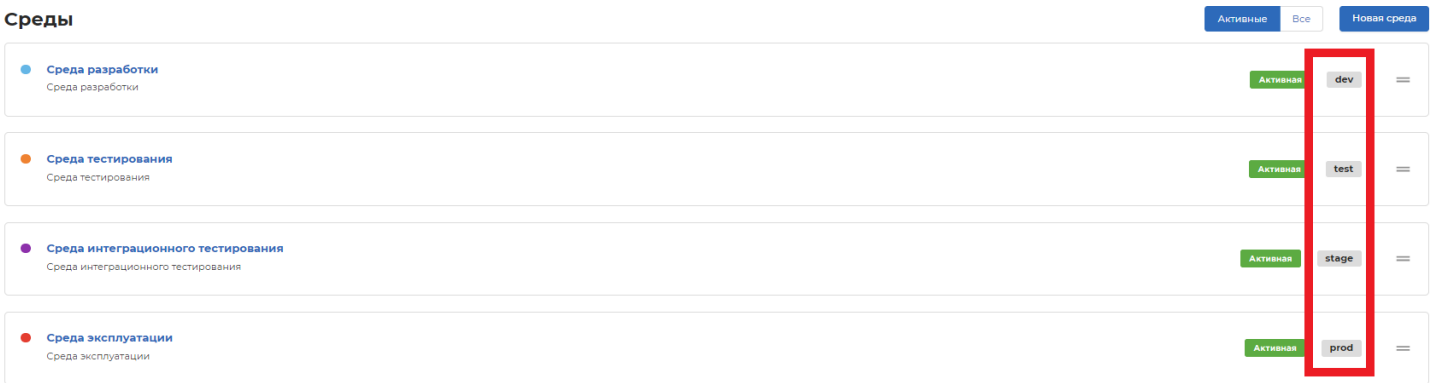
и другие.

- **Environments** - включает функцию для определенной среды по её коду.

5. Для определения среды развертывания сервиса необходимо оперировать переменной **envcode**:

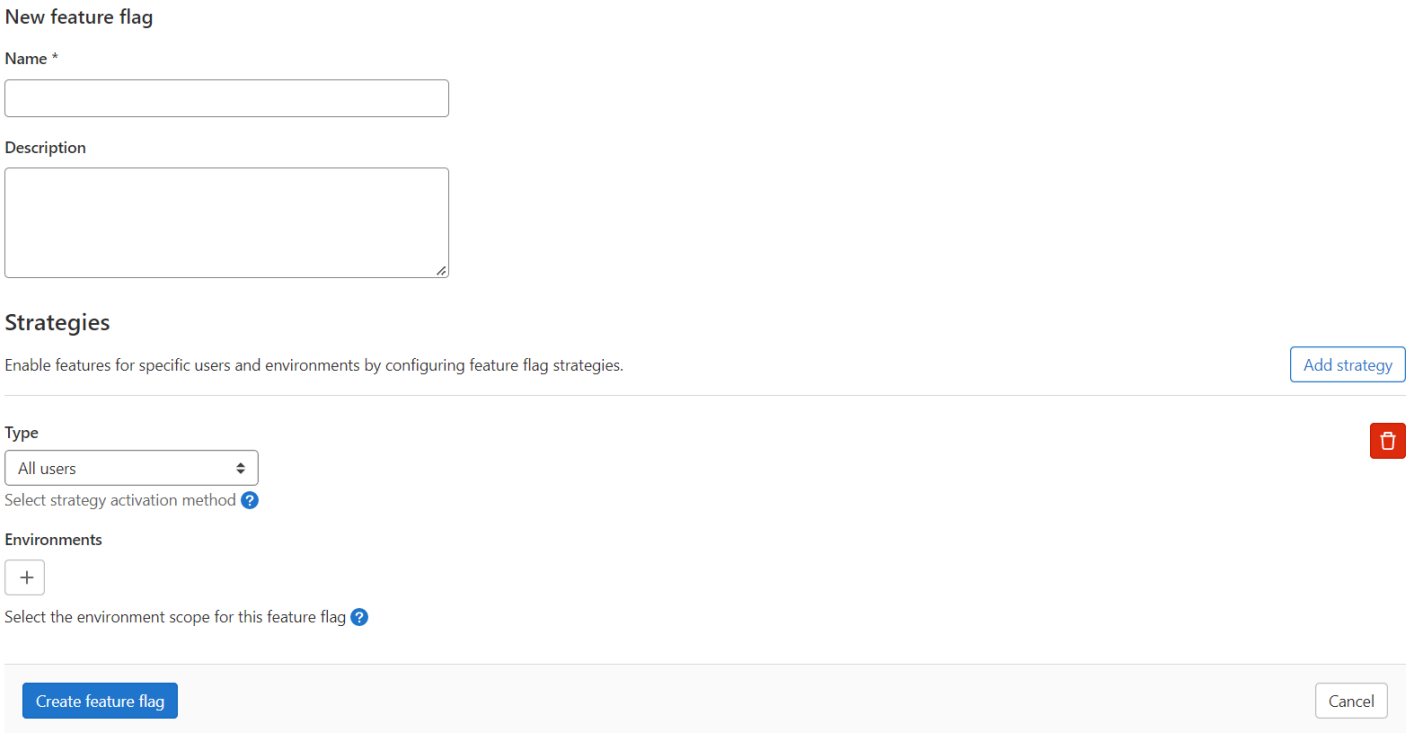
```
apiVersion: apps/v1
kind: Deployment
metadata:
    ...
spec:
    ...
    template:
        ...
        spec:
            containers:
                - name: {{ .Values.deployment.finalname }}
                  ...
                  env:
                    - name: ENV_CODE
                      value: {{ .Values.deployment.envcode | quote }}
```

6. Код среды можно найти на боковой вкладке "Среды" ПУ:



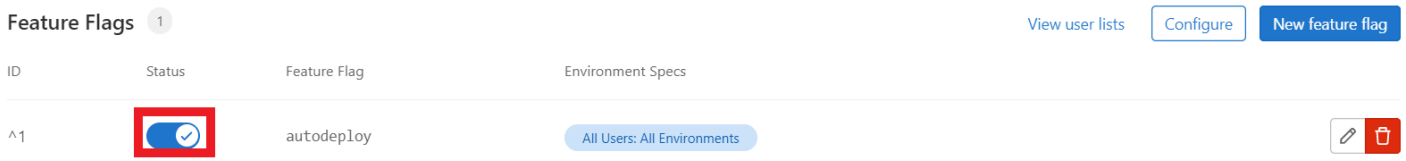
Для добавления среды впишите её код в строку "Environments" и нажмите "Create {name}"

7. Нажать "Create Feature Flag".



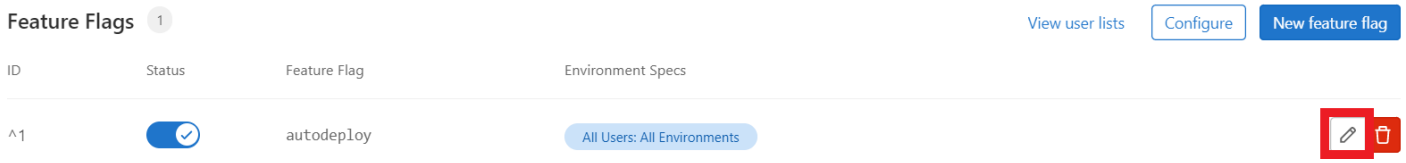
Для включения/выключения feature flag

1. После создания feature flag он автоматически включен. Для выключения необходимо перевести тоггл из статуса "ON" в статус "OFF".



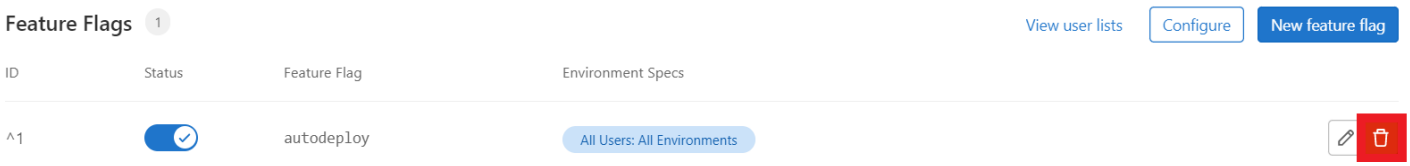
Для редактирования feature flag

1. Для внесения изменений в настройки feature flag необходимо нажать на кнопку "Edit" в строке выбранного feature flag.



Для удаления feature flag

1. Чтобы удалить feature flag необходимо нажать на кнопку "Delete" в строке выбранного feature flag.



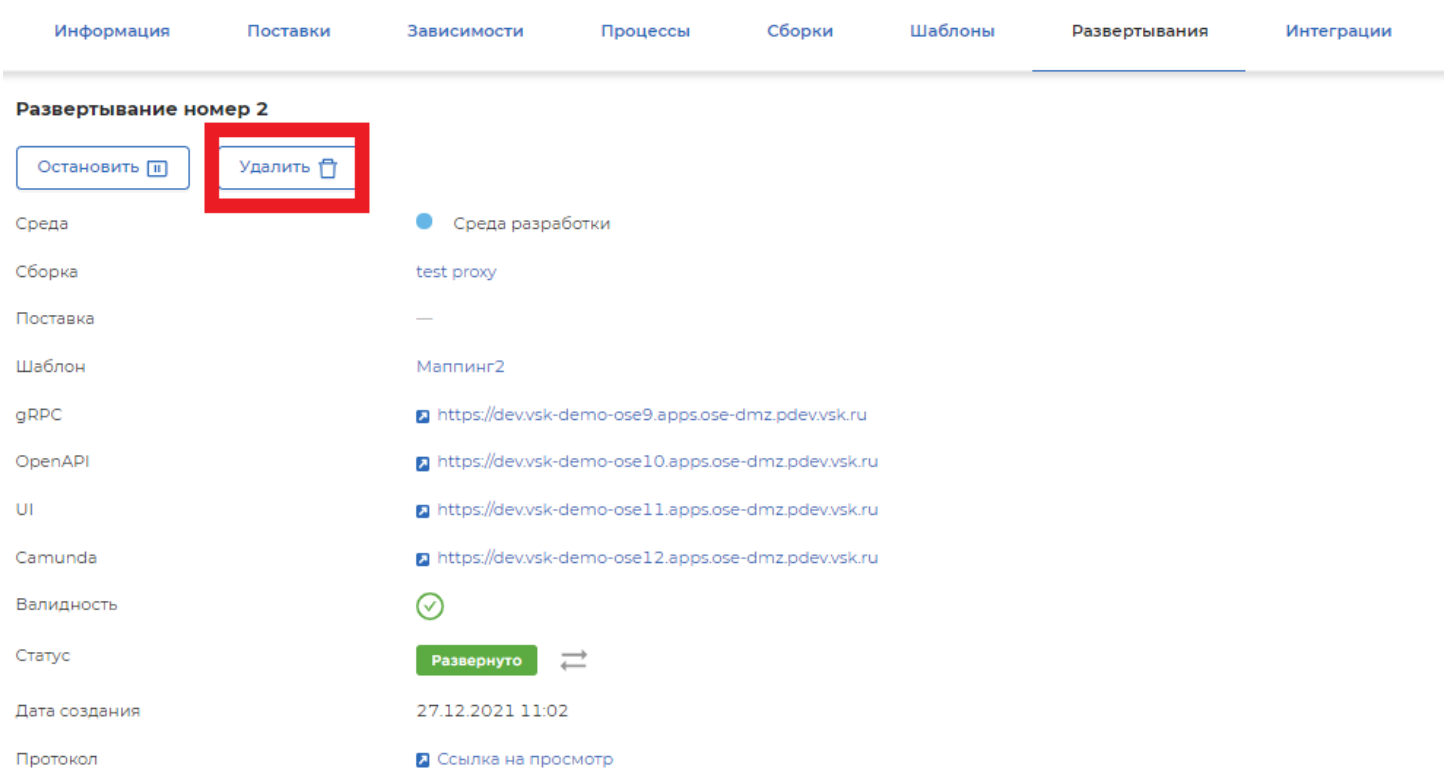
3.15 Очистка ресурсов

> ⓘ **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Предварительные действия: Не требуются

Удаление развертывания прикладного сервиса

```
Разрешено удалять в статусах:

- "Ошибка развертывания"
- "Развернуто"
```

1. Перейдите на вкладку "Сервисы приложений" и выберите необходимый сервис;
2. Перейдите на вкладку "Развертывания" и нажмите на названии требуемого развертывания;
3. Нажмите на кнопку "Удалить" в карточке развертывания;



4. Нажмите на кнопку "Ок".

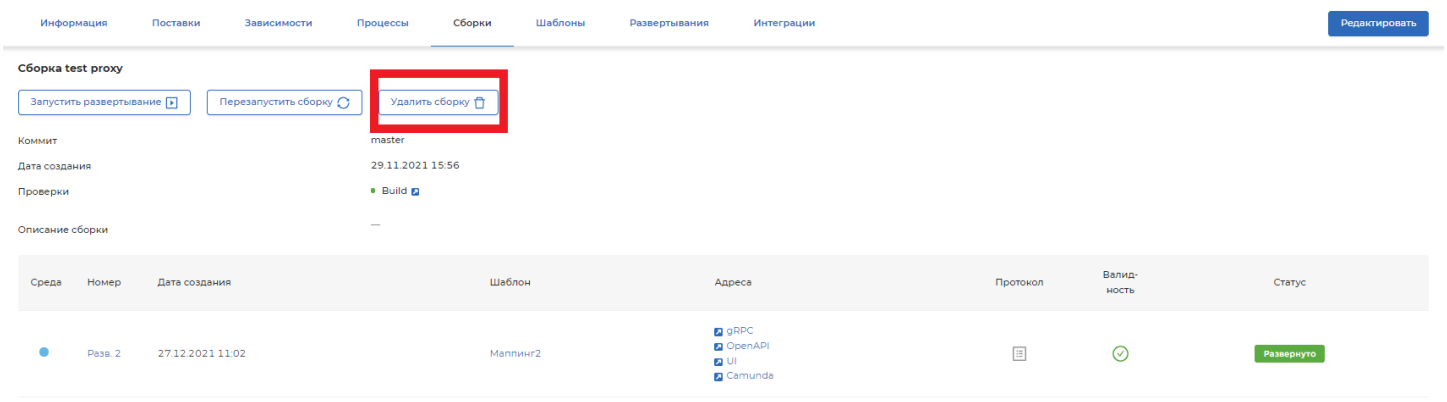
Удаление сборки

```
Разрешено удалять в статусах:

- "Ошибка"
- "Успешно"
```

1. Перейдите на вкладку "Сервисы приложений" и выберите необходимый сервис;

2. Перейдите на вкладку "Сборки" и нажмите на названии требуемой сборки;

3. Нажмите "Удалить сборку" в карточке сборки;



4. Нажмите на кнопку "Да".

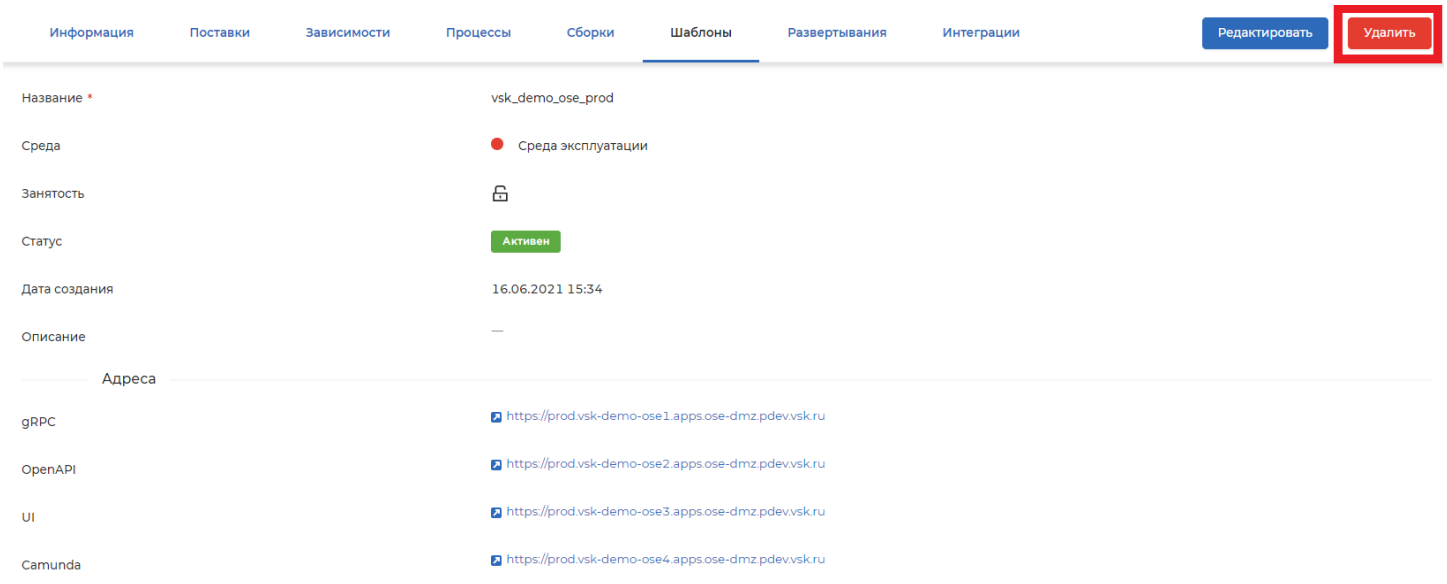
Удаление шаблона

```
Разрешено удалять в статусах:

- "Не занят"
```

1. Перейдите на вкладку "Сервисы приложений" и выберите необходимый сервис;

2. Перейдите на вкладку "Шаблоны" и нажмите на названии требуемого шаблона;

3. Нажмите "Удалить" в карточке шаблона;

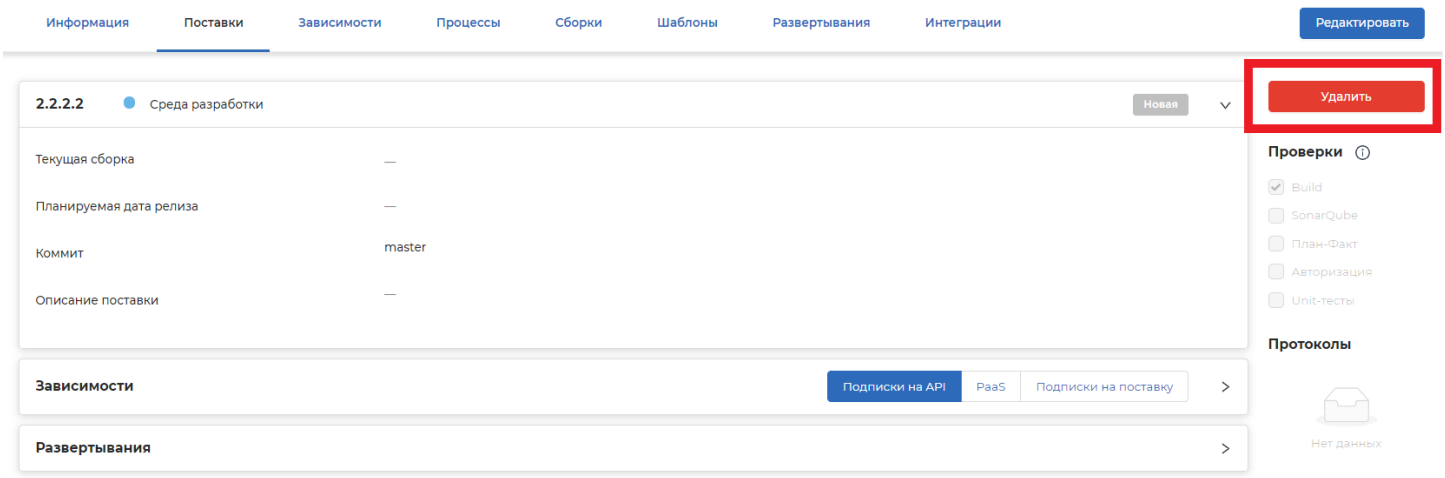


4. Нажмите на кнопку "Да".

Удаление поставки

Разрешено удалять, если в поставке нет активных развертываний сервиса и/или сборки.

1. Перейдите на вкладку "Сервисы приложений" и выберите необходимый сервис;
2. Перейдите на вкладку "Поставки" и нажмите на названии требуемой поставки;
3. Нажмите "Удалить" в карточке поставки;

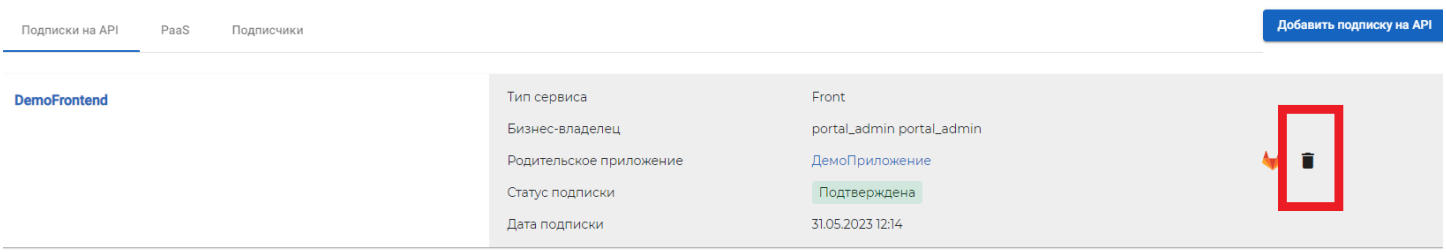


4. Нажмите на кнопку "Да".

Удаление зависимостей

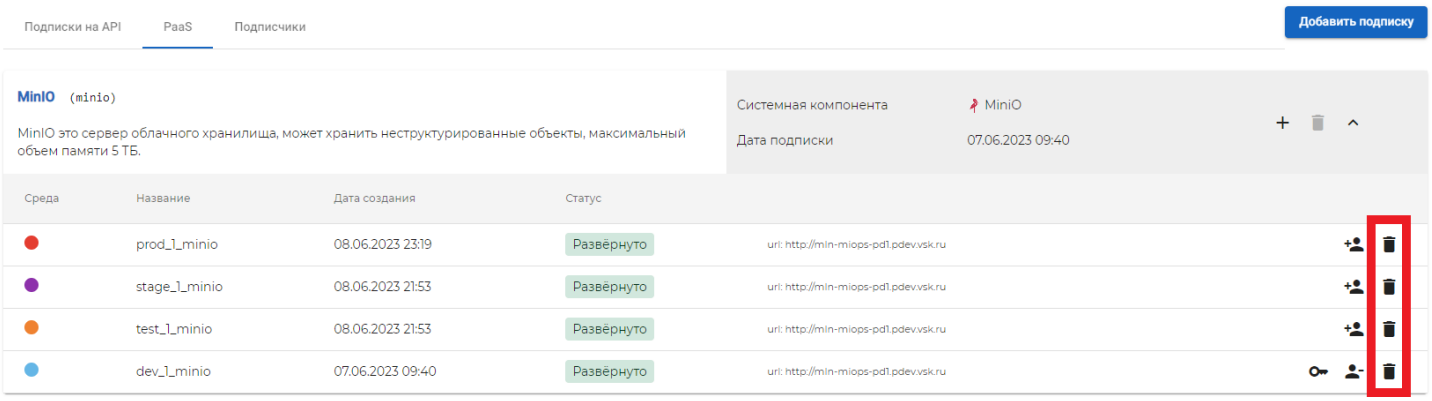
Разрешено удалять, если у сервиса нет активных развертываний.

1. Перейдите на вкладку "Сервисы приложений" и выберите необходимый сервис;
2. Перейдите на вкладку "Зависимости"-"Подписки на API";
3. Нажмите "Удалить" в строке выбранного сервиса;

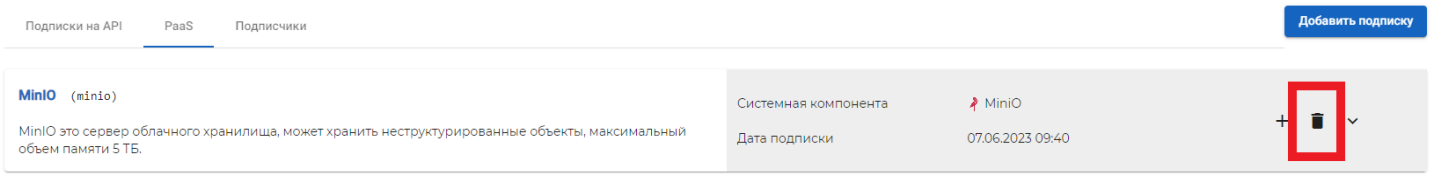


4. Перейдите на вкладку "Зависимости"-"PaaS";

5. Раскройте список развертываний выбранного PaaS-сервиса и удалите их;



6. Нажмите "Удалить" в строке выбранного PaaS-сервиса;



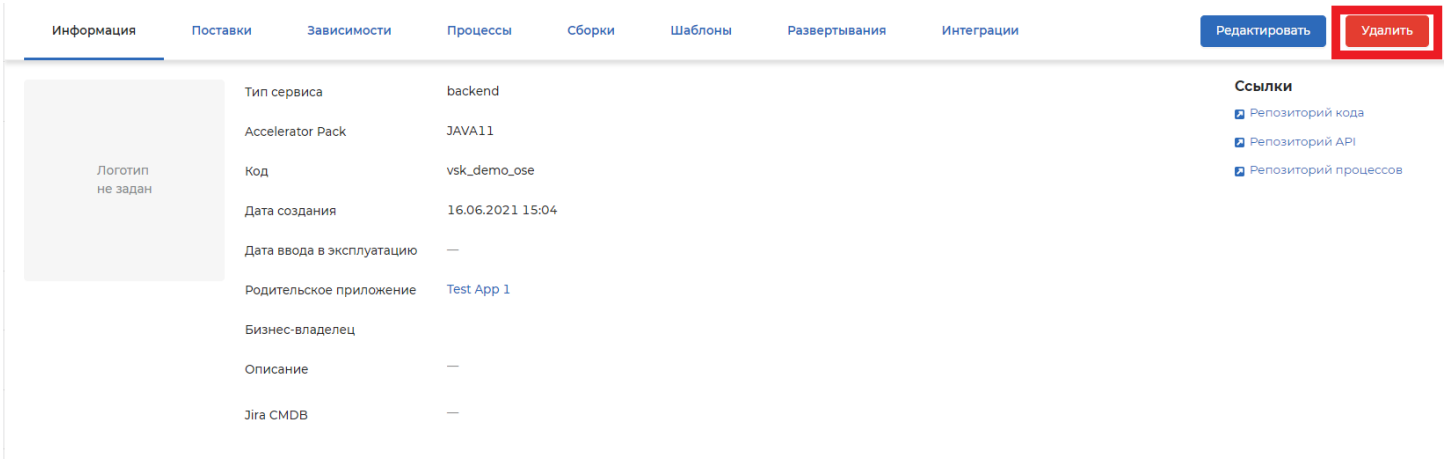
7. Нажмите на кнопку "Да".

Удаление сервиса

```
Разрешено удалять, если в нём удалены:

- Развертывания;
- Сборки;
- Зависимости PaaS;
- Сервисы-подписчики;
```

1. Перейдите на вкладку "Сервисы приложений" и выберите необходимый сервис;

2. Нажмите "Удалить" в карточке сервиса;



3. Нажмите на кнопку "Ок".

3.16 gRPC to JSON адаптер

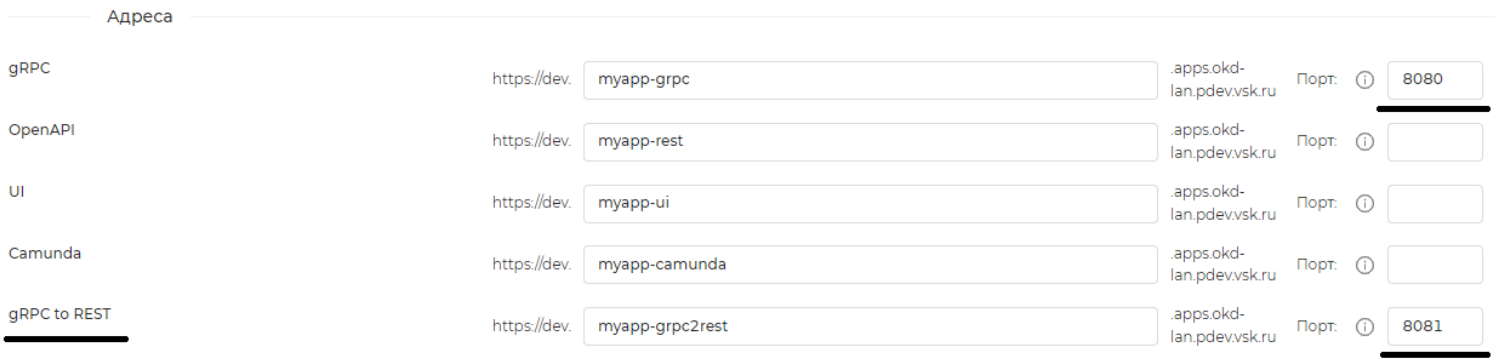
Marlin может автоматически предоставить для вашего gRPC сервиса proxy, реализующий конвертацию в JSON API.

Proxy-сервис построен на базе Envoy gRPC-JSON transcoder, автоматически настраивается на ваш gRPC-интерфейс.

Сигнатуры методов-прокси в точности совпадают с соответствующими gRPC запросами, тело запроса передаётся методом POST. Подробнее можно ознакомиться в документации gRPC-JSON transcoder

Требования:

- У сервиса в шаблоне разёртывания должен быть заполнен порт gRPC-интерфейса
- У сервиса в шаблоне должен быть заполнен порт gRPC-to-REST (*) интерфейса (и он должен не совпадать с иными открытыми приложением портам)
- У сервиса на gRPC-интерфейсе должна поддерживаться рефлексия (gRPC Reflection)



Если вы не используете универсальный helm-чарт, отдавая предпочтение самостоятельной реализации, то подобную автоматику можно получить, перенеся в свой чарт фрагмент, описывающий sidecar-контейнер:

```
spec:
  containers:
  - name: "envoy"
    image: {{ .Values.deployment.envoyImage | quote }}
    imagePullPolicy: Always
    readinessProbe:
```

```
      httpGet:
      path: /ready
      port: 9901
      initialDelaySeconds: 5
      periodSeconds: 10
    ports:
      - containerPort: {{ .Values.deployment.grpc2restPort }}
    env:
      - name: GRPC_PUBLIC_ENDPOINT
        value: {{ .Values.deployment.grpchost | quote }}
      - name: GRPC_UPSTREAM_PORT
        value: {{ .Values.deployment.grpcPort | quote }}
      - name: ENVOY_PORT
        value: {{ .Values.deployment.grpc2restPort | quote }}
```

3.17 Мониторинг

Платформа Marlin поддерживает сбор технических метрик по средствами Prometheus.

В своих приложения вы можете создавать экспортеры, метрики с которых будут аккумулироваться на платформенном серевере Prometheus. На основе собираемых метрик можно строить графиеские формы с помощью Grafana.

Подробную информацию о способах реализации метрик в своём приложении можно получить из официальной документации: https://prometheus.io/docs/instrumenting/clientlibs/. Пользователи Spring Boot могут использовать встроенные возможности фреймворка, описанные в документации.

Настроить сборк метрик для вашего микросервиса можно через блок "Мониторинг" в шаблоне развёртывания.

Для настройки вам потребуется:

1. Выбрать тип монитора:

    - `ServiceMonitor` - метрики, снимаемся с микросервиса вцелом. Например, "количество заказов". Запросы будут происходить на pod'ы в случайном

порядке согласно применяемой схеме балансировки.

- `PodMonitor` - метрики, снимаемые с каждого pod'а микросервиса. Например, "количетсво открытых соединений с СУБД". Запросы будут приходить на все поды параллельно, в базе данных prometheus будут сформированы серии значений по количеству pod'ов

Подробнее об отличиях между ServiceMonitor и PodMonitor можно прочитать в документации Prometheus Operator'a

2. Указать порт, на котором располагается web-сервис, передающий показатели. Порт может пересекаться или не пересекаться с портами, настроенными в шаблоне для публичного доступа.

3. Указать путь в URL к endpoint'у с метриками (типично, `/metrics` или `/actuator/prometheus`)

4. Указать частоту опроса метрки в формате `1h02m30s` (типично, `30s`)

3.18 Использование PaaS Keycloak

Введение

Пользовательская и межсервисная авторизация может быть реализована с помощью PaaS Keycloak.

Межсервисная и пользовательская реализуется с помощью отдельных сервисов PaaS, связанных с разными экземплярами системной компоненты Keycloak. Межсервисная авторизация необходима для управления доступами к подписанным прикладным сервисам. Пользовательская авторизация нужна для управления пользовательским доступом к приложениям.

Keycloak - это открытый (бесплатный) продукт для реализации аутентификации и авторизации, позволяющий создавать безопасные приложения и сервисы с минимальным написанием кода. Также Keycloak позволяет реализовать технологию единого входа.

Технология единого входа (англ. Single Sign-On) — технология, при использовании которой пользователь переходит из одного раздела портала в другой, либо из одной системы в другую, не связанную с первой системой, без повторной аутентификации.

Например, если на веб-портале существует несколько обширных независимых разделов (форум, чат, блог и т. д.) то, пройдя процедуру аутентификации в одном из сервисов, пользователь автоматически получает доступ ко всем остальным, что избавляет его от многократного ввода данных своей учётной записи.

В качестве стандарта взаимодействия используется стандарт OpenID Connect.

Подробную информацию о Keycloak можно получить в официальной документации

> **К СВЕДЕНИЮ**
>
> Realm — независимое пространство со своими клиентами.
>
> Client — сущность в Keycloak, создаваемая для прикладного сервиса. Client соответствует развертыванию PaaS.

> Роль - совокупность прав доступа. Роли сопоставляются с группами пользователей в ldap-каталоге: Active Directory или FreeIpa.
>
> С PaaS Keycloak могут использоваться только клиентские роли (настроенные для каждого клиента в отдельности), глобальные роли недоступны.

В Marlin существуют несколько PaaS системной компоненты Keycloak. Какой PaaS следует использовать зависит от того, кто будет выступать субъектом аутентификации для прикладного сервиса - другие сервисы или определенные группы пользователей.

В конкретной реализации платформы набор PaaS сервисов Keycloak может выглядеть так:

Назначение (Группа пользователей/сервисов)	**PaaS в Marlin**	**Realm (example)**
Межсервисная авторизация	Keycloak Межсервисная авторизация (LAN)	interservice_auth
Клиенты	Keycloak Клиенты	clients
Контрагенты: агенты + партнеры + другие юридические лица	Keycloak Сотрудники	employee

На Keycloak применены настройки:

1. Для всех Realm'ов, кроме interservice_auth* включены параметры anti-bruteforce:
   - блокировка после ввода 5 неправильных паролей на 15 минут
   - ограничение на частый повторный ввод паролей
   - аудит действий администратора Realm'а
2. Для всех не интегрированных через Identity Providers и User Federation - назначены политики сложности паролей:
   - для Realm Master
     - 12 символов в верхнем и нижнем регистре,
     - минимум одна цифра,
     - запрет использования в качестве нового одного из трех старых паролей,

        - отсутствие пароля в черном списке паролей.
    - для остальных Realm'ов
        - 8 символов в верхнем и нижнем регистре,
        - минимум одна цифра,
        - минимум один спецсимвол,
        - запрет использования в качестве нового одного из трех старых паролей,
        - отсутствие пароля в черном списке паролей.

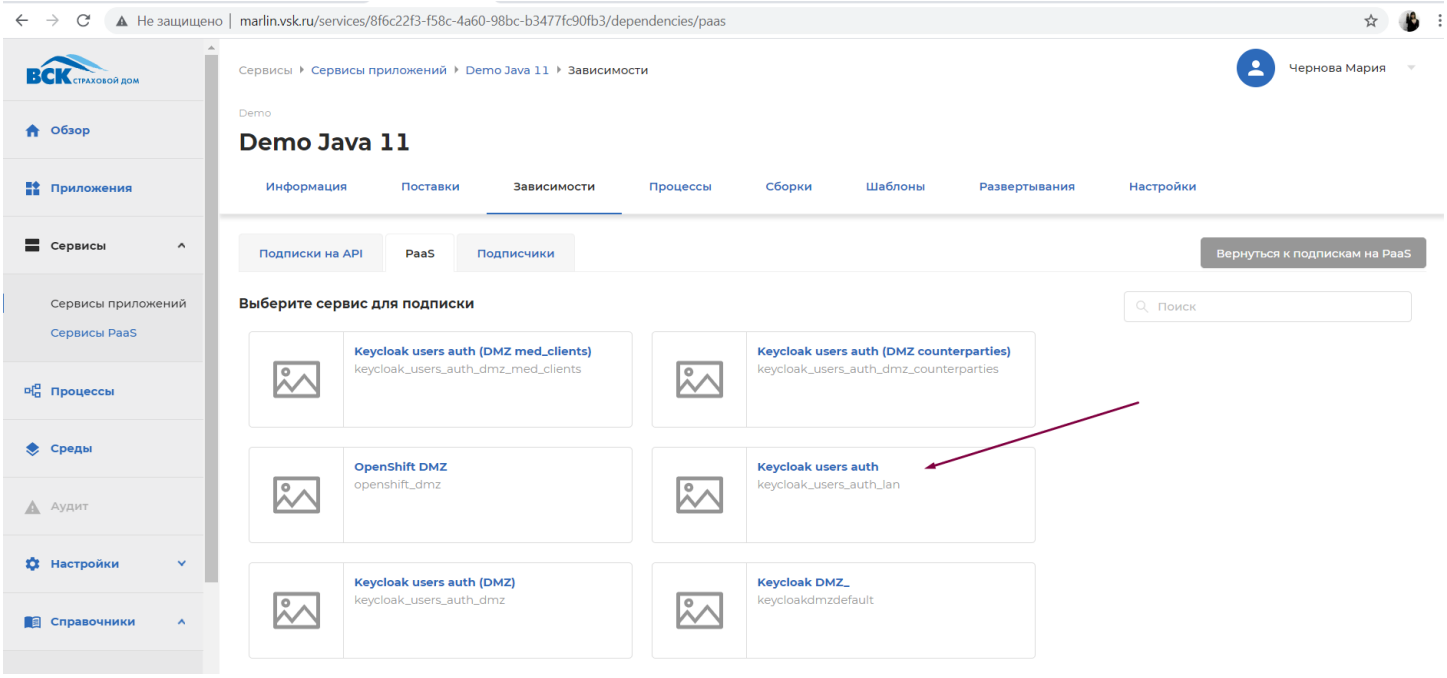
Настройка пользовательской авторизации

Предполагается, что в случае использования PaaS Keycloak разрабатываемый сервис будет обрабатывать запросы в соответствии с протоколом OpenID Connect, т.е следующим образом:

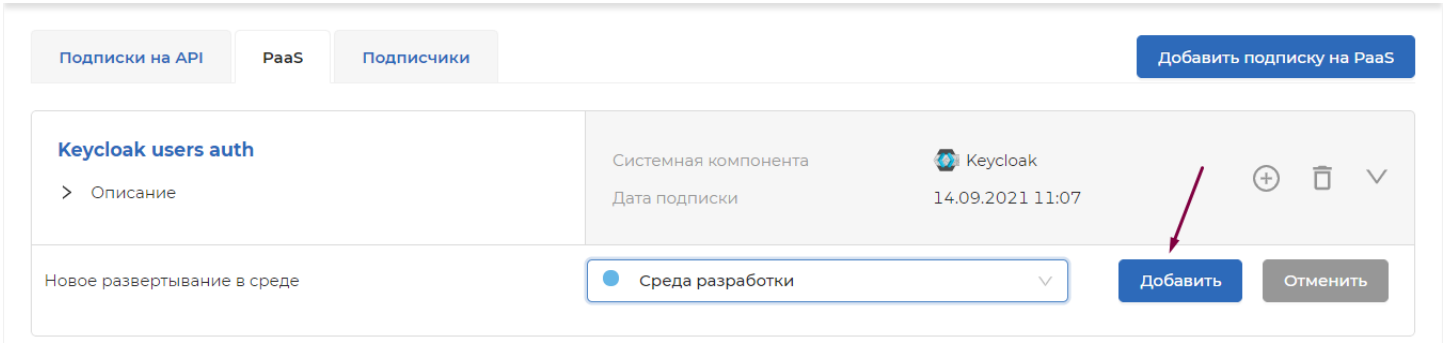
1. Неаутентифицированные пользователи перенаправляются на страницу Keycloak для аутентификации (ввод логина и пароля);
2. Для пользователей, прошедших аутентификацию в Keycloak, прикладной сервис получает access token, refresh token;
3. Для запросов в прикладной сервис используется access token;
4. Прикладной сервис должен валидировать access token при каждом запросе (проверяется срок действия, цифровая подпись, наличие прав на выполнение запроса);
5. При истечении срока действия access token, он может быть обновлен с помощью refresh token или пользователь становится неаутентифицированным;
6. Access token является JWT-токеном, т.е. несёт в себе информацию о пользователе и его ролях;
7. Прикладной сервис использует информацию о ролях для авторизации действий пользователя в сервисе.

Для реализации пользовательской авторизации необходимо:

1. Добавить подписку на PaaS



2. Выполнить развертывание PaaS на необходимых средах



После развертывания PaaS в шаблон развертывания прикладного сервиса возвращаются параметры созданного клиента и сервера Keycloak: URL, Realm, ClientID и ClientSecret.

3. Передать параметры подключения к Keycloak через таблицу маппинга переменных окружения в шаблонах развёртывания.

   При использовании custom'ного helm-чарта:

   Объявить параметры в values-файле

```
deployment:
  keycloakdefaultUrl: ""
  keycloakdefaultRealm: ""
```

```
  keycloakdefaultClientId: ""
  keycloakdefaultClientSecret: ""
```

где keycloakdefault - код используемого Keycloak

и монтировать их значения в контейнер приложения под необходимыми именами переменных окружения или через файловую систему.

Пример монтирования переменных окружения:

```
env:
  - name: KEYCLOAK_URL
    value: {{ .Values.deployment.keycloak_users_auth_lantUrl | quote }}
  - name: KEYCLOAK_REALM
    value: {{ .Values.deployment.keycloak_users_auth_lanRealm | quote }}
  - name: KEYCLOAK_CLIENT_ID
    value: {{ .Values.deployment.keycloak_users_auth_lanClientId | quote }}
  - name: KEYCLOAK_CLIENT_SECRET
    valueFrom:
      secretKeyRef:
        name: {{ .Values.deployment.finalName }}-password-secret
        key: keycloak-client-secret
```

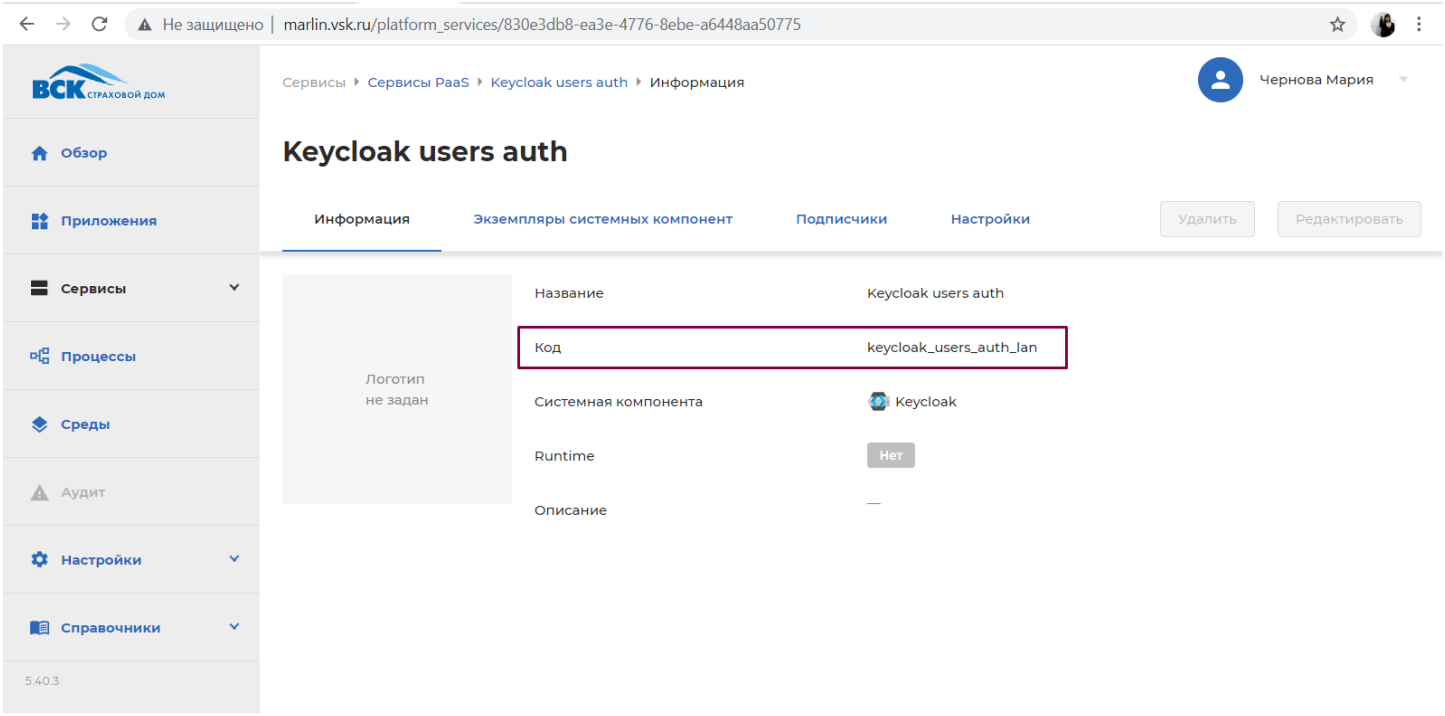
Значение ClientSecret не доступно непосредственно, а монтируется через kind: Secret из HashiCorp Vault

4. Сформировать 3 файла формата JSON: clients.json, roles.json, mapping.json

   Файлы нужно расположить в репозитории исходного кода сервиса, в папке, название которой соответствует коду PaaS Keycloak, на который подписан сервис, например: `keycloak_employee`.

   Если приложение не использует ролевую модель Keycloak для пользовательской авторизации, наличие файлов roles.json и mapping.json не обязательно.

   Код PaaS:



`clients.json` - файл с параметрами для настройки пользовательского клиента и соответственно способа авторизации пользователей в приложении.

```
{
  "enabled": true,
  "webOrigins": ["+"],
  "serviceAccountsEnabled": false,
  "publicClient": true
}
```

`roles.json` - файл с перечнем пользовательских ролей, определенных в приложении.

Обратите внимание! При создании ролевой модели должны использоваться только клиентские роли, глобальные роли не используются.

```
[
    {
        "name": "Admin",
        "composite": false,
        "clientRole": true,
        "attributes": {},
        "containerId": "client_id_default"
    },
    {
```

```
        "name": "Worker",
        "composite": false,
        "clientRole": true,
        "containerId": "client_id_default",
        "attributes": {}
    }
]
```

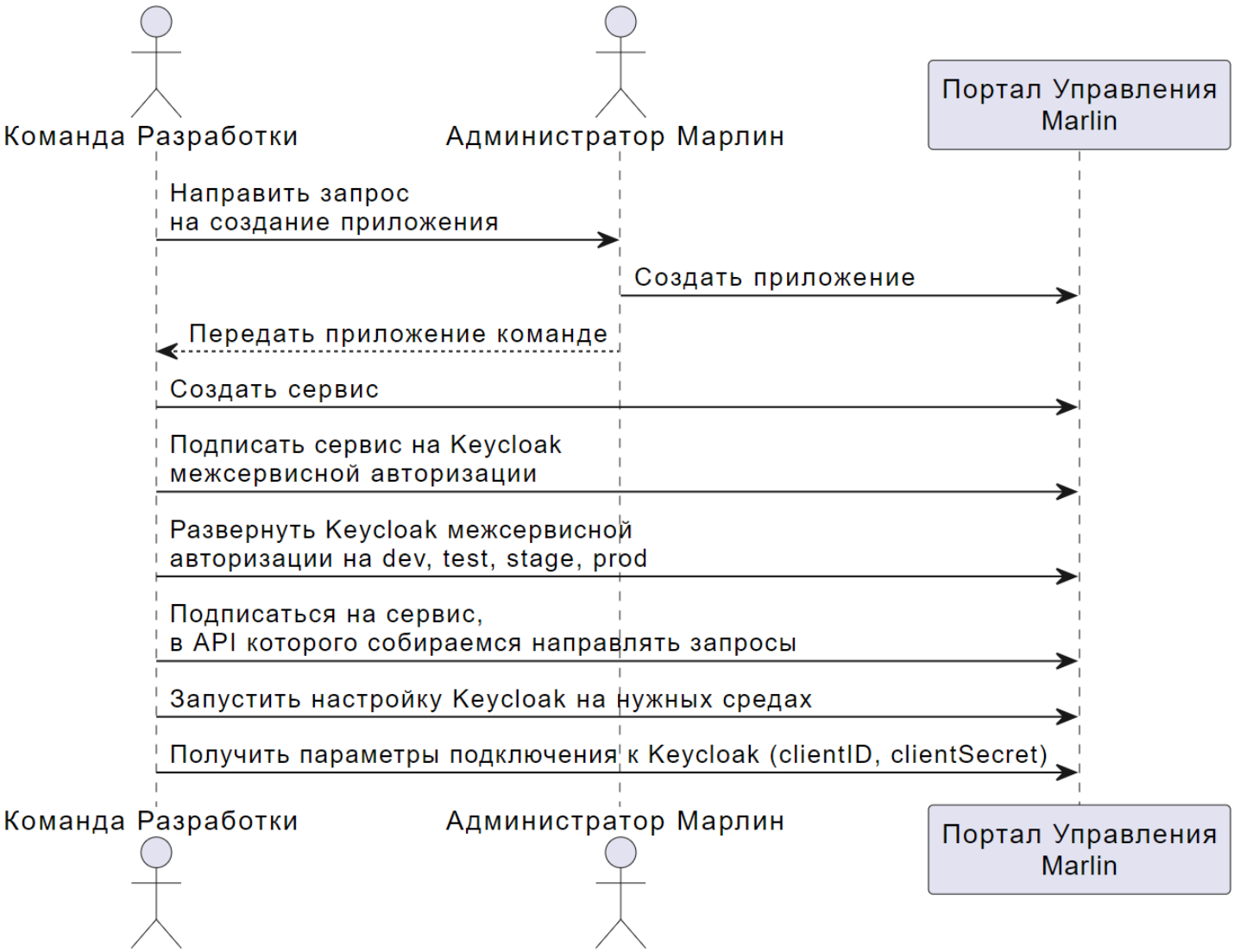
`mapping.json` - файл с маппингом групп пользователей из каталога LDAP пользовательским ролям в приложении. Группы заблаговременно определяются в AD (для внутренних пользователей) или FreeIPA (для внешних пользователей) эксплуатационным персоналом и переносятся в Keycloak через User Federation. Возможно задавать "среду" для маппинга - в этом случае, правило будет применяться только на этой среде (применимо, когда пользователи stage и prod берутся из одного каталога, но надо сделать так, чтобы роль у пользователя была только на середе staging)

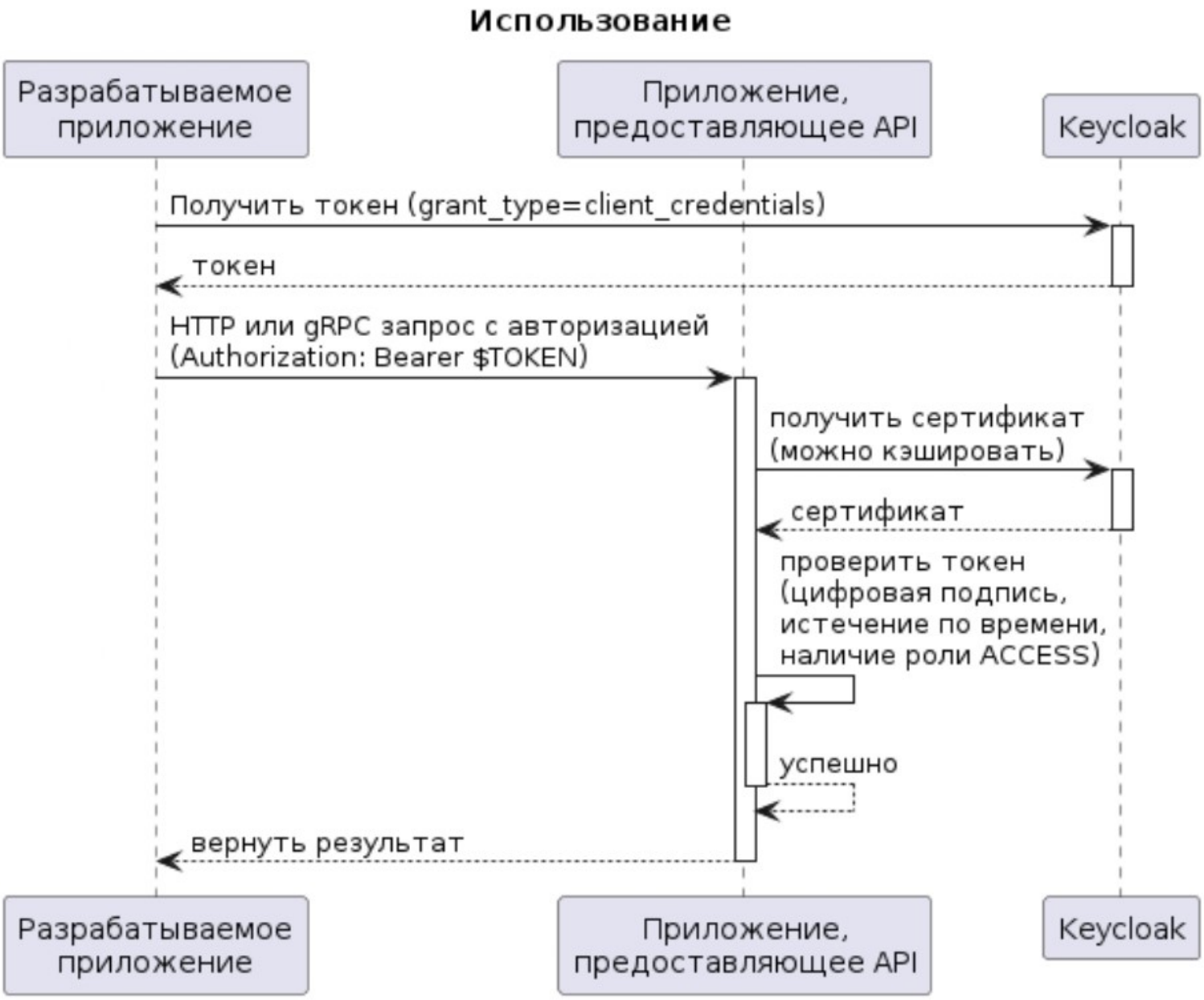
```
[
    {
        "role": "Admin",
        "group": "AdminGroup",
        "env": "stage"
    },
    {
        "role": "Worker",
        "group": "WorkerGroup"
    }
]
```

5. Выполнить развертывание своего сервиса

При реализации авторизации из веб приложения следует использовать библиотеку https://www.npmjs.com/package/keycloak-js

Настройка межсервисной авторизации


Команда Разработки
Администратор Марлин
Портал Управления Marlin
Направить запрос на создание приложения
Создать приложение
Передать приложение команде
Создать сервис
Подписать сервис на Keycloak межсервисной авторизации
Развернуть Keycloak межсервисной авторизации на dev, test, stage, prod
Подписаться на сервис, в API которого собираемся направлять запросы
Запустить настройку Keycloak на нужных средах
Получить параметры подключения к Keycloak (clientID, clientSecret)
Команда Разработки
Администратор Марлин
Портал Управления Marlin



Разработка

Рекомендованные вендором библиотеки приведены в документации https://www.keycloak.org/docs/latest/securing_apps/

Задачи клиента:

1. получить токен
2. направить в целевой сервис полезный запрос с токеном
3. хранить токен до 90% времени его жизни, перевыпустить при достижении

Задачи сервера:

1. получить сертификат (открытый ключ) из keycloak
2. проверить токен на валидность (срок истечения, цифровая подпись, наличие роли ACCESS с кодом сервиса)
3. оба этих пункта обычно делают клиентские библиотеки

Пример получения токена и использования его для вызова сервиса

```
#!/bin/bash

set -e

REALM=interservice_auth
CLIENT=client
CLIENT_SECRET=XXXX
GRANT_TYPE=client_credentials

# Fetch token
TOKEN=$(curl -s "http://keycloak_url/auth/realms/${REALM}/protocol/openid-
connect/token" \
  -d "grant_type=${GRANT_TYPE}&client_id=${CLIENT}&client_secret=${CLIENT_SECRET}"
| jq --raw-output ".access_token")

# Call service with authentication
curl -v -s http://service_url/ -H "Authorization: Bearer $TOKEN"
```

Предполагается, что в случае использования PaaS Keycloak разрабатываемый сервис будет обрабатывать запросы следующим образом:

1. Неаутентифицированный запрос завершается с ошибкой;
2. Для запросов с access token прикладной сервис валидирует access token при каждом запросе. Проверяется срок действия, подпись и наличие ACCESS роли клиента прикладного сервиса, принимающего запрос;
3. При истечении срока действия access token или отсутствия ACCESS роли, запрос возвращается с ошибкой.

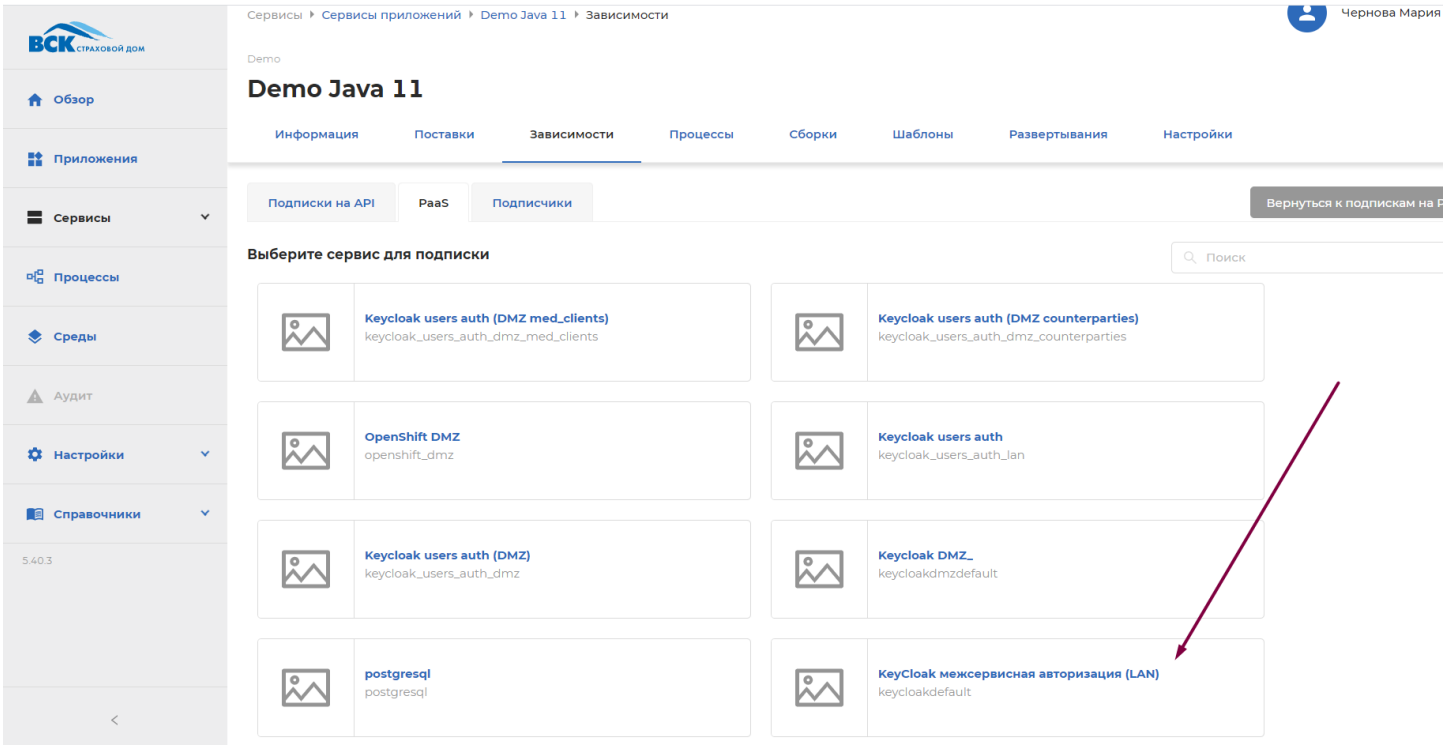
Для отправки запроса на авторизацию в связном сервисе:

1. Используя ClientId и ClientSecret прикладной сервис получает access token со списком ACCESS ролей прикладных сервисов, к которым может обращаться данный сервис;

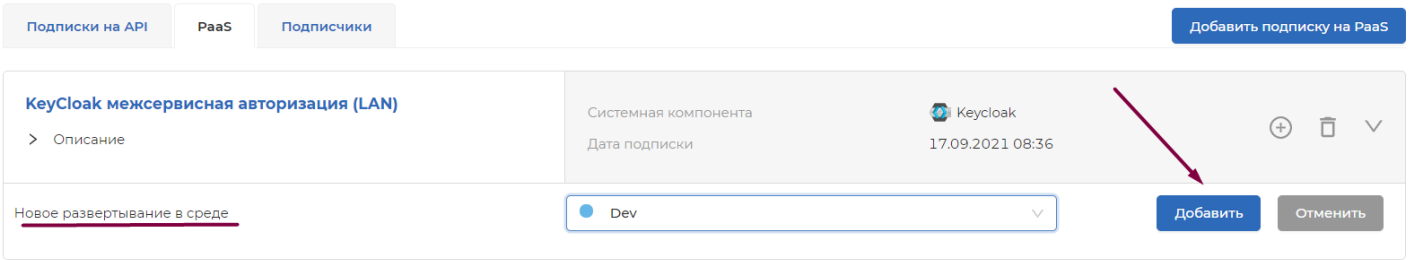
2. Все исходящие запросы прикладной сервис сопровождает полученным access token;
3. При истечении срока действия, прикладной сервис перевыпускает access token.

Чтобы реализовать межсервисную авторизацию с помощью PaaS Keycloak, необходимо:

1. Добавить подписку на PaaS



2. Выполнить развертывание PaaS



3. Сконфигурировать передачу параметров подключения к Keycloak в свой сервис

   В шаблоне развёртывания: добавить 4 переменные окружения, задать им необходимые имена и настроить их на соответствующие значения от PaaS Keycloak: URL, Realm, ClientID, ClientSecret

   Если вы используете свой helm chart:

```yaml
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
spec:
  ...
  template:
    spec:
      containers:
        - ...
          env:
          - name: KEYCLOAK_URL
            value: {{ .Values.deployment.keycloakdefaultUrl | quote }}
          - name: KEYCLOAK_REALM
            value: {{ .Values.deployment.keycloakdefaultRealm | quote }}
          - name: KEYCLOAK_CLIENT_ID
            value: {{ .Values.deployment.keycloakdefaultClientId | quote }}
          - name: KEYCLOAK_CLIENT_SECRET
            valueFrom:
              secretKeyRef:
                name: {{ .Values.deployment.finalName }}-password-secret
                key: keycloak-client-secret

---
# externalsecrets.yaml
apiVersion: kubernetes-client.io/v1
kind: ExternalSecret
metadata:
  name: {{ .Values.deployment.finalName }}-password-secret
  namespace: {{ .Values.deployment.namespace | quote }}
spec:
  backendType: vault
  data:
    - key: {{ .Values.deployment.keycloakdefaultClientSecret | quote }}  #путь
секрета в Vault
      name: keycloak-client-secret #имя поля в data секрета
      property: password # ключ в Vault
  kvVersion: 1
  vaultMountPoint: {{ .Values.deployment.clustertype | quote }}
  vaultRole: {{ .Values.deployment.namespace | quote }}
```

4. Выполнить развертывание своего сервиса

Обработка входящих запросов

Пользовательский запрос

Обработка пользовательского запроса происходит по алгоритмам протокола OpenID Connect. Прикладной сервис обрабатывает пользовательские запросы следующим образом:

1. Неаутентифицированные пользователи перенаправляются на страницу Keycloak для аутентификации (ввод логина и пароля);
2. Для пользователей прошедших аутентификацию в Keycloak прикладной сервис получает access token, refresh token и id token;
3. Для дальнейших запросов пользователь использует access token;
4. Прикладной сервис должен валидирует access token при каждом запросе (проверяется срок действия и подпись);
5. При истечении срока действия access token, он может быть обновлен с помощью refresh token или пользователь становится неаутентифицированным;
6. Внутри access token передается информация о пользователе и его ролях;
7. Прикладной сервис использует информацию о ролях для авторизации действий пользователя в сервисе.

Межсервисный запрос

Прикладной сервис обрабатывает запрос от сервисов следующим образом:

1. Неаутентифицированный запрос возвращается с ошибкой;
2. Для запросов с access token прикладной сервис валидирует access token при каждом запросе. Проверяется срок действия, подпись и наличие ACCESS роли клиента прикладного сервиса, принимающего запрос;
3. При истечении срока действия access token или отсутствия ACCESS роли, запрос возвращается с ошибкой.

Выполнение исходящих запросов

Для отправки запроса на авторизацию в связном сервисе:

1. Используя ClientId и ClientSecret прикладной сервис получает access token со списком ACCESS ролей прикладных сервисов, к которым может обращаться данный сервис;

2. Все исходящие запросы прикладной сервис сопровождает полученным access token;
3. При истечении срока действия, прикладной сервис перевыпускает access token.

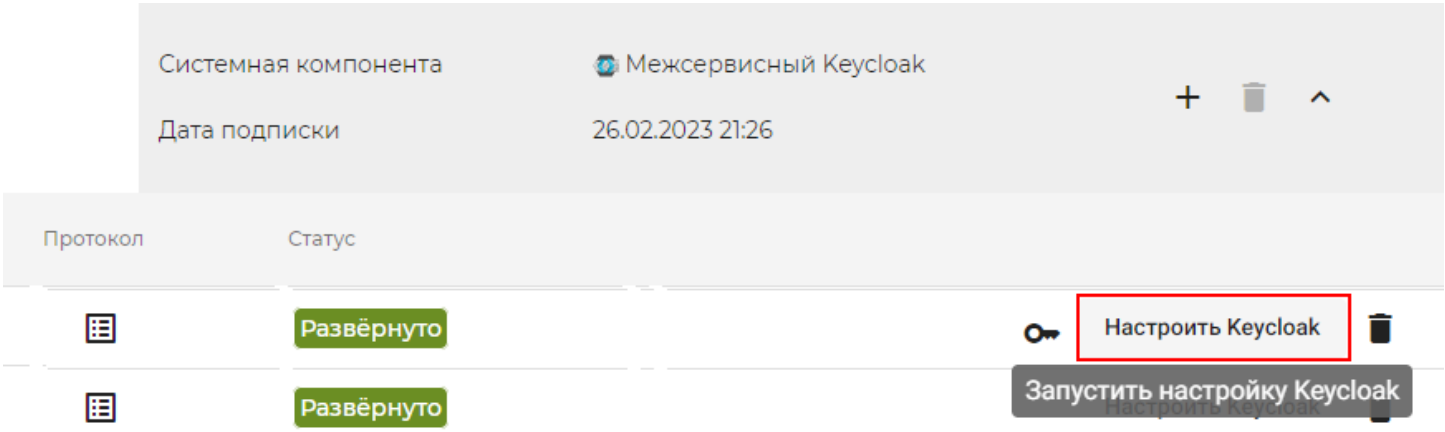
Пример реализации клиента с межсервисной аутентификацией

```
#!/bin/bash

set -e

REALM=interservice_auth
CLIENT=client
CLIENT_SECRET=XXXX
GRANT_TYPE=client_credentials

# Fetch token
TOKEN=$(curl -s "http://keycloak_url/auth/realms/${REALM}/protocol/openid-
connect/token" \
  -d "grant_type=${GRANT_TYPE}&client_id=${CLIENT}&client_secret=${CLIENT_SECRET}"
| jq --raw-output ".access_token")

# Call service with authentication
curl -v -s http://service_url/ -H "Authorization: Bearer $TOKEN"
```

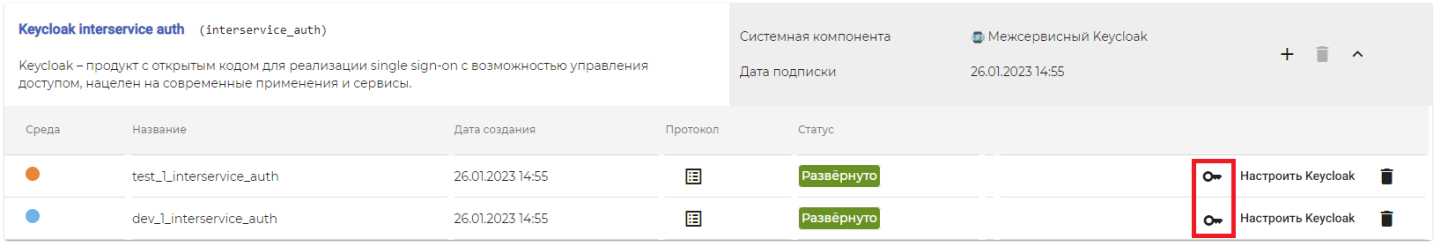
3.19 Получение ClientID и ClientSecret

> ⓘ **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации
>
> Данная инструкция предназначена для пользователей, которым необходимо создать сервис-заглушку в ПУ Marlin для реализации межсервисной аутентификации.

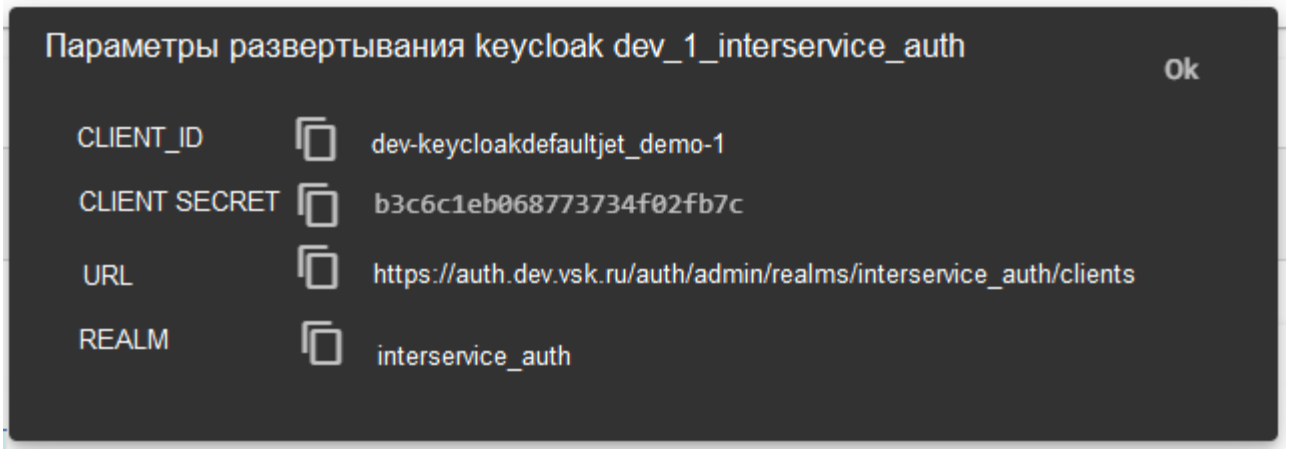
1. Открыть вкладку "Зависимости" → "Подписка на API" в карточке сервиса;
2. Нажать на кнопку "Добавить подписку на API";
3. Найти сервис-подписку, с которым вы собираетесь взаимодействовать. Подписаться на сервис (подробнее Межсервисная подписка прикладных сервисов);
4. Открыть вкладку "Зависимости"→ "PaaS" в карточке сервиса;
5. Нажать кнопку "Добавить подписку на PaaS";
6. Выбрать PaaS-сервис "Keycloak interservice auth", нажать на кнопку "Подписаться" (подробнее Подписка на платформенный сервис);
7. Выбрать среду для развёртывания и нажать на кнопку "Добавить";
8. Дождитесь перехода развёртывания в статус "Развернуто";
9. Нажмите на кнопку "Настроить Keycloak", которая запускает проливку ролевой модели;



10. Нажмите на кнопку "Ключ";



11. Скопируйте необходимые Параметры.



3.20 Helm-chart и секреты в Vault

Hashicorp Vault - инструмент для хранения секретов (паролей, токенов) портала и разворачиваемых на его мощностях приложений.

Доставка секрета до приложения делится на этапы:

1. Секрет порталом размещается в Vault;
2. В неймспейсе приложения создается сущность ExternalSecret, описывающая способ получения и место доставки секрета;
3. ExternalSecret создаёт Secret;
4. Содержимое Secret монтируется в Pod как переменная окружения или файл.

Ссылки:

- ExternalSecrets
- Hashicrop Vault

В работе с порталом есть три сценария, в которых встречаются секреты:

1. Секреты PaaS'ов (e.g. пароль к postgres);
2. Секреты Ingress'ов (TLS private key + certificate);
3. Секреты приложений (e.g. env variables).

Секреты PaaS'ов

1. Добавить в директорию charts/templates с репозиторием кода файл externalsecrets.yaml, пример:

```
apiVersion: kubernetes-client.io/v1
kind: ExternalSecret
metadata:
  name: {{ .Values.deployment.finalName }}-password-secret
  namespace: {{ .Values.deployment.namespace }}
spec:
  backendType: vault
  data:
    - key: {{ .Values.deployment.keycloakdefaultClientSecret | quote }}  #путь
```

```
секрета в Vault
      name: keycloak-client-secret #имя поля в data секрета
      property: password # ключ в Vault
    - key: {{ .Values.deployment.postgresqlPassword | quote }}  #путь секрета в
Vault
      name: postgresql-password #имя поля в data секрета
      property: password # ключ в Vault
  kvVersion: 1
  vaultMountPoint: {{ .Values.deployment.clustertype | quote }}
  vaultRole: {{ .Values.deployment.namespace | quote }}
```

2. В файле externalsecrets.yaml в блоке "data" указать переменную, которую использует приложение, например {{ .Values.deployment.keycloakdefaultClientSecret }}: **Фрагмент файла externalsecrets.yaml с секцией data**

```
data:
  - key: {{ .Values.deployment.keycloakdefaultClientSecret | quote }}
#переменная указывающая на путь секрета в Vault, например:
/v1/marlin/paas/dev/keycloak/demo_keycloak
    name: keycloak-client-secret #имя поля в secret data
    property: password # ключ в Vault
```

3. В файле deployment.yaml для переменной окружения KEYCLOAK_CLIENT_SECRET в поля secretKeyRef.name и secretKeyRef.key указать на соотвествующие metadata.name и spec.data.key[{{ .Values.deployment.keycloakdefaultClientSecret }}].name файла externalsecrets.yaml: **Фрагмент файла deployment.yaml с секцией env**

```
- name: KEYCLOAK_CLIENT_SECRET
  valueFrom:
    secretKeyRef:
      name: {{ .Values.deployment.finalName }}-password-secret
      key: keycloak-client-secret
```

Секреты Ingress'ов (UI, Camunda, OpenAPI)

1. В директорию chart/templates репозитория кода загрузить необходимые файлы для секретов cert-externalsecret-ui.yaml, cert-externalsecret-openapi.yaml, cert-

externalsecret-camunda.yaml, cert-externalsecret-grpc.yaml для приложения. Примеры файлов:

cert-externalsecret-ui.yaml

```
apiVersion: kubernetes-client.io/v1
kind: ExternalSecret
metadata:
  name: {{ .Values.deployment.secretNameUI | quote }}
  namespace: {{ .Values.deployment.namespace }}
spec:
  backendType: vault
  template:
    type: kubernetes.io/tls
  data:
    - key: {{ .Values.deployment.storagename }}/apps/{{
.Values.deployment.envcode }}/{{ .Values.deployment.namespace }}/{{
.Values.deployment.finalName }}/certs/{{ .Values.deployment.finalName }}-ui
      name: tls.crt  #имя поля в data секрета
      property: crt
    - key: {{ .Values.deployment.storagename }}/apps/{{
.Values.deployment.envcode }}/{{ .Values.deployment.namespace }}/{{
.Values.deployment.finalName }}/certs/{{ .Values.deployment.finalName }}-ui
      name: tls.key  #имя поля в data секрета
      property: key
  kvVersion: 1
  vaultMountPoint: {{ .Values.deployment.clustertype | quote }}
  vaultRole: {{ .Values.deployment.namespace | quote }}
```

cert-externalsecret-camunda.yaml

```
apiVersion: kubernetes-client.io/v1
kind: ExternalSecret
metadata:
  name: {{ .Values.deployment.secretNameCamunda | quote }}
  namespace: {{ .Values.deployment.namespace }}
spec:
  backendType: vault
  template:
    type: kubernetes.io/tls
  data:
    - key: {{ .Values.deployment.storagename }}/apps/{{
.Values.deployment.envcode }}/{{ .Values.deployment.namespace }}/{{
.Values.deployment.finalName }}/certs/{{ .Values.deployment.finalName }}-
```

```
camunda
      name: tls.crt  #имя поля в data секрета
      property: crt
    - key: {{ .Values.deployment.storagename }}/apps/{{
.Values.deployment.envcode }}/{{ .Values.deployment.namespace }}/{{
.Values.deployment.finalName }}/certs/{{ .Values.deployment.finalName }}-
camunda
      name: tls.key  #имя поля в data секрета
      property: key
  kvVersion: 1
  vaultMountPoint: {{ .Values.deployment.clustertype | quote }}
  vaultRole: {{ .Values.deployment.namespace | quote }}
```

cert-externalsecret-openapi.yaml

```
apiVersion: kubernetes-client.io/v1
kind: ExternalSecret
metadata:
  name: {{ .Values.deployment.secretNameOpenApi | quote }}
  namespace: {{ .Values.deployment.namespace }}
spec:
  backendType: vault
  template:
    type: kubernetes.io/tls
  data:
    - key: {{ .Values.deployment.storagename }}/apps/{{
.Values.deployment.envcode }}/{{ .Values.deployment.namespace }}/{{
.Values.deployment.finalName }}/certs/{{ .Values.deployment.finalName }}-
openapi
      name: tls.crt  #имя поля в data секрета
      property: crt
    - key: {{ .Values.deployment.storagename }}/apps/{{
.Values.deployment.envcode }}/{{ .Values.deployment.namespace }}/{{
.Values.deployment.finalName }}/certs/{{ .Values.deployment.finalName }}-
openapi
      name: tls.key  #имя поля в data секрета
      property: key
  kvVersion: 1
  vaultMountPoint: {{ .Values.deployment.clustertype | quote }}
  vaultRole: {{ .Values.deployment.namespace | quote }}
```

2. В файле ingress.yaml в tls.secretName добавить значение на metadata.name tls-сертификатом. Пример для файла cert-externalsecret-camunda.yaml:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
name: {{ .Values.deployment.finalName }}-camunda-ui-ingress
namespace: {{ .Values.deployment.namespace }}
spec:
  rules:
    - host: {{ .Values.deployment.camundahost | quote }}
      http:
        paths:
        - backend:
            serviceName: {{ .Values.deployment.finalName }}-camunda-ui
            servicePort: 8080
            path: /
        - backend:
            serviceName: service-hub-actuator-service
            servicePort: 8094
            path: /actuator
  tls:
    - secretName: {{ .Values.deployment.secretNameCamunda | quote }}
      hosts:
        - {{ .Values.deployment.camundahost | quote }}
```

Секреты gRPC (TLS-сертификат)

gRPC работает отличным от остальных протоколов способом: терминация TLS трафик происходит на конечном приложении. Это означает, что приложение должно уметь поднимать https-порт с ключевой парой, предоставленной платформой.

Платформа предоставляет 2 формата private key: PCKS8 (преимущественно Java) и PEM (весь остальной мир)

В варианте PCKS8 ключ и сертификат будут доступны по путям /tls-key.pcks8 и /tls.crt соответственно, если их смонтировать следующим образом:

```
# cert-externalsecret-grpc.yaml
kind: Secret
apiVersion: v1
metadata:
  name: {{ .Values.deployment.finalName }}-grpc-pkcs8
data:
  tls.crt: {{ .Values.deployment.grpc.tls.certificateChain | b64enc | quote }}
```

```
  tls-key.pcks8: {{ .Values.deployment.grpc.tls.privateKeyPcks8 | b64enc | quote }}
type: Opaque

---
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      volumes:
        - name: grpc-tls
          secret:
            secretName: {{ .Values.deployment.finalName }}-grpc-pkcs8
      containers:
        - name: {{ .Values.deployment.finalName | quote }}
          ports:
            - name: grpc
              containerPort: 9091
          volumeMounts:
            - mountPath: "/tls-key.pkcs8"
              name: grpc-tls
              readOnly: true
              subPath: "tls-key.pkcs8"
            - mountPath: "/tls.crt"
              name: grpc-tls
              readOnly: true
              subPath: "tls.crt"
```

В варианте PEM ключ и сертификат будут доступны по путям /tls.key и /tls.crt соответственно, если их смонтировать следующим образом:

```
# cert-externalsecret-grpc.yaml
apiVersion: kubernetes-client.io/v1
kind: ExternalSecret
metadata:
  name: {{ .Values.deployment.secretNameGrpc | quote }}
spec:
  backendType: vault
  template:
    type: kubernetes.io/tls
  data:
    - key: {{ .Values.deployment.storagename }}/apps/{{ .Values.deployment.envcode
}}/{{ .Values.deployment.namespace }}/{{ .Values.deployment.finalName }}/certs/{{
.Values.deployment.finalName }}-grpc
```

```
      name: tls.crt  #имя поля в data секрета
      property: crt
    - key: {{ .Values.deployment.storagename }}/apps/{{ .Values.deployment.envcode
}}/{{ .Values.deployment.namespace }}/{{ .Values.deployment.finalName }}/certs/{{
.Values.deployment.finalName }}-grpc
      name: tls.key  #имя поля в data секрета
      property: key
  kvVersion: 1
  vaultMountPoint: {{ .Values.deployment.clustertype | quote }}
  vaultRole: {{ .Values.deployment.namespace | quote }}

---
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      volumes:
        - name: grpc-tls
          secret:
            secretName: {{ .Values.deployment.secretNameGrpc | quote }}
      containers:
        - name: {{ .Values.deployment.finalName | quote }}
          ports:
            - name: grpc
              containerPort: 9091
          volumeMounts:
            - mountPath: "/tls.key"
              name: grpc-tls
              readOnly: true
              subPath: "tls.key"
            - mountPath: "/tls.crt"
              name: grpc-tls
              readOnly: true
              subPath: "tls.crt"
```

В обоих случаях вместо Ingress для публикации следует использовать Route, сконфигурированный на проброс TLS трафика до приложения

```
# route.yaml
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: {{ .Values.deployment.finalName }}-grpc-route
```

```
spec:
  host: {{ .Values.deployment.grpchost | quote }}
  to:
    kind: Service
    name: {{ .Values.deployment.finalName }}-grpc-service
    weight: 100
  port:
    targetPort: 9091
  tls:
    termination: passthrough
  wildcardPolicy: None
```

Секреты приложений

1. В директории chart/templates создать файл, например, externalsecrets.yaml, следующего содержания:

   externalsecret.yaml

   ```
   apiVersion: kubernetes-client.io/v1
   kind: ExternalSecret
   metadata:
     name: {{ .Values.deployment.finalName }}-some-secret
   spec:
     backendType: vault
     data:
       - key: {{ .Values.secret_variables.SOME_SECRET | quote }}  # путь секрета в
   Vault
         property: value # ключ в Vault
         name: some-secret # имя поля в data создаваемого секрета
     kvVersion: 1
     vaultMountPoint: {{ .Values.deployment.clustertype | quote }}
     vaultRole: {{ .Values.deployment.namespace | quote }}
   ```

   , где SOME_SECRET - имя секретного параметра в шаблонах. Для нескольких секретов можно использовать один файл.

   ExternalSecret автоматически создаст Secret, значения из которого можно (и нужно) смонтировать в Deployment вашего приложения

2. В kind: Deployment вашего сервиса впишите следующие инструкции:

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
        - env:
            - name: APP_VARIABLE_NAME
              valueFrom:
                secretKeyRef:
                  name: {{ .Values.deployment.finalName }}-some-secret
                  key: some-secret
```

, где APP_VARIABLE_NAME - имя переменной окружения, ожидаемой приложением.

Распределение pod'ов по worker'ам

Для обеспечения большей надёжности приложению обычно устанавливают количество реплик равное трём (replicas=3).

Этого не достаточно: механизм не гарантирует, что все 3 копии не окажутся на одном worker'е.

Чтобы получить и это качество, необходимо добавить следующие инструкции:

podAntiaffinity

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  replicas: {{ .Values.deployment.replicas | default 1 }}
  template:
    metadata:
      labels:
        app: {{ .Values.deployment.finalName | quote }}
    spec:
      ...
      affinity:
        podAntiAffinity:
```

```
        preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              topologyKey: "kubernetes.io/hostname"
              labelSelector:
                matchExpressions:
                  - key: app
                    operator: In
                    values:
                      - {{ .Values.deployment.finalName | quote }}
            weight: 100
```

Это реализовано и работает автоматически для приложений, запущенных на универсальном helm-chart'e

На самописном helm-chart'e это необходимо делать самостоятельно.

3.21 Взаимодействие с PaaS PostgreSQL

Заказ

Функциональность БД PostgreSQL на Платформе "Marlin" предоставляется с помощью сервиса PaaS. Сервисов PaaS, связанных с системной компонентой PostgreSQL, может быть несколько.

Заказ БД заключается в добавлении подписки на сервис PaaS в карточке прикладного сервиса на вкладке "Зависимости", подвкладке "PaaS". При наличии нескольких сервисов PaaS, можно подписаться несколько раз. В таком случае прикладному сервису будет доступно несколько БД одновременно в период исполнения.

Создание

Развертывания PaaS на средах запускаются по команде пользователя с Портала управления, создаются БД. Также можно выполнить несколько развертываний на одной среде для параллельного тестирования разных развертываний на разных шаблонах прикладного сервиса.

Выбор

При развертывании прикладного сервиса в карточке шаблона развертывания разработчику необходимо выбрать конкретное развертывание сервиса PaaS для каждой из подписок, в случае если развертываний было несколько.

Проливка

Для проливки БД используется команда **liquibase:update** в рамках процесса развертывания прикладного сервиса. Данная команда получает следующие параметры:

- -Djdbc.url
- -Djdbc.user
- -Djdbc.password

Эти параметры доступны как **changelog Parameters** и их можно подставлять используя синтаксис **${parameter-name}**: https://docs.liquibase.com/concepts/basic/changelog-parameters.html.

Для примера выше параметры -D* заданы через следующие переменные доступные разработчику в ПУ в шаблоне развертывания для каждой подписки на PaaS PostgreSQL:

- URL_PARAM_NAME = jdbc.url
- USER_PARAM_NAME = jdbc.user
- PASSWORD_PARAM_NAME = jdbc.password

Также разработчику доступен универсальный параметр, который добавляется в конец команды **liquibase:update**:

- ADDITIONAL_PARAMS

Этот параметр позволяет указать любые дополнительные параметры для проливки, например: ADDITIONAL_PARAMS="-f db-folder" позволяет указать директорию в которой **liquibase:update** будет выполняться.

В итоге для каждой подписки разработчик должен указать в шаблоне развертывания переменные URL_PARAM_NAME, USER_PARAM_NAME, PASSWORD_PARAM_NAME, ADDITIONAL_PARAMS. Таким образом разработчик может управлять подпиской на PaaS PostgreSQL и использовать ее в скриптах проливки.

В рамках проливки допускается использовать только код который можно выполнить из под УЗ владельца (owner) базы. Например установка расширений (extensions) в postgres <13 не допускается, так как требует прав superuser.

Использование в среде исполнения

Параметры подключения к БД передаются в команду **helm install** как переменные helm-chart через параметр **--set**:

- deployment.postgredefaultHost - URL для подключения
- deployment.postgredefaultUser - пользователь
- deployment.postgredefaultPassword - пароль

Причем **postgredefault** в примере - это название сервиса PaaS на который подписан прикладной сервис. Если подписок несколько - то таких наборов будет несколько.

Разработчик должен декларировать эти переменные в файле **values.yml**. Тогда эти переменные будут доступны в шаблонах helm-chart.

3.22 Настройка probe check для приложений

Необходимо прописать проверку в коде приложения (файл development) в соответствии с примером ниже:

```
  startupProbe:
    tcpSocket:
      port: 8080
    failureThreshold: 30
    periodSeconds: 10
  livenessProbe:
    httpGet:
      path: /actuator/health/liveness
      port: 8080
    initialDelaySeconds: 3
    periodSeconds: 3
  readinessProbe:
    httpGet:
      path: /actuator/health/readiness
      port: 8080
    initialDelaySeconds: 5
    periodSeconds: 10
```

Ссылка на документацию: https://docs.openshift.com/container-platform/4.6/applications/application-health.html

Пример реализации для Spring Boot: Spring Boot Actuator

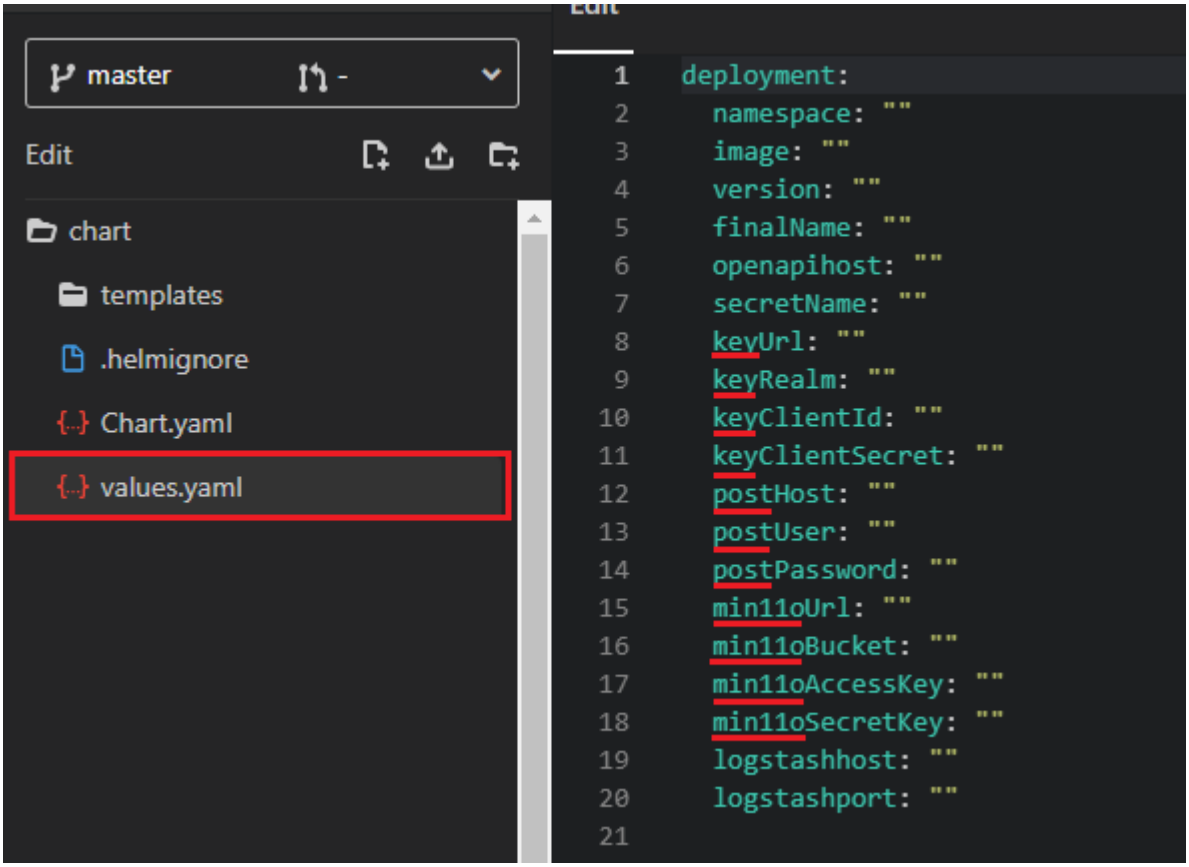
3.23 Инструкция по передаче переменных окружения в скрипты развертывания

> ⓘ **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик/Сотрудник эксплуатации

Добавление параметров подписок в скрипты развертывания

1. Добавить подписку на PaaS или на другой прикладной сервис в карточке прикладного сервиса;

2. В папке chart - файл values.yaml указать **код PaaS-сервисов**, на которые подписан прикладной сервис (чтобы посмотреть код, нужно перейти на вкладку "Зависимости" карточки сервиса → PaaS → нажать на название PaaS-сервиса → перейти в карточку выбранного PaaS-сервиса):

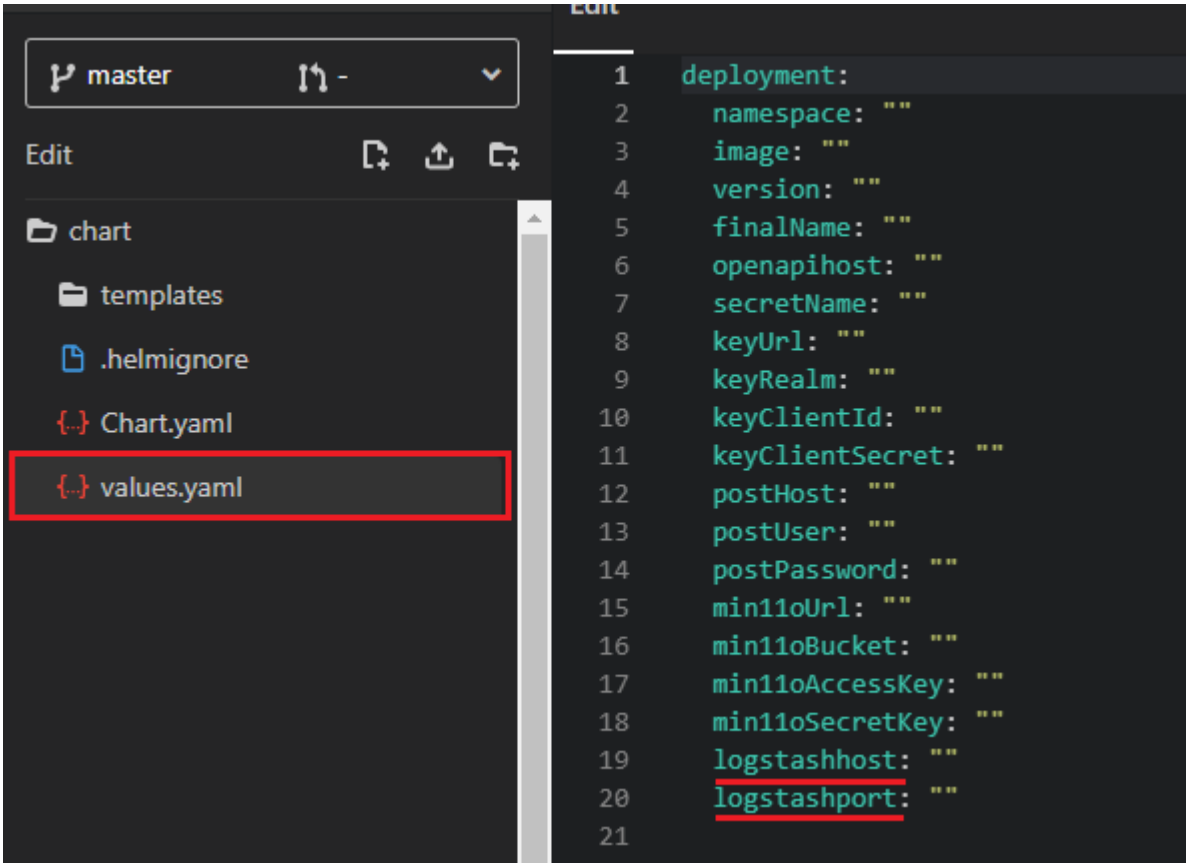


3. В папке chart - templates - файл deployment.yaml указать **код PaaS-сервисов** в значения ключей переменных окружений, например: value: {{ .Values.deployment.keyClientSecret }}, где key - это код PaaS-сервиса, на который подписан прикладной сервис

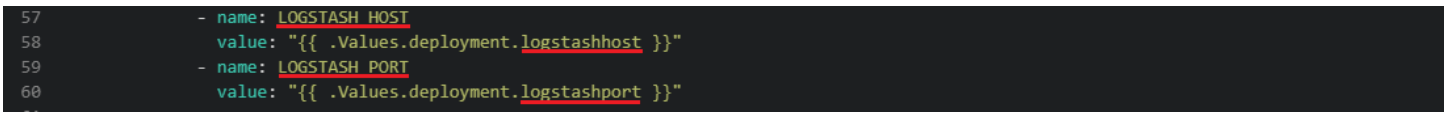
Добавление переменных окружения в скрипты развертывания

1. Добавить переменные окружения в шаблон развертывания сервиса;

2. В папке chart - файл values.yaml указать **ключи переменных окружений**, которые были указаны при создании шаблона развертывания:



3. В папке chart - templates - файл deployment.yaml указать **ключи** и **значения** переменных окружения, например:



3.24 Квоты и Масштабирование

Квоты

Пример **Limits & Requests**

```yaml
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
        - name: application
          image: redis:5.0.3-alpine
          resources:
            limits:
              memory: "600Mi"
              cpu: "1000m"
            requests:
              memory: "100Mi"
              cpu: "100m"
```

Limits определяет предельный объем ресурсов, выделяемых одному поду. Определение этого ограничения позволяет избежать незапланированного потребления ресурсов и нанесения ущерба соседям по процессору, например, в случае утечек памяти.

Requests определяет минимальный объем ресурсов, который должен быть зарезервирован для Pod'а. Это позволяет осуществлять планирование и вовремя масштабировать вычислительные мощности кластера.

1000m - эквивалент 1 ядру процессора. (миллиядра)

Более подробную информацию можно получить из официальной документации

Типовые конфигурации

В частных случаях, зная свой сервис, от этих значений надо отклоняться, как в большую, так и в меньшую сторону.

Java

Java Limits & Requests

```yaml
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
        - name: application
          image: openjdk:11
          resources:
            limits:
              memory: "1500Mi"
              cpu: "1000m"
            requests:
              memory: "300Mi"
              cpu: "100m"
```

NodeJS

NodeJS Limits & Requests

```yaml
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
        - name: application
          image: nodejs:16
          resources:
            limits:
              memory: "500Mi"
              cpu: "1000m"
            requests:
              memory: "300Mi"
              cpu: "10m"
```

Масштабирование

В типовой ситуации, микросервис должен запускаться в 3 копиях для обеспечения отказоустойчивости и бесшовного разворачивания релизов.

Для экономии ресурсов тестовых сред, приветствуется ограничение количества реплик до одной. Кроме того, это делает возможным наблюдать логи контейнера в одном окне/терминале.

Limits & Requests

```
apiVersion: apps/v1
kind: Deployment
spec:
  strategy:
    type: RollingUpdate
  {{- if contains .Values.deployment.envcode "stage,prod" }}
  replicas: 3
  {{- else }}
  replicas: 1
  {{- end }}
  template:
    spec:
      containers:
        - ...
```

3.25 Интеграция с GitLab

1. Настройка Quality Gates к Merge Request'ам в GitLab

Применение Marlin Quality Gates к commit'ам или merge request'ам в GitLab

> **ПРИМЕЧАНИЕ**
>
> Необходимые права: Разработчик или Сотрудник эксплуатации
>
> Предварительные действия: Создано приложение, Создан сервис приложения, Существует доступ в репозиторий кода для запуска сборки, Настроены Quality Gates.

Описываемый в данной инструкции функционал может быть применён для запуска сборки, unit-тестирования и проверки статическими анализаторами на стороне Marlin для каждого commit'а или merge request'а в репозитории вашего сервиса.

Журнал работы процедур сборки и тестирования отображается непосредственно в интерфейсе GitLab.

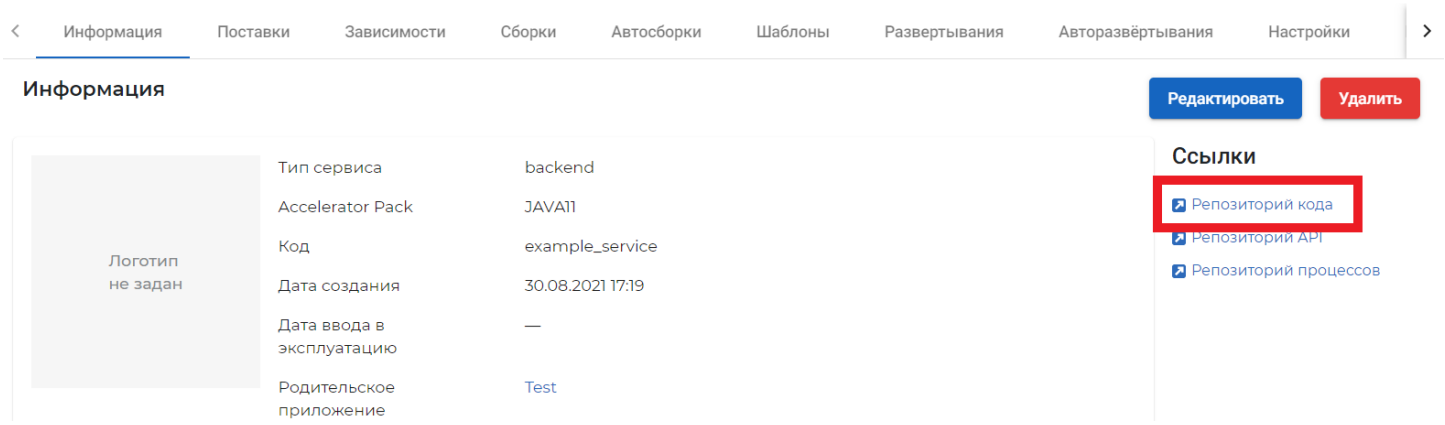
Результаты проверки могут быть применены для:

- обнаружения дефектов и иных проблем на ранней стадии;
- реализации ограничений на слияние веток.

Бизнес логика описанного функционала доступна по ссылке: Quality Gates в Marlin

Настройка

1. Перейти в "Репозиторий кода" на вкладке "Информация" сервиса приложения



2. Создать файл .gitlab-ci.yml в корне репозитория

   Пример конфигурации для запуска проверок на стороне Marlin при каждом commit't или merge request'e:

```
stages:
  - test

marlin-test:
  stage: test
  image: repo.vsk.ru:5013/marlin-platform/marlinctl
  script:
    - marlinctl -C "$MARLINCTL" create build --service=$SERVICE_CODE --
title=GitLabTriggeredBuild --dry-run --commit=$CI_COMMIT_SHA --check units --
check quality --follow
```

   , где

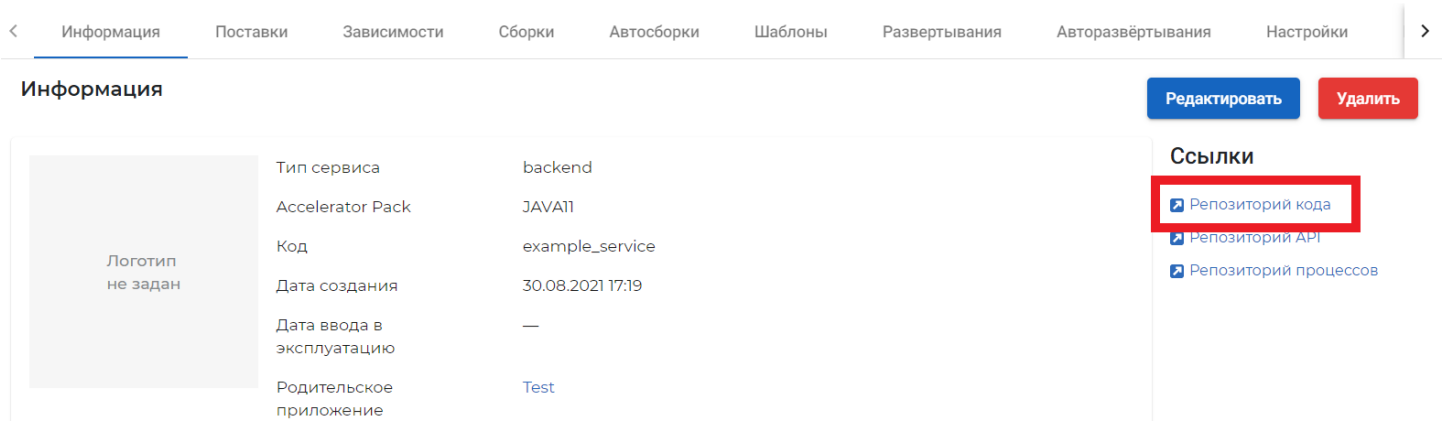
   - `$MARLINCTL` — путь до файла с параметрами подключения к Marlin (должен храниться в разделе Settings → CI/CD → Variables);
   - `$SERVICE_CODE` — код сервиса;
   - `--dry-run` — флаг, сообщающий Marlin, что результаты сборки не должны быть сохранены в хранилище артефактов;
   - `--check units --check quality` — проверки, которые необходимо запускать;
   - `--follow` — флаг, сообщающий, что необходимо дождаться окончания процедуры проверки и вывести журнал на экран.

   Дополнительную ифнморацию о применении утилиты `marlinctl` можно получить в этом разделе.

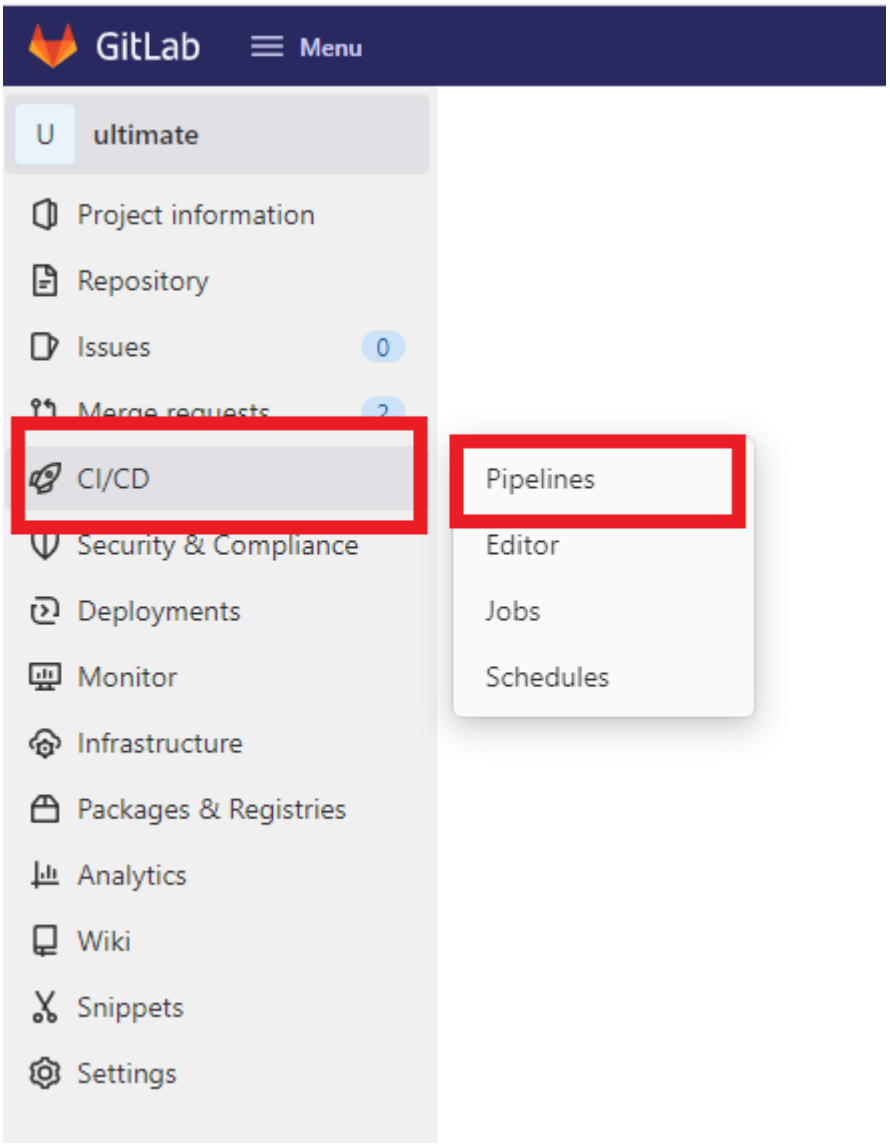
Более подробную информацию о GitLab CI можно получить из официального руководства

Просмотр результатов

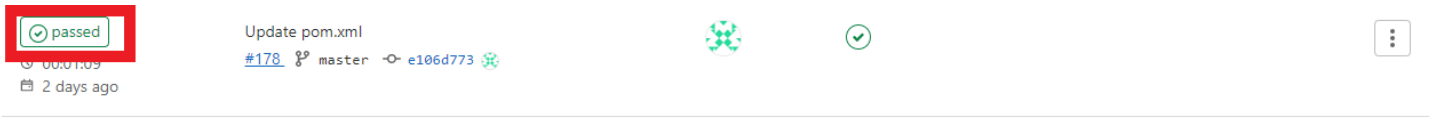
1. Перейти в "Репозиторий кода" на вкладке "Информация" сервиса приложения



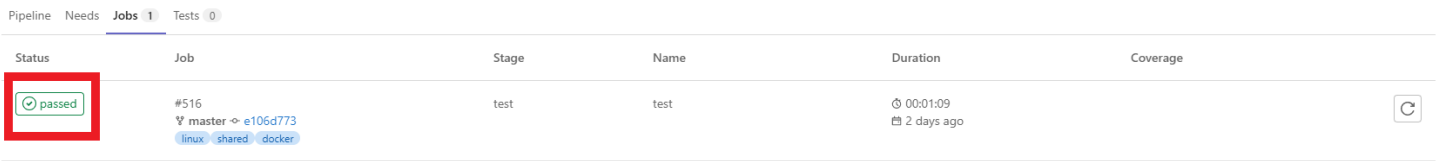
2. Перейти на вкладку "CI/CD" → Pipelines левого меню GitLab



3. Нажать на статус необходимого pipeline`а



4. Нажать на job`у во вкладке pipeline



СМОТРИТЕ ТАКЖЕ

Глоссарий "Marlin"

3.26 Интеграция с Платформой автоматизации тестирования

Для реализации процессов тестирования, Портал Управления (ПУ) направляет информацию о готовности сервиса к тестированию, среде, на которой он развёрнут и иную информацию, необходимую для проведения тестирования.

Платформа автоматизации тестирования, ориентируясь на полученную информацию, запускает тестирование, уведомляет команду о начале тестирования, предоставляет для портала обратную связь о ходе тестирования.

ПУ предоставляет в своём интерфейсе информацию о текущем статусе тестирования, проценте выполнения.

Перечень типов тестирования

Название	Описание	Среда	Тип сервиса
end-to-end	End-to-end	stage	*
functional	Функциональное	test	*
performance	Нагрузочное	stage	backend, bff
regression	Регрессионное	stage, prod	*
smoke	Smoke	test, stage, prod	*

Сообщение Marlin о готовности сервиса к тестированию

Поле	Тип	Пример	Описани
marlinLaunchId	uuid	"9ba13e09-b116-4f0d-b0e7-c123cb00c2e0"	Идентификато запуска
handler	string	"SPRUT"	Система тестирования
deployment.id	uuid	"9ba13e09-b116-4f0d-b0e7-c123cb00c2e0"	Идентификато разворачивани
project.code	string	"servicehub"	Уникальный ко идентифициру проект
service.code	string	"infogateway_adapter"	Уникальный ко идентифициру сервис
release.version	string	"0.0.4.0"	Версия сервис
template.addresses[].title	string	"gRPC"	Наименование адреса
template.addresses[].url	string	"https://dev.demo-backend5.apps.prdmz.example.org"	Ссылка на развернутый с
environment.code	enum of string	"test" или "stage"	Код среды, на которой разве сервис


```
{
  "marlinLaunchId": "9ba13e09-b116-4f0d-b0e7-c123cb00c2e0",
  "handler": "SPRUT",
  "deployment": {
    "id": "9ba13e09-b116-4f0d-b0e7-c123cb00c2e0"
  },
  "project": {
```

```
    "code": "servicehub"
  },
  "service": {
    "code": "infogateway_adapter"
  },
  "release": {
    "version": "0.0.4.0"
  },
  "template": {
    "addresses": [
      {
        "title": "gRPC",
        "url": "https://dev.demo-backend5.apps.prdmz.example.org"
      }
    ]
  },
  "environment": {
    "code": "stage"
  }
}
```

Сообщение Платформы автоматизации тестирования в Marlin о статусе тестирования

Поле	Тип	Пример	Описание
marlinLaunchId	uuid	"9ba13e09-b116-4f0d-b0e7-c123cb00c2e0"	Идентификатор запуска в Marlin
message	json		Сообщение для пользователя
testingType	string	"smoke"/"functional"/"regression"/...	Тип тестирования
executorLaunchId	long	12345	Идентификатор запуска в системе тестирования

Поле	Тип	Пример	Описание
status	enum of string	"in-progress", "finished", "error"	Успешность процесса тестирования
success	int	20	Количество успешных тесто (Необязательны параметр при статусе "error")
failed	int	10	Количество провалившихся тестов (Необязательны параметр при статусе "error")
total	int	50	Общее количество тестов (Необязательны параметр при статусе "error")
url[]	string	["https://allure.example.org/suite/12345"]	Ссылки для перехода к просмотру результатов тестирования (Необязательны параметр при статусе "error")


```
{
  "marlinLaunchId": "9ba13e09-b116-4f0d-b0e7-c123cb00c2e0",
```

```
  "status": "finished",
  "testingType": "smoke",
  "executorLaunchId": 12345,
  "message": "{\"timestamp\":1687854831149,\"status\":404,\"message\":\"job run with
bc910926-37dc-4837-ac9f-e9763c2f9f06 not found for job 79\",\"errors\":[]}",
  "stacktrace": "Exception of type
'Sprut.Integration.Application.Common.Exceptions.HttpExceptions.InvalidHttpStatusCodeE
was thrown.\r\n   at
Sprut.Integration.Application.Extensions.HttpClientEx.PostAsJsonAsync[TResponce](HttpC
httpClient, String requestUrl, Object requestValue, CancellationToken cancellationTok
/build/Sprut.Integration.Application/Extensions/HttpClientEx.cs:line 35\n   at
Sprut.Integration.Allure.ApiManagers.AllureUploadApiManager.StopJob(JobStopDto dto,
CancellationToken cancellationToken) in
/build/Sprut.Integration.Allure/ApiManagers/AllureUploadApiManager.cs:line 85\n   at
Sprut.Integration.Kafka.Requests.Launches.LaunchRequestHandler.Handle(LaunchRequest re
CancellationToken cancellationToken) in
/build/Sprut.Integration.Kafka/Requests/Launches/LaunchRequest.cs:line 89\n   at
Sprut.Integration.Kafka.Behaviors.KafkaExceptionHandlerBehavior`2.Handle(TRequest requ
RequestHandlerDelegate`1 next, CancellationToken cancellationToken) in
/build/Sprut.Integration.Kafka/Behaviors/KafkaExceptionHandlerBehavior.cs:line 35",
  "success": 20,
  "failed": 10,
  "total": 30,
  "url": "https://allure.example.org/suite/12345"
}
```

3.24 Реализация требований

1. Реализация требований к структуре проекта API
2. Реализация требований к сборке
3. Реализация требований к взаимодействию с PaaS OpenShift
4. Реализация требований к взаимодействию с PaaS Keycloak
5. Реализация требований к взаимодействию с PaaS PostgreSQL
6. Реализация требований к взаимодействию с PaaS MinIO
7. Реализация требований к взаимодействию с PaaS Kubernetes
8. Реализация требований к логированию
9. Реализация требований к проверке SonarQube
10. Реализация подключений юнит-тестов

Реализация требований к структуре проекта API

Проектирование сервиса следует начинать с того API, который определен как публичный, исходя из общесистемных требований к разрабатываемому сервису. Сам API определяется, как контракт взаимодействия по одному из следующих протоколов и соглашений:

- gRPC API, IDL: protobuffer;
- REST API, IDL: Open API Specification 3.0 (ex. SWAGGER);
- Kafka messaging, IDL: Avro - в первой очереди создания Платформы реализация протокола не предусмотрена.

Приоритетным протоколом является gRPC.

Для опубликованного API сервиса средствами ПУ создается отдельный проект в Git-репозитории в рамках группы проектов API и подгруппы проектов для приложения.

Для реализации требований необходимо создать структуру проекта API сервиса.

Проект API сервиса состоит из:

- набора папок - содержащих файлы на соответствующем IDL;
- документа в формате MD с описанием структуры и функций API;
- вложенных папок с файлами в формате json - содержащих примеры запросов для методов сервиса и шаблоны для валидации полученных ответов.

На основании этих файлов проводится проверка соответствия планируемой и фактической реализации API.

Посмотреть структуру API можно по следующим примерам:

Пример работы gRPC Server - предназначен для межсервисной авторизации с "Пример работы gRPC Client" по протоколу gRPC (в исходном коде **gRPC Server** лежит статичная страница с отображением курсов валют, которая передается по протоколу gRPC в приложение **gRPC Client** при подписке).

1. Необходимо подписаться на платформенный сервис Keycloak.
2. Необходимо подписаться на платформенный сервис OpenShift.
3. В проекте репозитория кода API необходимо добавить файл .IDL для проверки план-факт.
4. Обязательно заполнение в шаблоне развертывания адреса для GRPC.

Пример работы OpenAPI Server - предназначен для межсервисной авторизации с "Пример работы OpenAPI Client" по протоколу REST (в исходном коде **OpenAPI Server** лежит статичная страница с отображением курсов валют, которая передается по протоколу REST в приложение **OpenAPI Client** при подписке).

1. Необходимо подписаться на платформенный сервис Keycloak.
2. Необходимо подписаться на платформенный сервис OpenShift
3. В проекте репозитория кода API необходимо добавить файл .IDL для проверки план-факт.
4. Обязательно заполнение в шаблоне развертывания адреса для OpenAPI.

Реализация требований к сборке

Сборка - сущность ПУ, которая объединяет версию исходного кода сервиса, процесс сборки с редактированием его шагов и результаты сборки, включая артефакты и протоколы.

Для создания сборки необходимо заранее положить исходный код по ссылке "Репозиторий кода" в GitLab.

Пример структуры репозитория исходного кода:

- chart - опционально, папка, предназначена для хранения helm-чарта разворачивания сервиса в ситуации, когда нужно использовать собственную реализацию вместо встроенной
- keycloak - папка, содержит сведения о проливке пользовательской ролевой модели, необходима при использовании PaaS Keycloak. Подробнее см. Использование PaaS Keycloak
- отдельные файлы в корневом каталоге:
  - .gitignore - файл содержит список правил игнорирования для git. Подробнее
  - .dockerignore - файл содержит список правил игнорирования, позволяющий ускорить процедуру сборки. Подробнее
  - Dockerfile - файл, описывающий процедуру сборки docker image из исходных кодов и артефактов сборки приложения. Подробнее
  - README.md - файл содержит описание проекта в формате markdown

Инструкция с подробным описанием по работе со сборкой - см. Создание сборки.

После создания сборки в ПУ, автоматически запускается процесс сборки. При успешной сборке маркер Build загорится зеленым, в обратном случае красным. Протокол сборки можно просмотреть, нажав соответствующую кнопку рядом с маркером Build.

Типовые ошибки при работе со сборкой:

1. Сборка в статусе "Ошибка" - ознакомьтесь с протоколом сборки.

Реализация требований к взаимодействию с PaaS Kubernetes

Ресурсы для прикладного сервиса выделяются и настраиваются Платформой при создании подписки на сервис PaaS Kubernetes и его развертывании. Выделение ресурсов на средах необходимо произвести самостоятельно - см. Подписка на платформенный сервис PaaS.

Конфигурирование сервисов производится путем передачи переменных окружения. Среда исполнения Kubernetes получает список переменных от ПУ.

Отредактировать helm-chart следующим образом:

1. добавить название передаваемой переменной в файл values.yaml.
2. добавить мэпинг передаваемой переменной на переменные окружения конкретного контейнера в рамках deployment.yaml.

Реализация требований к логированию

Логирование должно осуществляться только в stdout/stderr, для обеспечения работы стандартного механизма сбора логов Kubernetes, OpenShift.

Для отслеживания цепочек вызовов в контексте одной операции, необходимо логировать вызовы сервисов с добавлением к каждой записи идентификатора корреляции.

Структура и уровень логов должны соответствовать этим соглашениям.

Разрабатываемые сервисы должны поддерживать механизм динамического изменения уровня логирования, путем предоставления стандартизированного API, который может быть вызван средствами Портала управления.

Пример работы Базовое логирование:

1. Необходимо подписаться на платформенный сервис Keycloak;
2. Необходимо подписаться на платформенный сервис OpenShift;
3. В шаблоне развертывания ввести переменные окружения (переменные окружения указываются в файле "values.yml" и в файле "deployment.yaml" - нижеуказанные ключи уже внесены в исходный код):
   - deployment.logIndex
   - deployment.logServiceCode
   - deployment.logServiceVerison
   - обязательно заполнение в шаблоне развертывания адреса
4. При успешном развертывании можем в ELK посмотреть логи;
5. В главном меню ELK надо открыть вкладку "Discover", в поисковой строке написать: kubernetes.namespace_name: (указать код сервиса);
6. Интерфейс приложения аналогичен с приложением Keycloak.

Инструкция по установке fluenttd

Сбор логов с docker-контейнеров ПУ осуществляется клиентом td-agent-bit (fluent-bit) на стороне портала и обрабатывается коллектором td-agent (fluentd) на мастер-ноде elasticsearch

Настройки на стороне портала:

/etc/yum.repos.d/td-agent-bit.repo:

```
[td-agent-bit]
name = TD Agent Bit
baseurl = https://packages.fluentbit.io/centos/7/$basearch/
gpgcheck=1
gpgkey=https://packages.fluentbit.io/fluentbit.key
enabled=1
```

```
$ yum install td-agent-bit
```

/etc/td-agent-bit/td-agent-bit.conf:

```
[SERVICE]

# Flush
# =====
# Set an interval of seconds before to flush records to a destination
Flush 10

# Daemon
# ======
# Instruct Fluent Bit to run in foreground or background mode.
Daemon Off

# Log_Level
# =========
# Set the verbosity level of the service, values can be:
#
# - error
# - warning
# - info
# - debug
# - trace
#
# By default 'info' is set, that means it includes 'error' and 'warning'.
Log_Level info

# Parsers_File
# ============
```

```
# Specify an optional 'Parsers' configuration file
Parsers_File parsers.conf
Plugins_File plugins.conf

[INPUT]
Name tail
Path /var/lib/docker/containers/*/*.log
DB /etc/td-agent-bit/docker-logs.db
DB.locking true
Parser docker
Refresh_Interval 30
Ignore_Older 6h
Docker_Mode On
Tag source.docker.<container_id>
Tag_Regex (.*\/(?<container_id>.*)-json\.log)

[FILTER]
name record_modifier
match *
record hostname ${HOSTNAME}

[FILTER]
Name lua
Match source.docker.*
script /etc/td-agent-bit/bin/docker-metadata.lua
call encrich_with_docker_metadata

[FILTER]
Name grep
Match source.docker.*
Regex docker.container_name dev_backend_*|dev_sidekiq_*

[OUTPUT]
Name forward
Match source.docker.*
Host marlin-esm-d1
Port 24224
```

/etc/td-agent-bit/bin/docker-metadata.lua:

```
DOCKER_VAR_DIR = '/var/lib/docker/containers/'
DOCKER_CONTAINER_CONFIG_FILE = '/config.v2.json'
DOCKER_CONTAINER_METADATA = {
['docker.container_name'] = '\"Name\":\"/?(.-)\"',
['docker.container_image'] = '\"Image\":\"/?(.-)\"',
```

```
['docker.container_started'] = '\"StartedAt\":\"/?(.-)\"'
}

cache = {}

-- Gets metadata from config.v2.json file for container
function get_container_metadata_from_disk(container_id)
local docker_config_file = DOCKER_VAR_DIR .. container_id ..
DOCKER_CONTAINER_CONFIG_FILE
fl = io.open(docker_config_file, 'r')

if fl == nil then
return nil
end

-- Parse json file and create record for cache
local data = {}
for line in fl:lines() do
for key, regex in pairs(DOCKER_CONTAINER_METADATA) do
local match = line:match(regex)
if match then
data[key] = match
end
end
end
fl:close()

if next(data) == nil then
return nil
else
return data
end
end

function encrich_with_docker_metadata(tag, timestamp, record)
-- Get container id from tag
container_id = tag:match'.*%.(.*)'
if not container_id then
return 0, 0, 0
end

-- Add container_id to record
new_record = record
new_record['docker.container_id'] = container_id

-- Check if we have fresh cache record for container
local cached_data = cache[container_id]
```

```
if cached_data == nil then
cached_data = get_container_metadata_from_disk(container_id)
end

-- Metadata found in cache or got from disk, enrich record
if cached_data then
for key, regex in pairs(DOCKER_CONTAINER_METADATA) do
new_record[key] = cached_data[key]
end
end

return 1, timestamp, new_record
end
```

```
$ sudo service td-agent-bit start
$ systemctl enable td-agent-bit.service
```

Настройки на мастер-ноде elasticsearch:

```
$ curl -L https://toolbelt.treasuredata.com/sh/install-redhat-td-agent4.sh | sh
```

/etc/td-agent/td-agent.conf:

```
<source>
@type forward
port 24224
bind 0.0.0.0
</source>

<match source.docker.*>
@type elasticsearch
host marlin-esm-d1
port 9200
ca_file /etc/td-agent/certificates/ca.crt # CA для elasticsearch
scheme https
ssl_verify false
user elastic
password ldap0102
logstash_format true
</match>
```

/etc/td-agent/certificates/ca.crt - CA для elasticsearch

```
$ sudo service td-agent start
$ systemctl enable td-agent.service
```

Реализация требований к обработке ошибок

Для обеспечения прозрачной обработки ошибок взаимодействия используются механизмы Istio, реализующие балансировку и маршрутизацию трафика, а также паттерны Retry и Circuit Breaker.

Для обеспечения работы Istio каждый сервис должен обеспечить минимизировать время сообщения о произошедшей ошибке: вернуть код ошибки, упасть с разрывом связи и т.п.

Требования к логированию

Формат JSON

Все логи должны быть в JSON.

Локально запущенный сервис может продолжать писать логи в консоль в стандартном виде, удобном разработчику.

Настройки логирования должны быть настроены в одном месте для всего сервиса. Например для Java - это конфигурация Logback.

Стандартные потоки вывода

Необходимо использовать стандартные потоки STDERR и STDOUT.

Вывод логов в файлы запрещен.

Уровни логирования

Следует грамотно использовать уровни логирования в соответствии с принятыми стандартами в вашем технологическом стеке.

В каждом языке свои уровни. Например, в Java нет Critical, который есть в .NET и Python, а в PHP есть ещё alert и emergency.

Следовательно, один и тот же error в Java, .NET, Python и PHP - семантически разные и имеют различную критичность реакции на них.

Логи сервиса

Сервис должен возвращать объект со следующими полями:

Поле	Тип	Формат	Примеры	Обязательность
level	string	Фиксированный набор допустимых значений, не зависящих от языка	Info	Обязательно
message	string	Произвольная человеко-читаемая строка	"Сервис запущен на порту 8080"	Обязательно
traceId	string	Произвольная уникальная строка	"549f55dfae1b61c4"	Обязательно

Поле	Тип	Формат	Примеры	Обязательность
parentSpanId	string	Произвольная уникальная строка	"07816f00988f2d59"	Обязательно

Поле	Тип	Формат	Примеры	Обязательность

Поле	Тип	Формат	Примеры	Обязательность
spanId	string	Произвольная уникальная	"07816f00988f2d59"	Обязательно

Поле	Тип	Формат	Примеры	Обязательность
		строка		
module	string	Латинские буквы в нижнем регистре и цифры, тире для разделения слов в одном названии, точка для разделения уровней вложенности	"my-module.my-submodule.my-process" "my-module.my-process" "my-module"	Желательно для монолитных приложений и крупных сервисов

Поле	Тип	Формат	Примеры	Обязательность
markers	string	Латинские буквы в нижнем регистре и цифры, тире для разделения слов	["audit", "sensitive", "request", "response", my-marker]	Опционально
timestamp	string	ISO 8601	2022-10-07T08:19:28.325+03:00	Обязательно
businessData	object	json	"userId": "some id" "userLogin" : "some	Опционально

Поле	Тип	Формат	Примеры	Обязательность
			login"	
payload	object	json, поля в произвольном формате	{"foo":{"bar": "baz"}}	Опционально

Поле	Тип	Формат	Примеры	Обязательность

Пример

```
{
  "level": "INFO",
  "message": "Transaction confirmed",
  "traceId": "549f55dfae1b61c4",
  "parentSpanId": "0ad1348f14031692",
  "spanId": "07816f00988f2d59",
  "module": "gate.psb",
  "markers": ["audit"],
  "timestamp": "2022-10-07T08:19:28.325+03:00",
  "businessData": {
    "userId": "some id",
    "userLogin" : "some login"
  },
  "payload": {
    "foo": {
      "bar": {}
    }
  }
}
```

В примере выше, gate.psb говорит нам, что в сервисе ЕПШ сообщение в логе именно от модуля интеграции с ПСБ и мы можем легко отфильтровать его от других сообщений этого сервиса.

Ограничения

- При большом количестве полей в JSON и использовании динамического (по умолчанию) маппинга индекса в ElasticSearch, возможны ошибки с превышением лимита на количество полей индекса (Limit of total fields [1000] in index has been exceeded”) для некоторых записей. Подробнее и варинты решения: https://discuss.elastic.co/t/approaches-to-deal-with-limit-of-total-fields-1000-in-index-has-been-exceeded/241039

- Допустимые суточные размеры логов:

Среда	Максимальное значение
dev	3gb
test	5gb
stage	10gb
prod	20gb

  При превышении установленных значений, доступ к записи логов сервису автоматически блокируется до полуночи по московскому времени.

  Логи должны храниться не менее месяца на продуктовом конутре.

Логи платформы

К логу сервиса будут добавлены параметры kubernetes, а также следующие данные:

Поле	Тип	Формат	Примеры	Обязательность	
application	string	Латинские буквы, цифры, подчёркивание	my_application_name	Обязательно	К п

Поле	Тип	Формат	Примеры	Обязательность	
service	string	Латинские буквы, цифры, подчёркивание	my_service_name	Обязательно	К
version	string	Цифры и точки	3.15.42	Обязательно	В с
time	string	ISO 8601	2022-10-07T08:19:28.325+03:00	Обязательно	В с л с п м

ELOG

Все логи приложений собираются средой исполнения и отправляются в коллектор логов.

В случае Marlin, это CRI-O и FluentD, в иных случаях, это может быть Fluent Bit или LogStash (до поры до времени).

Затем все логи отправляются в ElasticSearch, который создаёт по ним индексы и позволяет получить к ним доступ из интерфейса Kibana.

Наличие коллекторов позволяет привести логи от различных приложений к каноническому виду:

Пример

```
{
  "platform": {
    "application": "my_application_name",
    "service": "my_service_name",
    // ...
  },
```

```
  "service": {
    "level": "DEBUG",
    "message": "Some message",
    // ...
  },
  "kubernetes": {
    "namespace_name": "stage-kaskofactory",
    "container_name": "ae12a8be-ac71-42",
    // ...
  },
  "applog": {
    // ...
  }
}
```

Важно отметить, что содержимое поля service всегда возвращает сам сервис, но если сервис является "коробочным продуктом" и не может писать логи в формате из Таблицы 1, коллектор должен привести их к стандартному виду индивидуально для данного сервиса. Аналогично с полем platform, но сервисы в Marlin сами его не заполняют. Поле kubernetes отсутствует для сервисов вне Marlin. Также для обратной совместимости во всеми текущими сервисами оставлено старое поле applog, в которое сейчас попадают логи сервисов.

Ошибки работы сервиса

Логирование не имеет никакого отношения к возврату и обработке ошибок API! И наоборот :)

Важно помнить:

- возврат ошибок из вашего API нужен только стороннему сервису для принятия решения "что делать в случившейся ситуации".
- логирование ошибок в вашем сервисе нужно только вам для разбора причин и предотвращения возникновения в будущем

Если в вашей системе произошла ошибка, вы можете её залогировать со всеми подробностями, но это не значит что вы вернёте ошибку в ответе другой системе, а другая система, получив код ошибки от вас в API, может также что-то у себя залогировать, если захочет. И наоборот, если вы возвращаете некую ошибку во

внешнюю систему при вырове API, это не значит что вы у себя логируете её,а если и логируете, то не факт что как ошибку. а как info, trace, debug.

Пример: при вызове API вашего сервиса, выясняется, что у вызывающей системы недостаточно прав для выполнения конкретного запроса с конкретными параметрами. В этом случае вы вернёте в API ошибку (см. подробнее описания ошибок) и статус ответа 403. Но внутри вашего сервиса - это не ошибка, ведь ваш сервис как и положено проверил права, выяснил что их не хватает и штатно прекратил дальнейшее выполнение, т.е. в вашем сервисе никаких ошибок не возникло и логировать её как error или warn ни в коем случае нельзя!

Необходимо стремиться к тому. чтобы при штатной работе сервиса в логах не было warn, error и прочих критичных ошибок и предупреждений.

Другой пример: NullPointerException, деление на нуль, падение вашей БД и прочие внештатные ситуации обязательно должны логироваться!

Реализация требований к взаимодействию с PaaS MinIO

Выделение ресурсов для хранения неструктурированных данных осуществляется с помощью сервиса PaaS MinIO. MinIO - реализация протокола AS3, в соответствии с которым передача файлов осуществляется по SSL. MinIO - открытый сервер хранения объектов, который позволяет хранить неструктурированные данные. Предоставляется Платформой как платформенный сервис (PaaS).

Ресурсы для прикладного сервиса выделяются и настраиваются Платформой при создании подписки на сервис PaaS PostgreSQL и его развертывании. Выделение ресурсов на средах необходимо произвести самостоятельно - см. Подписка на платформенный сервис PaaS.

При добавлении развертывания для сервиса PaaS MinIO формируются следующие переменные окружения:

- USERACCESSKEY - наименование пользователя;
- USERSECRETKEY - секретный;
- POLICYNAME - наименование политики, связывающей USER и BUCKET;
- FORCEDELETE - параметр удаления;
- BUCKET - наименование хранилища [код прикладного сервиса] + [номер развертывания в среде].

При использовании PaaS, в чарт в values.yml и в необходимые шаблоны (т.е., например, шаблон для deployment) необходимо добавить переменные окружения в следующем формате:

- PAAS_CODE + "Url"
- PAAS_CODE + "Bucket"
- PAAS_CODE + "AccessKey"
- PAAS_CODE + "SecretKey"

Т.е. если **PAAS_CODE = miniodefault**, то необходимо добавить переменные **miniodefaultUrl, miniodefaultBucket, miniodefaultAccessKey,**

miniodefaultSecretKey, значения которых автоматически подставятся сгенерированными.

После успешного развертывания прикладного сервиса генерируются:

- bucket пользователя (хранилище данных, предназначенное для одного пользователя; состав наименования Bucket - [код сервиса] + [код подписанного PaaS] + [bucket]);
- политика использования корзины для пользователя (привязка пользователя к bucket, которые он использует).

Пользователю может быть доступно несколько bucket. По умолчанию для каждого bucket выделен определенный объем памяти, который не регулируется в рамках Платформы.

Использование файлового хранилища (MinIO) - чтение или запись объекта:

1. Необходимо подписаться на платформенный сервис MinIO;
2. Необходимо подписаться на платформенный сервис OpenShift;
3. Обязательно заполнение в шаблоне развертывания адресов;
4. После успешного развертывания перейдите по адресу развертывания - откроется страница, где будет написано "Нет файлов";
5. Далее необходимо перейти в интерфейс minio;
6. В главном меню по коду сервиса найти созданный bucket;
7. Загрузите документ;
8. Обновите страницу (открывается по адресу развертывания), загруженный документ отобразится в рабочем поле приложения.

Реализация требований к взаимодействию с PaaS OpenShift

OpenShift - среда исполнения прикладных сервисов, предоставляемся Платформой как платформенный сервис. Ресурсы для прикладного сервиса (проект) выделяются Платформой автоматически при создании подписки на сервис PaaS OpenShift.

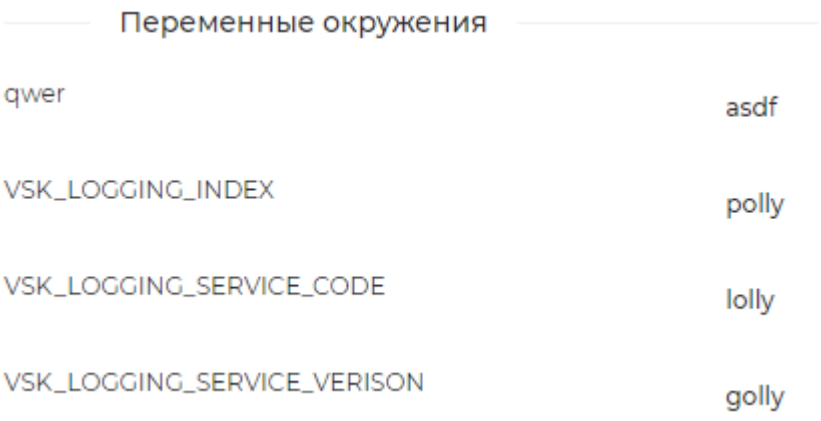
Предоставляется платформой, как платформенный сервис (PaaS).

Подписка прикладного сервиса на PaaS может быть **только одна** (PaaS "OpenShift LAN" или PaaS "OpenShift DMZ").

Ресурсы для прикладного сервиса выделяются и настраиваются Платформой при создании подписки на сервис PaaS OpenShift и его развертывании. Выделение ресурсов на средах необходимо произвести самостоятельно - см. Подписка на платформенный сервис PaaS.

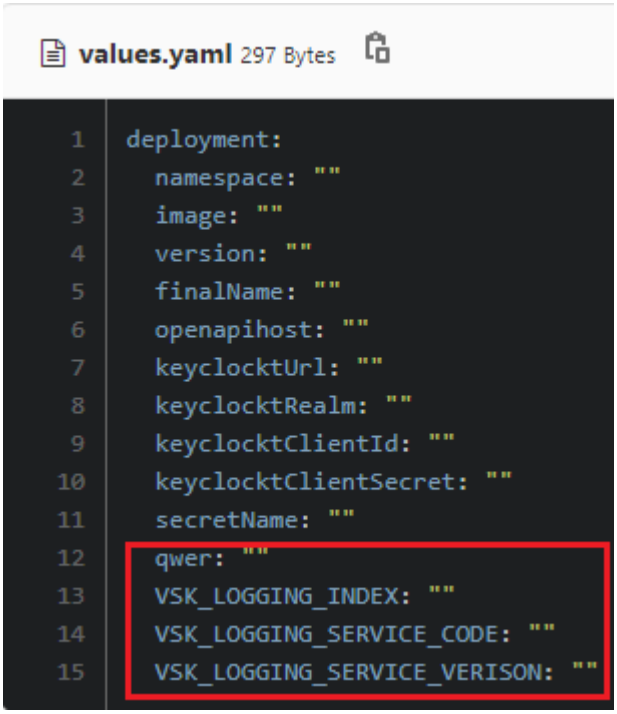
Переменные окружения должны быть единственным способом конфигурировать сервис, запускаемый на Платформе. Для этого:

1. они должны быть зарегистрированы в шаблоне развертывания прикладного сервиса, см. Создание шаблона развертывания (необходимо обеспечить соответствие ключей переменных окружения в шаблоне развертывания сервиса и записи в файле);



2. они должны быть объявлены в исходном коде - файл .yml следующим образом:

- в папке "chart", далее файл "values.yaml" должны быть добавлены ключи переменной окружения, соответствующие зарегистрированным в ПУ;



- в папке "chart", подпапка "templates", файл "deployment.yaml" в блок переменных окружения конкретного контейнера добавить параметр шаблона из предыдущего пункта.

deployment.yaml 1.58 KB

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.deployment.finalName }}
  namespace: {{ .Values.deployment.namespace }}
  labels:
    marlin.app: {{ .Values.deployment.finalName }}
spec:
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: {{ .Values.deployment.finalName }}
  replicas: 1
  template:
    metadata:
      labels:
        app: {{ .Values.deployment.finalName }}
    spec:
      dnsConfig:
        nameservers:
        - 8.8.8.8
      containers:
        - name: {{ .Values.deployment.finalName }}
          image: {{ .Values.deployment.image }}/{{ .Values.deployment.namespace }}
          imagePullPolicy: Always
          ports:
            - name: http
              containerPort: 8080
          env:
            - name: KEYCLOAK_URI
              value: {{ .Values.deployment.keycloak_devUrl }}
            - name:  KEYCLOAK_REALM
              value: {{ .Values.deployment.keycloak_devRealm }}
            - name:  KEYCLOAK_CLIENT_ID
              value: {{ .Values.deployment.keycloak_devClientId }}
            - name:  KEYCLOAK_CLIENT_SECRET
              value: {{ .Values.deployment.keycloak_devClientSecret }}
            - name:  VSK_LOGGING_INDEX
              value: {{ .Values.deployment.logIndex }}
            - name:  VSK_LOGGING_SERVICE_CODE
              value: {{ .Values.deployment.logServiceCode }}
            - name:  VSK_LOGGING_SERVICE_VERISON
              value: {{ .Values.deployment.logServiceVerison }}
```

Реализация требований к взаимодействию с PaaS PostgreSQL

PostgreSQL - система управления реляционными базами данными, которая предоставляется Платформой, как платформенный сервис (PaaS).

> **ПРИМЕЧАНИЕ**
>
> Используемая версия драйвера должна быть не ниже 42.1.4 (в противном случае возможны проблемы с PgBouncer. https://github.com/pgjdbc/pgjdbc/issues/869)

Ресурсы для прикладного сервиса выделяются и настраиваются Платформой при создании подписки на сервис PaaS PostgreSQL и его развертывании. Выделение ресурсов на средах необходимо произвести самостоятельно - см. Подписка на платформенный сервис PaaS.

Статья с подробным описанием взаимодействия с PaaS PostgreSQL.

При использовании PaaS, в чарт в values.yml и в необходимые шаблоны (т.е., например, шаблон для deployment) необходимо добавить переменные окружения в следующем формате:

- PAAS_CODE + "Host"
- PAAS_CODE + "User"
- PAAS_CODE + "Password"

Т.е. если **PAAS_CODE = postgredefault**, то необходимо добавить переменные **postgredefaultHost, postgredefaultUser, postgredefaultPassword**.

Чтение или запись в БД:

1. Необходимо подписаться на платформенный сервис PostgreSQL;
2. Необходимо подписаться на платформенный сервис OpenShift;
3. Обязательно в параметрах в chart/values.yaml и chart/templates/deployment.yaml:
    - заменить postgredefault на код используемого PaaS;

- postgredefaultHost: "" postgredefaultUser: "" postgredefaultPassword: "".

4. Обязательно заполнение в шаблоне развертывания адресов;

5. После успешного развертывания перейдите по адресу развертывания - откроется страница, где отобразится таблица со значениями;

6. Далее необходимо войти в БД под администратором;

7. Выберите из списка Databases нужную БД, перейдите в таблицу и внесите все необходимые изменения;

8. Обновите страницу с таблицей, внесенные изменения отобразятся в рабочем поле приложения.

Требования к приложениям о репликации, healthcheck'ax, graceful shutdown

Liveness probe

Liveness - проба позволяет Кубернетусу делать Health-check'и и при непрохождении определенного в описании Пробы количества проверок Kubernetes перезагрузит контейнер в Поде.

Проверка:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: my-pod
  name: my-pod-http
spec:
  containers:
  - name: containername
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 2
```

В данном случае, если сервис не отвечает на Health-check, или отвечает не 200-ым или 400-ым кодом ответа, он провалит одну из 3 (по умолчанию) проверок.

- *initialDelaySeconds*: 3 - означает, что после запуска Пода и перед первой Проверкой пройдет 3 секунды. По умолчанию - 0 секунд;

- *periodSeconds*: 2 - означает, что проверки будут происходить каждый 2 секунды. По умолчанию - 10 секунд;
- *failureThreshold*: количество повторных Проверок перед рестартом. По умолчанию 3;
- *timeoutSeconds*: Количество секунд ожидания ответа на Проверке. По умолчанию 1 секунда;
- *successThreshold*: Минимальное количество последовательных проверок, чтобы проба считалась успешной после неудачной. По умолчанию 1.

Многие приложения, работающие в течение длительного времени, ломаются и могут быть восстановлены только перезапуском. Kubernetes предоставляет liveness пробы, чтобы обнаруживать и исправлять такие ситуации.

Если команда успешна, она возвращает 0, и kubelet считает контейнер живым и здоровым. Если команда возвращает ненулевое значение, kubelet убивает и перезапускает контейнер.

Когда контейнер запускается, он исполняет команду:

```
/bin/sh -c "touch /tmp/healthy; sleep 30; rm -f /tmp/healthy; sleep 600"
```

Для первых 30 секунд жизни контейнера существует файл /tmp/healthy. Поэтому в течение первых 30 секунд команда cat /tmp/healthy возвращает код успеха. После 30 секунд cat /tmp/healthy возвращает код ошибки.

Readiness probe

Readiness пробы запускаются на контейнере в течение всего жизненного цикла.

Kubernetes предоставляет readiness пробы для определения и нивелирования ситуаций, когда приложения временно не могут обслужить траффик. Pod с контейнерами сообщает, что они не готовы принимать траффик через Kubernetes Services.

Readiness и liveness пробы могут быть использованы одновременно на одном контейнере. Использование обеих проб может обеспечить отсутствие траффика в

контейнер, пока он не готов для этого, и контейнер будет перезапущен, если сломается.

Допустим у Вас деплоймент с 3 репликами Подов, на один из Подов попал крупный запрос и он ушел 20 секунд его отрабатывать, лучшим решением будет перенаправлять трафик на другие Поды, пока загруженный не отработает. Так работает readinessProbe, в случае ее непрохождение, Kubernetes Service не будет посылать запросы на Под.

```
readinessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

Типы Kubernetes Health Checks

- HTTP-запросы HTTP-запрос — это механизм, который позволяет вам создать пробу живучести. Вы можете открыть конечную точку HTTP, реализовав любой упрощенный HTTP-сервер в контейнере. Зонд выполняет HTTP-запрос GET к этой конечной точке по IP-адресу контейнера, чтобы проверить, работает ли служба. Если конечная точка возвращает код успеха, считает контейнер живым и работоспособным. Если нет, kubelet завершает работу и перезапускает этот контейнер.
- Команды Вы можете настроить командный зонд, чтобы kubelet выполнял команду cat /tmp/healthy в определенном контейнере. Если команда выполнена успешно, kubelet считает контейнер живым и работоспособным. Если нет, он закрывает контейнер и перезапускает его.
- TCP-соединения Зонд сокета TCP сообщает Kubernetes об открытии TCP-соединения на указанном порту в контейнере. В случае успеха Kubernetes считает контейнер работоспособным. Вы должны использовать зонды TCP для служб gRPC и FTP, которые уже включают инфраструктуру протокола TCP.

Graceful shutdown (Завершение работы узла)

Kubernetes пытается обнаружить отключение системы узла и завершает работу модулей, работающих на узле.

Kubernetes гарантирует, что модули следуют обычному процессу завершения работы модуля во время отключения узла.

Функция Graceful node shutdown зависит от systemd, поскольку она использует блокировку ингибитора systemd для задержки выключения узла на заданную продолжительность.

Мягкое завершение работы узла контролируется с помощью *GracefulNodeShutdown* шлюза функций , который включен по умолчанию в версии 1.21.

Обратите внимание, что по умолчанию оба параметра конфигурации, описанные ниже, *shutdownGracePeriod* установлены *shutdownGracePeriodCriticalPods* на ноль, поэтому функция корректного отключения узла не активируется. Чтобы активировать эту функцию, необходимо соответствующим образом настроить два параметра конфигурации kubelet и установить ненулевые значения.

Во время корректного завершения работы kubelet завершает работу pod’ов в два этапа:

1. Завершите обычные модули, работающие на узле;
2. Завершите критически важные модули, работающие на узле.

Функция плавного отключения узла настраивается с двумя *KubeletConfiguration* параметрами:

1. *shutdownGracePeriod*: Указывает общую продолжительность, на которую узел должен задержать завершение работы. Это общий льготный период для прекращения работы как обычных, так и [критических модулей] (https://kubernetes.io/docs/tasks/administer-cluster/guaranteed-scheduling-critical-addon-pods/#marking-pod-as-critical).
2. *shutdownGracePeriodCriticalPods*: Указывает продолжительность, используемую для завершения [критических модулей](https://kubernetes.io/docs/tasks/administer-cluster/guaranteed-scheduling-critical-addon-pods/#marking-pod-as-critical) во время отключения узла. Это значение должно быть меньше *shutdownGracePeriod*.

Например, если *shutdownGracePeriod=30s*, и *shutdownGracePeriodCriticalPods=10s*, kubelet задержит выключение узла на 30 секунд. Во время выключения первые 20 (30-10) секунд будут зарезервированы для корректного завершения обычных модулей, а последние 10 секунд будут зарезервированы для завершения критических модулей.

> **К СВЕДЕНИЮ**
>
> Когда модули были вытеснены во время корректного отключения узла, они помечаются как отключенные. Running *kubectl get pods* показывает статус выселенных модулей как *Terminated*. И *kubectl describe pod* указывает, что модуль был исключен из-за отключения узла:
>
> Reason: Terminated Message: Pod was terminated in response to imminent node shutdown.

StartupProbe

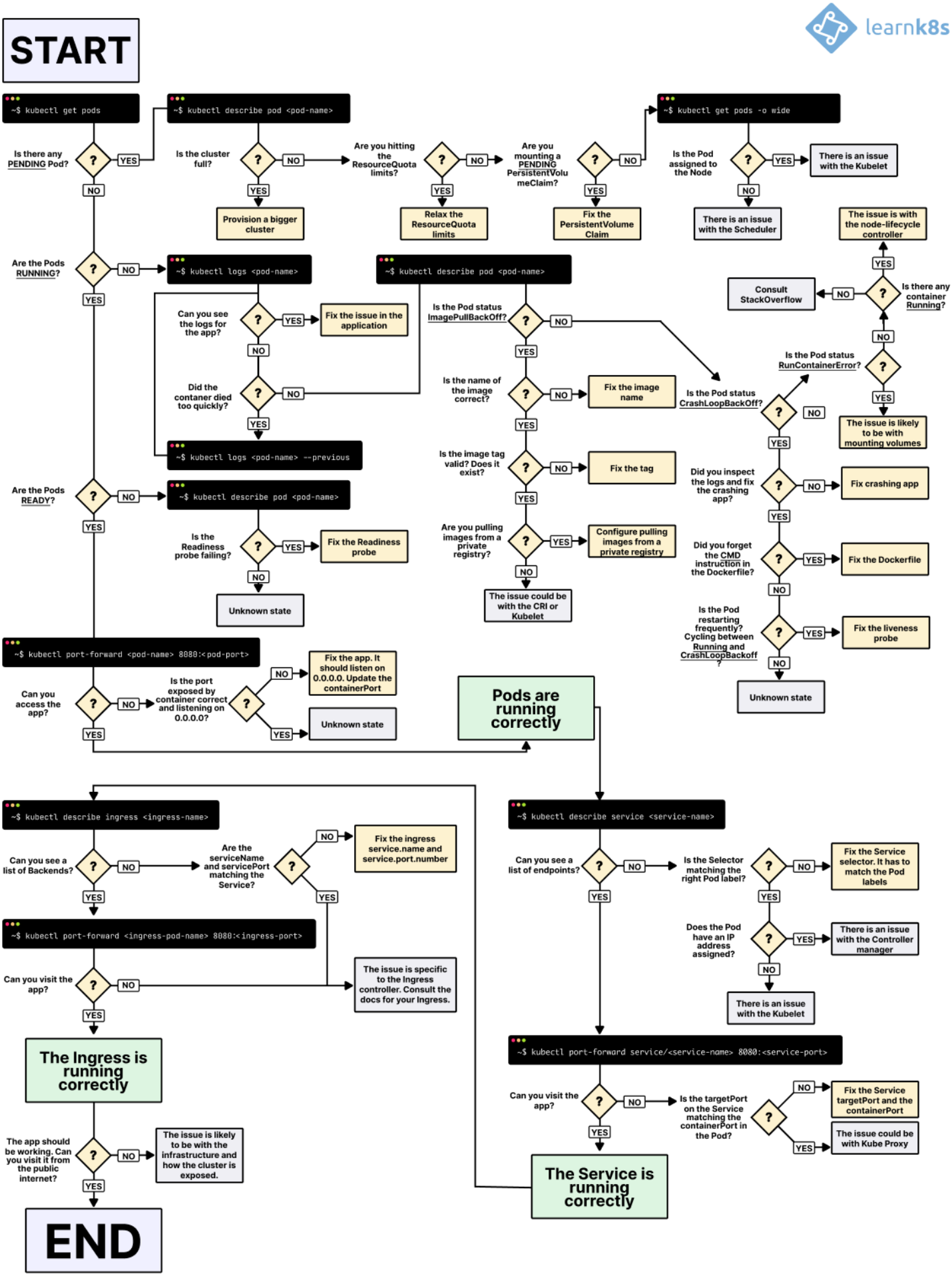
Мы периодически сталкиваемся с кейсами, когда приложение стартует довольно долго. Например, мы дали ему мало CPU, и оно запускается 5 минут, в результате чего не проходит все readiness/liveness проверки и попадает в вечный ребут (CrashLoop).

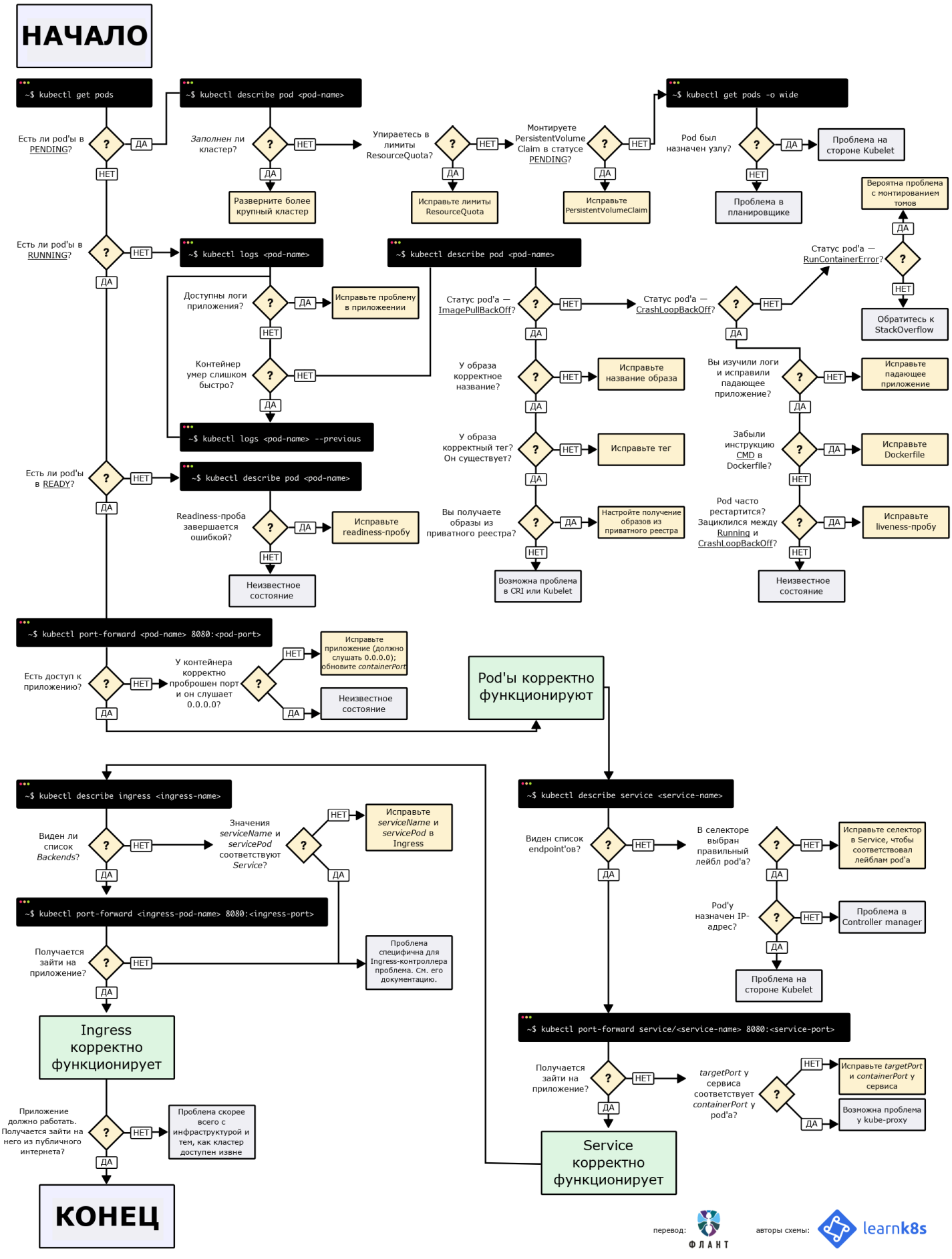
StartupProbe решает эту проблему, ведь если она настроена, liveness и readiness Пробы не будут отрабатывать, пока не завершится работа Стартап Пробы.

```
startupProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 30
  periodSeconds: 10
```

Запуск

learnk8s


START
~$ kubectl get pods
Is there any PENDING Pod?
YES
NO
~$ kubectl describe pod <pod-name>
Is the cluster full?
YES
Provision a bigger cluster
NO
Are you hitting the ResourceQuota limits?
YES
Relax the ResourceQuota limits
NO
Are you mounting a PENDING PersistentVolumeClaim?
YES
Fix the PersistentVolume Claim
NO
~$ kubectl get pods -o wide
Is the Pod assigned to the Node
YES
There is an issue with the Kubelet
NO
There is an issue with the Scheduler
Are the Pods RUNNING?
NO
~$ kubectl logs <pod-name>
Can you see the logs for the app?
YES
Fix the issue in the application
NO
Did the contaner died too quickly?
NO
YES
~$ kubectl logs <pod-name> --previous
~$ kubectl describe pod <pod-name>
Is the Pod status ImagePullBackOff?
NO
YES
Is the name of the image correct?
NO
Fix the image name
YES
Is the image tag valid? Does it exist?
NO
Fix the tag
YES
Are you pulling images from a private registry?
YES
Configure pulling images from a private registry
NO
The issue could be with the CRI or Kubelet
Is the Pod status CrashLoopBackOff?
NO
YES
Did you inspect the logs and fix the crashing app?
NO
Fix crashing app
YES
Did you forget the CMD instruction in the Dockerfile?
YES
Fix the Dockerfile
NO
Is the Pod restarting frequently? Cycling between Running and CrashLoopBackoff?
YES
Fix the liveness probe
NO
Unknown state
Is the Pod status RunContainerError?
NO
YES
The issue is likely to be with mounting volumes
Is there any container Running?
YES
The issue is with the node-lifecycle controller
NO
Consult StackOverflow
Are the Pods READY?
NO
~$ kubectl describe pod <pod-name>
Is the Readiness probe failing?
YES
Fix the Readiness probe
NO
Unknown state
~$ kubectl port-forward <pod-name> 8080:<pod-port>
Can you access the app?
NO
Is the port exposed by container correct and listening on 0.0.0.0?
NO
Fix the app. It should listen on 0.0.0.0. Update the containerPort
YES
Unknown state
YES
Pods are running correctly
~$ kubectl describe service <service-name>
Can you see a list of endpoints?
NO
Is the Selector matching the right Pod label?
NO
Fix the Service selector. It has to match the Pod labels
YES
Does the Pod have an IP address assigned?
YES
There is an issue with the Controller manager
NO
There is an issue with the Kubelet
YES
~$ kubectl port-forward service/<service-name> 8080:<service-port>
Can you visit the app?
NO
Is the targetPort on the Service matching the containerPort in the Pod?
NO
Fix the Service targetPort and the containerPort
YES
The issue could be with Kube Proxy
YES
The Service is running correctly
~$ kubectl describe ingress <ingress-name>
Can you see a list of Backends?
NO
Are the serviceName and servicePort matching the Service?
NO
Fix the ingress service.name and service.port.number
YES
~$ kubectl port-forward <ingress-pod-name> 8080:<ingress-port>
Can you visit the app?
NO
The issue is specific to the Ingress controller. Consult the docs for your Ingress.
YES
The Ingress is running correctly
The app should be working. Can you visit it from the public internet?
NO
The issue is likely to be with the infrastructure and how the cluster is exposed.
YES
END


НАЧАЛО
~$ kubectl get pods
Есть ли pod'ы в PENDING?
ДА
НЕТ
~$ kubectl describe pod <pod-name>
Заполнен ли кластер?
НЕТ
ДА
Разверните более крупный кластер
Упираетесь в лимиты ResourceQuota?
НЕТ
ДА
Исправьте лимиты ResourceQuota
Монтируете PersistentVolume Claim в статусе PENDING?
НЕТ
ДА
Исправьте PersistentVolumeClaim
~$ kubectl get pods -o wide
Pod был назначен узлу?
ДА
НЕТ
Проблема на стороне Kubelet
Проблема в планировщике
Вероятна проблема с монтированием томов
ДА
Есть ли pod'ы в RUNNING?
НЕТ
ДА
~$ kubectl logs <pod-name>
Доступны логи приложения?
ДА
НЕТ
Исправьте проблему в приложеении
Контейнер умер слишком быстро?
НЕТ
ДА
~$ kubectl logs <pod-name> --previous
~$ kubectl describe pod <pod-name>
Статус pod'а — ImagePullBackOff?
НЕТ
ДА
Статус pod'а — CrashLoopBackOff?
НЕТ
ДА
Статус pod'а — RunContainerError?
ДА
НЕТ
Обратитесь к StackOverflow
У образа корректное название?
НЕТ
ДА
Исправьте название образа
У образа корректный тег? Он существует?
НЕТ
ДА
Исправьте тег
Вы получаете образы из приватного реестра?
ДА
НЕТ
Настройте получение образов из приватного реестра
Возможна проблема в CRI или Kubelet
Вы изучили логи и исправили падающее приложение?
НЕТ
ДА
Исправьте падающее приложение
Забыли инструкцию CMD в Dockerfile?
ДА
НЕТ
Исправьте Dockerfile
Pod часто рестартится? Зациклился между Running и CrashLoopBackOff?
ДА
НЕТ
Исправьте liveness-пробу
Неизвестное состояние
Есть ли pod'ы в READY?
НЕТ
ДА
~$ kubectl describe pod <pod-name>
Readiness-проба завершается ошибкой?
ДА
НЕТ
Исправьте readiness-пробу
Неизвестное состояние
~$ kubectl port-forward <pod-name> 8080:<pod-port>
Есть доступ к приложению?
НЕТ
ДА
У контейнера корректно проброшен порт и он слушает 0.0.0.0?
НЕТ
ДА
Исправьте приложение (должно слушать 0.0.0.0); обновите containerPort
Неизвестное состояние
Pod'ы корректно функционируют
~$ kubectl describe ingress <ingress-name>
Виден ли список Backends?
НЕТ
ДА
Значения serviceName и servicePod соответствуют Service?
НЕТ
ДА
Исправьте serviceName и servicePod в Ingress
~$ kubectl port-forward <ingress-pod-name> 8080:<ingress-port>
Получается зайти на приложение?
НЕТ
ДА
Проблема специфична для Ingress-контроллера проблема. См. его документацию.
Ingress корректно функционирует
Приложение должно работать. Получается зайти на него из публичного интернета?
НЕТ
ДА
Проблема скорее всего с инфраструктурой и тем, как кластер доступен извне
КОНЕЦ
~$ kubectl describe service <service-name>
Виден список endpoint'ов?
НЕТ
ДА
В селекторе выбран правильный лейбл pod'а?
НЕТ
ДА
Исправьте селектор в Service, чтобы соответствовал лейблам pod'а
Pod'у назначен IP-адрес?
НЕТ
ДА
Проблема в Controller manager
Проблема на стороне Kubelet
~$ kubectl port-forward service/<service-name> 8080:<service-port>
Получается зайти на приложение?
НЕТ
ДА
targetPort у сервиса соответствует containerPort у pod'а?
НЕТ
ДА
Исправьте targetPort и containerPort у сервиса
Возможна проблема у kube-proxy
Service корректно функционирует
перевод:
ФЛАНТ
авторы схемы:
learnk8s

Реализация требований к проверке SonarQube

Проверка SonarQube осуществляется на выявление заданных параметров исходного кода на качество.

Особенности реализации:

- управление настройками параметров проверок осуществляется при сборке сервиса посредством ручного выбора проверок в соответствующем блоке - см. Создание сборки;
- обеспечивается с помощью стандартных политик.

В интерфейсе SonarQube должна быть реализована возможность добавления правил проверки (Quality profile) и признаков, по которым оценивается качество кода (Quality gate). В SonarQube должны быть предусмотрены стандартные наборы правил и признаки для проверки кода (с пометкой "default"), которые должны применятся по умолчанию в случае, если не было добавлено других. Должно быть предусмотрено создание наборов правил для любого языка (включая JavaScript и XML), перечень языков должен определяться автоматически.

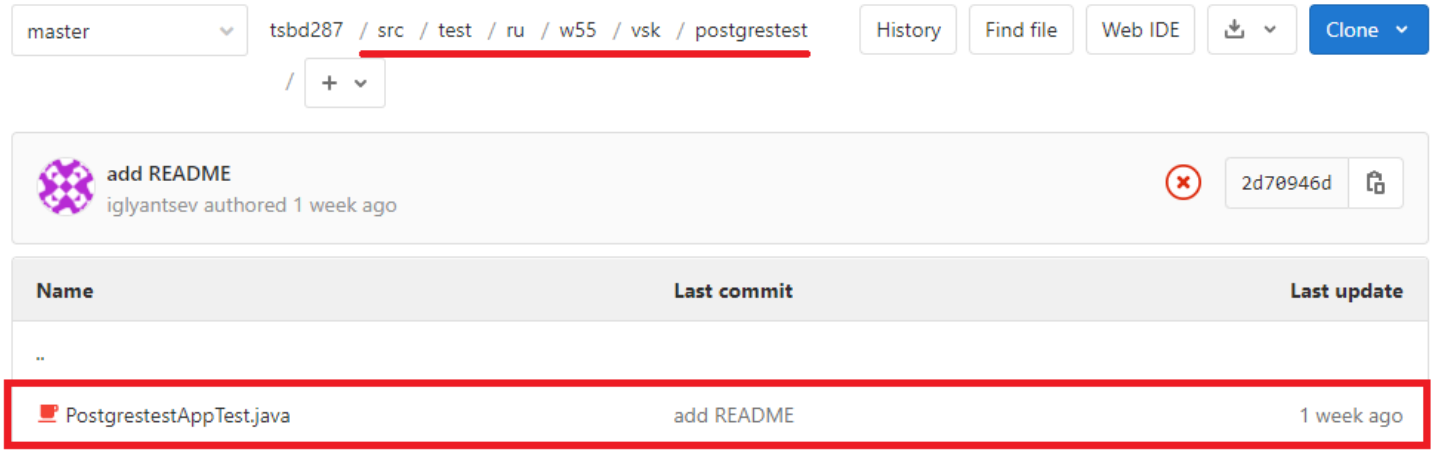
Успешное прохождение проверки необходимо для перевода на следующую среду.

Результат проверки качества своего проекта будет доступен по ссылке, предоставленной в конце анализа.

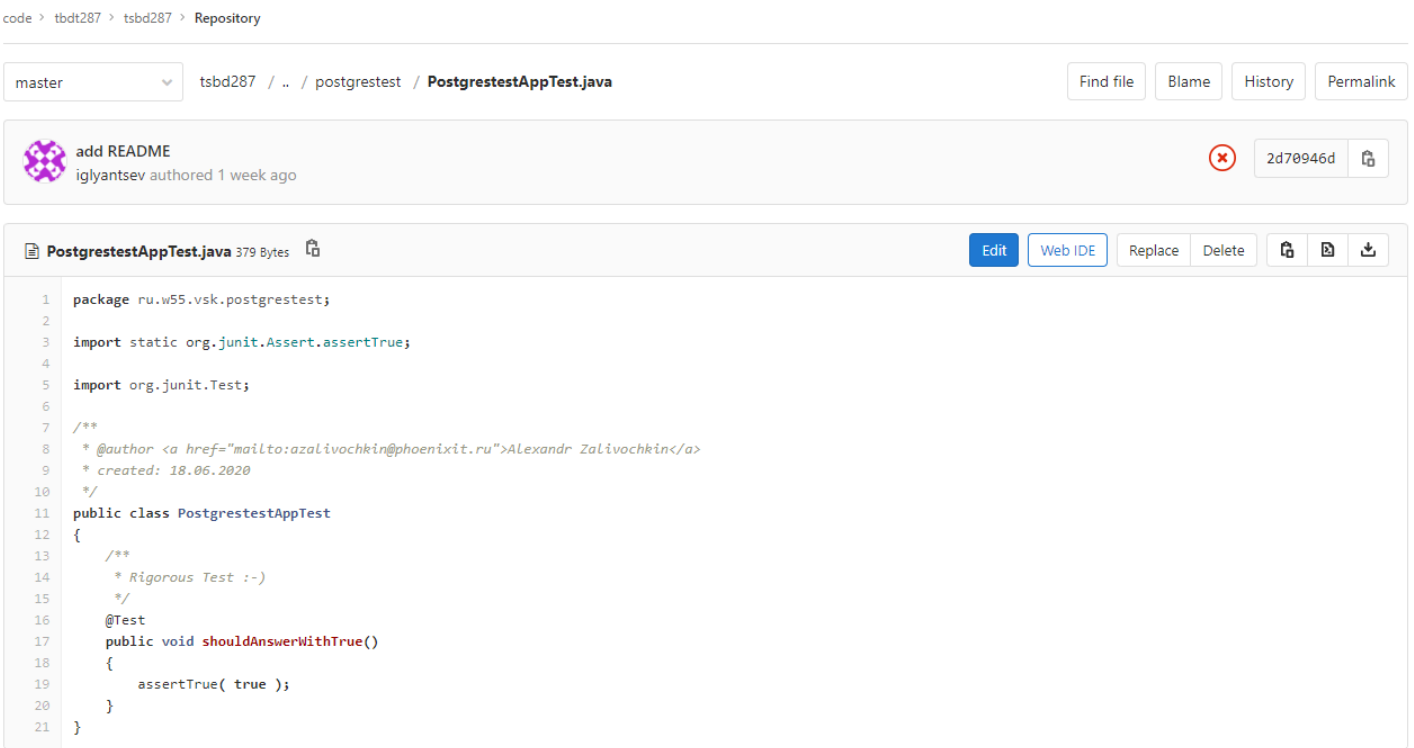
Реализация подключений юнит-тестов

Юнит-тесты должны запускаться во время запуска сборки.

1. Открыть файл "PostgrestestAppTest.java". В структуре исходного кода он располагается: code → src → test → java → ru → w55 → vsk → postgrestest.



2. Внести изменения: указать префикс "test" и аннотацию "@Test".



3. В файле pom.xml добавить модуль junit:

pom.xml

View file @ aea8474d

```
@@ -14,6 +14,7 @@
        <java.version>1.8</java.version>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
+       <junit-jupiter.version>5.2.0</junit-jupiter.version>
    </properties>
```

4. В файле pom.xml добавить зависимости:

```
-
+       <dependency>
+           <groupId>org.junit.jupiter</groupId>
+           <artifactId>junit-jupiter-engine</artifactId>
+           <version>${junit-jupiter.version}</version>
+           <scope>test</scope>
+       </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
@@ -44,7 +50,12 @@
            <version>4.13</version>
            <scope>test</scope>
        </dependency>
-
+       <dependency>
+           <groupId>org.junit.jupiter</groupId>
+           <artifactId>junit-jupiter-params</artifactId>
+           <version>${junit-jupiter.version}</version>
+           <scope>test</scope>
+       </dependency>
```

5. В файле pom.xml добавить плагин:

```
-               <version>2.20</version>
+               <version>2.22.0</version>
+           </plugin>
+           <plugin>
+               <artifactId>maven-failsafe-plugin</artifactId>
+               <version>2.22.0</version>
            </plugin>
        <plugin>
        <artifactId>maven-jar-plugin</artifactId>
```

6. Открыть в ПУ карточку требуемого сервиса, перейти на вкладку "Сборка" и создать новую сборку.

7. Успешность запуска юнит-тестов можно посмотреть в протоколе сборки:

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running ru.w55.vsk.miniotest.MiniotestAppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.014 s - in ru.w55.vsk.miniotest.MiniotestAppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

4. Требования к приложениям

Требования к приложениям, разрабатываемым и запускаемым на платформе Marlin.

Поддерживаемые стеки

Приложения должны быть реализованы на одном из поддерживаемых стеков разработки:

Стек	Unit-тесты	Контроль степени покрытия	SonarQube	Базовые образы	Прокси-репозитории
Maven, JDK v17 (Java, Kotlin, Groovy и т.д.)	да	да	да	да	да
.NET v6	да	да	да	да	да
Gradle	да	да	да	да	да
nodeJS v18			да	да	да
PHP			да	да	да
Python			да	да	да
Angular			да	да	да
GoLang			да	да	да
Иные (docker)			да		

Поддерживаемые PaaS

В своей работе приложения должны применять инструменты, предоставляемые платформой

- PostgreSQL v11
- MinIO (S3)
- Keycloak для реализации пользовательской аутентификации (OIDC)
- Keycloak для реализации межсервисной аутентификации
- Redis
- иные сервисы, реализованные внешними провайдерами услуг

Контейнеризация, сборка

Микросервисы собираются в docker-образы и эксплуатируются на кластерах kubernetes.

Исходные коды приложения должы содержать Dockerfile, описывающий способ контейнеризации сервиса.

Должно обеспечиваться четкое разделение на стадию сборки приложения и его запуск. Доустановка пакетов при запуске - не допускается.

Запуск процесса в контейнре осуществляется от non-root пользователя. Процедуры сборки образа должны это учитывать.

Образ приложения собирается один раз для применения на всех средах (testing/staging/production). Образы должны поддерживать возможность включения и выключения режима отладки/трассировки. Образы не должны содержать в себе предустановленной конфигурации, в т.ч. адресов смежных систем, паролей.

Образы должны быть настроены на установку пакетов только через платформенный Nexus.

Конфигурирование микросервисов

Конфигурирование микросервисов должно осуществляться через переменные окружения.

Существует возможность монтирования файлов, чаще всего используемая для монтирования сертификатов/ключей.

Журналирование

Приложения должны писать логи в stdout/stderr. Приложения не должны использовать файловую систему для хранения логов.

Для осуществления централизованного сбора и хранения, логирование должно осуществлять в формате json. Одна строчка - один json-объект.

Подробнее о требованиях к формату логирования в confluence.

Отказоустойчивость (Design for failure) и масштабирование

Для обеспечения масштабируемости, отказоустойчивости и обслуживаемости, микросервисы должны реализовывать

- Liveness, Readiness - пробы
- процедуры штатной остановки Graceful shutdown
- возможность работы в нескольких копиях (replicas)

Приложения на базе Spring Boot для реализации health-check'ов должны использовать Actuator

Stateless, Децентрализация данных

Проекты, запускаемые на платформе должны соответствовать принципам организации микросервисных приложений

- Микросервис не хранит состояние. Файловая система может использоваться для буферизации или организации кэширования.

- Хранение данных осуществляется во внешних по отношению к микросервисам хранилищах: PostgreSQL, MinIO, Redis, Keycloak и т.д.
- Один микросервис - одна база данных. Доступ к данным другого микросервиса - через его API

Поддерживаемые протоколы взаимодействия

- Протоколы на базе HTTP (REST, SOAP, websocket)
- Протоколы семейства gRPC (gRPC, gRPC-web)

При реализации протокола gRPC приложение должно уметь терминировать TLS на себе, используя предоставляемые через файловоую систему ключевую пару.

Пример конфигурирования Spring Boot приложения:

```
env:
  - name: GRPC_SERVER_SECURITY_ENABLED
    value: 'true'
  - name: GRPC_SERVER_SECURITY_CERTIFICATE_CHAIN
    value: 'file:/tls.crt'
  - name: GRPC_SERVER_SECURITY_PRIVATE_KEY
    value: 'file:/tls-key.pkcs8'
  - name: GRPC_TRUST_CERT_COLLECTION
    value: 'file:/tls.crt'
```

Подход API-first

Разработка приложений должна осуществляться по принципам api-first.

Хранение IDL-файлов (swagger, openapi, protobuf) дожно осуществляться в специализированном репозитории "api1", предоствляемом платформой каждому сервису.

Слабая связанность

Микросервис должен уметь работать в условиях отсутствия гарантии непрерываности доступности связанных узлов и адекватно обрабатывать эти ситуации.

Так, например, потеря соединения с БД или другим сервисов должна сопровождаться попыткой повторного подключения.

Аутентификация и авторизация

В межсервисном взаимодействии

В межсервисном взаимодействии должны быть реализованы аутентификация и авторизация на базе PaaS "Keycloak межсервисной авторизации".

Client'ская сторона должна:

- получать access токен
- хранить его до истечения времени действия access-токена
- получать новый access-токен заблаговременно, чтобы нивелировать последствия разницы во времени с принимающей стороной
- сопровождать все запросы к server'у полученным access токеном

Server'ная сторона должна:

- проверять наличие access-токена в запросе
- верифицировать время действия access-токена (время начала действия и истечения)
- верифицировать цифровую подпись токена с помощью открытого ключа, полученного от keycloak
- авторизовывать выполнение запросов, анализируя содержимое токена на предмет наличия там роли ACCESS своего keycloak client'a
- журналировать обращения, осуществляемые без должной аутентификации и авторизации

В сценариях пользовательской аутентификации

При реализации сценариев пользовательской авторизации в приложении:

- Должен быть использован один из PaaS Keycloak;
- Хранение пользователей должно осуществляться в Keycloak или провайдерах, интегрированных с ним;

- Ролевая модель (перечень ролей) и механника присвоения ролей пользователям должны быть описаны согласно инструкции
- Процедура аутентификации должны быть делегирована Keycloak (редирект на Keycloak для ввода логина и пароля)
- Микросервис должен принимаеть access-токен:
  - верифицировать время действия access-токена (время начала действия и истечения);
  - верифицировать цифровую подпись токена с помощью открытого ключа, полученного от keycloak;
  - авторизовывать выполнение запросов, анализируя содержимое токена на предмет наличия там соответствующих запросу ролей;
- Микросервис должен журналировать обращения, осуществляемые без должной аутентификации и авторизации

Сбор метрик

Для реализации сбора технических метрик приложения должны реализовывать exportner'ы согласно требованиям к протоколу prometheus.

Приложения Sprint Boot для реализации метрик должны использовать Actuator Observability

Выделение вычислительных мощностей

Микросервис должен декларировать минимально требуемые для его работы вычислительные мощности (CPU, RAM, disk space), а также устанавливать лимиты, предотвращающие последствия от утечек.

Подробнее с этим можно познакомиться в этом разделе и документации kubernetes

Термины и определения

К СВЕДЕНИЮ

Платформа Marlin - Совокупность программных средств, разработанных, установленных и настроенных в целях предоставления средств по разработке по разработке прикладного программного обеспечения по функциональным направлениям деятельности Заказчика по модели Платформа как сервис (PaaS)

Термин	Определение
Accelerator pack	На платформе реализована возможность управления конвейерами Jenkins для разных языков программирования прикладных сервисов с помощью Accelerator pack
Elasticsearch (ELK)	ELK используется для логирования прикладных сервисов на платформе
GitLab	Средство управления репозиториями кода для Git
Grafana	Инструмент мониторинга, который настраивается по одной инсталляции на среду в Платформе
Kafka	Распределённый программный брокер сообщений
Keycloak	Средства управления аутентификацией и идентификацией пользователей, а также авторизацией в сервисе, как пользователей, так и других сервисов
Kibana	Инструмент визуализации и изучения данных, который применяется для таких задач, как анализ журналов и временных рядов, мониторинг приложений и текущих процессов
Kubernetes	Средство исполнения и управления сервисами в виде контейнеров

Термин	Определение
MinIO	Высокопроизводительное объектное хранилище
Nexus	Репозиторий образов и сборок приложений, библиотек и зависимостей
PaaS	Platform as a Service - "платформа как услуга". Модель предоставления возможности использовать различные приложения и ресурсы: вычислительные сети, системы хранения, средства разработки. Платформенный сервис, предоставляющий различные услуги прикладным сервиса
PostgreSQL	Сервис баз данных для хранения данных компонентов Платформы
Quality Gates	Раздел Настроек в ПУ "Marlin", установленные значения пользователем показывают процент прохождения различных видов автотестирования(автотестрование, unit-тестирование), при которых будет совершены действия: автопереход в поставке на следуюзую среду при автотестах, возможность запуска развёртывания для юнит тестов
Sonarqube	ПО проверки качества исходного кода
Allure	Инструмент управляения ручным и автоматизированным тестированием
Зона	Совокупность программно-технических средств, на которых развернута Платформа с целью совершения операций по разработке и тестированию Платформы. В целевой архитектуре планируется три зоны - разработки, тестирования и промышленной эксплуатации
Среда	Совокупность программных средств Платформы (в рамках Зоны), предназначенная для поддержки и обеспечения одного

Термин	Определение
	из этапов жизненного цикла прикладного программного обеспечения, разработанного с использованием Платформы
Подписка	Декларирование зависимостей сервиса от сервиса. Необходима для учета взаимозависимостей прикладных сервисов
Приложение (проект)	Приложение является управленческой структурой, которая объединяет в себе несколько (микро) сервисов. Приложение необходимо для ведения реестра сервисов, выделенных для них ресурсов и управления командой с целью выделения доступа участникам команды к ресурсам приложения и входящих в его состав сервисов
Прикладной сервис	Сервис ПУ - это микросервис, который является частью приложения. . При создании сервиса автоматически создаются необходимые связанные объекты, набор которых в общем случае зависит от типа создаваемого сервиса
Сборка	Сборка - сущность Портала управления, которая объединяет версию исходного кода сервиса, процесс сборки с редактированием его шагов и результаты процесса сборки, включая артефакты и протоколы. Сборка осуществляется с помощью Jenkins и параметры ее выполнения зависят от Accelerator Pack, который был выбран при создании прикладного сервиса
Автосборка	Надстройка к сборке. Автосборка объединяет версию исходного кода сервиса, процесс сборки с редактированием его шагов и результаты процесса сборки, включая артефакты и протоколы
Тип сервиса	Тип прикладного сервиса определяет конвейер (pipeline) сборки и развертывания (микро)сервиса, возможные для автоматического выделения ресурсы Платформы (доступные для заказа сервисы PaaS)

Термин	Определение
Тип сервиса Backend	Backend - для сервиса доступны подписки как на платформенные, так и на прикладные сервисы, добавление развертываний и создание областей для хранения исходного кода, процессов
Тип сервиса External (legacy)	External (legacy) - для сервиса недоступны подписки на другие сервисы, добавление развертываний и создание областей для хранения исходного кода, процессов. Переменные окружения сервиса с типом Legacy передаются в шаблон развертывания сервиса-подписчика и настраиваются через интерфейс ПУ
Тип сервиса Front	Front - для сервиса доступны подписки на следующие платформенные сервисы: OpenShift и Keycloak, и подписки на прикладной сервис с типом Front
Тип сервиса Kafka	Kafka - для сервиса недоступна подписка на другие прикладные сервисы, доступна подписка только на PaaS Kafka (является средой исполнения для сервиса с данным типом), доступно создание сборки, поставки и развертывания и создание репозитория API для хранения версий IDL
Шаблон	Шаблон развертывания - это структура, являющаяся частью сервиса, созданная для управления переменными окружения сервиса через ПУ. Шаблон используется для создания развертывания и для создания шаблона развертывания сервиса-подписчика
Развертывание	Запуск микросервиса в его рабочей среде, то есть на сервере или хостинге

Marlin Command Line Interface

Download

- Windows: marlinctl-windows-amd64.exe
- MacOS: marlinctl-darwin-amd64
- Linux: marlinctl-linux-amd64
- Docker: `docker pull repo.vsk.ru:5013/marlin-platform/marlinctl:latest`

Installation

Linux

Quick install

```
sudo curl
https://gitlab.vsk.ru/api/v4/projects/2668/packages/generic/marlinctl/841115//marlinc
linux-amd64 -o /usr/local/bin/marlinctl && chmod +x /usr/local/bin/marlinctl
```

Setup auto-completion guide: https://cli.urfave.org/v2/examples/bash-completions/

Configuration

Create in home directory file `.marlinctl.yaml`

```
endpoint: grpc.marlin.vsk.ru:443
username: YOUR_USERNAME
password: YOUR_PASSWORD
```

Global options

Flag	Type	Description	Default	
--config, -C	string	Path to config file	~/.marlinctl.yaml	
--username, -U	string	Marlin username		
--password, -P	string	Marlin password		
--endpoint	string	Marlin endpoint (gRPC)	grpc.marlin.vsk.ru:443	
--loglevel	string	LogLevel (trace/debug/info/warn/error/fatal/panic)	info	

Commands

`get project / get projects`

Returns project list.

Authentication is required.

Flag	Type	Description	Default
--page	int	page number	1
--per	int	items per page	20
--output, -o	string	Change output. Possible values: default, json, yaml	short

Example

```
marlinctl get project --page=1 --per=100
```

Output example

```
                  ID                  |                NAME                |
OWNERID
--------------------------------------+------------------------------------+------
----------------
  166763a2-c6b3-4bde-919c-61057baa7e24 | ИМ ОСАГО                           | 1
  1c499aea-f8ed-4b1d-91a3-9dd97f766cf2 | ИМ ДВС                             | 1
  32543310-df84-408f-b14f-4e447c121a1e | DWH                                | 5
  00dd209e-8aba-442b-9f15-cd671d625abb | Creatio                            | 1
  68cf385b-ecae-464f-9cab-47ec4a57b170 | ПУМА                               | 1
  06f2332d-ca76-42cd-81ff-f350cebdfa19 | SettlementTeam                     | 1
  32deb7a0-5d60-4533-98e7-6284edb02ae6 | Part Api                           | 1
  aac283c2-ba49-4623-99b0-1059e783d90f | WebAuto                            | 1
  e18592ec-d5b2-4205-9526-9bc4848bafec | Файловый транспорт                 | 1
  7470a804-5641-4a49-bb19-86c18d86ef0f | Инновационные технологии           | 1
  9a631a62-efd1-4901-99a1-b94197bf3a8b | AdInsureServ                       | 2
  8737a955-4736-49c5-a062-bd70879bba36 | Инфошлюз 2.0                       | 3
  f4e975f6-79c1-4c58-9dfa-8bd217be8c33 | Сервис проверки контрагента в ЦСЮЛ | 1
  914fea1b-0fcb-4465-910b-eb0daa19c637 | Ais2.Osago                         | 3
  7ad194e4-76d6-47a5-98bb-dc9ff00601df | RDM Справочники                    | 1
  a98620a0-28d0-4b14-a9c1-458a2265769c | RDM Справочники (удалить)          | 1
  50586679-cf3f-4570-be17-e6f51d7f90a3 | Фабрика ОСАГО                      | 1
  b8b7523e-5dc9-40e9-bf67-c1f8bdab320a | НССО                               | 1
  dab61f97-d4fd-44e2-8640-68b17eae87bb | Фабрика Ипотеки                    | 1
  344cd265-fa19-414d-8d2f-25f2dc6f4dde | ВСС                                | 1
```

`create build`

Create build.

Authentication is required.

Flag	Type	Description	Default
--service	string	service code	
--title	string	build title	

Flag	Type	Description	Default
--commit	string	commit, branch or tag	
--dry-run	bool	prevent save artifacts to repository	false
--follow, -f	bool	follow display build logs in console	false
--check	string	enable quality gates. Possible values: units, quality	

Example

```
marlinctl create build --service=code --title="Example build" --commit=master --
dry-run true -f --check units --check quality
```

sync

Authentication is required. Administrative role is required

Flag	Type	Description	Default
--component	string	Components to sync. Possible values: all, sonarqube, ldap, kubernetes	

Example

```
marlinctl sync --component=ldap
```

Protocol Documentation

Table of Contents

- template/template_address_service.proto
  - AddressValidationRequest
  - AddressValidationResponse
  - DefaultAddressesRequest
  - GenerateAddressRequest
  - GeneratedAddresses
  - GeneratedTemplateAddress
  - ServiceKindAddress
  - ServiceKindAddresses
  - TlsReadiness
  - TemplateAddressService
- common/paas_deployment_id.proto
  - PaasDeploymentId
- common/auto_build_id.proto
  - AutoBuildId
- common/date.proto
  - Date
- common/order_direction.proto
  - OrderDirection
- common/service_id.proto
  - ServiceId
- common/build_id.proto

Top

system_components/exemplar_certificate_service.proto

AddCertificateRequest

Field	Type	Label	Description
systemComponentExemplarId	string		
privateKey	string		

Field	Type	Label	Description
certificate	string		

Certificate

Field	Type	Label	Description
id	string		
systemComponentExemplarId	string		
certificate	string		
secretPath	string		
sans	string	repeated	certificate subject alternative names

CertificateId

Field	Type	Label	Description
id	string		

Certificates

Field	Type	Label	Description
certificates	Certificate	repeated	

CertificatesRequest

Field	Type	Label	Description
systemComponentExemplarId	string		

ExemplarCertificateService

Method Name	Request Type	Response Type	Description
addCertificate	AddCertificateRequest	CertificateId	
getCertificates	CertificatesRequest	Certificates	
deleteCertificate	CertificateId	.google.protobuf.Empty	

Top

user_service.proto

User

Field	Type	Label	Description
userId	int64		
firstName	string		
lastName	string		
email	string		
phone	string		

UserRequest

Field	Type	Label	Description
userId	int64		

Users

Field	Type	Label	Description
users	User	repeated	
pagination	common.Pagination		

UsersRequest

Field	Type	Label	Description
excludeMemberOfProjectId	string		
pagination	common.Pagination		
orderDirection	common.OrderDirection		
orderBy	UsersRequest.OrderBy		

UsersRequest.OrderBy

Name	Number	Description
CREATED_AT	0	
FULL_NAME	1	

UserService

Method Name	Request Type	Response Type	Description
getUser	UserRequest	User	
getUsers	UsersRequest	Users	

Top

build/build_service.proto

Build

Field	Type	Label	Description
id	string		
status	Build.Status		
title	string		
description	string		
commit	string		
data	string		
serviceId	string		
userId	int64		
environmentId	string		
serviceReleaseId	string		
autoReleaseId	string		

Field	Type	Label	Description
createdAt	google.protobuf.Timestamp		
updatedAt	google.protobuf.Timestamp		
dryRun	bool		
checks	Check	repeated	

BuildLog

Field	Type	Label	Description
logs	bytes		
textSize	int64		
hasMoreData	bool		

BuildLogsRequest

Field	Type	Label	Description
buildId	string		
start	int64		

BuildUrls

Field	Type	Label	Description
PoseidonReportUrl	string		
ArtifactoryUrl	string		

Field	Type	Label	Description

BuildUrlsRequest

Field	Type	Label	Description
serviceId	string		
buildId	string		

Check

Field	Type	Label	Description
name	string		read only
code	string		
mandatory	bool		read only
active	bool		
status	Check.Status		read only
order	int32		read only

Build.Status

Name	Number	Description
S_UNSPECIFIED	0	
PENDING	1	

Name	Number	Description
RUNNING	2	
FAILED	3	
SUCCESS	4	
DELETING	5	
SKIPPED	6	
FAILED_TO_DELETE	7	

Check.Status

Name	Number	Description
S_UNSPECIFIED	0	
PENDING	1	
RUNNING	2	
SUCCESS	3	
SKIPPED	4	
FAILED	5	

BuildService

Method Name	Request Type	Response Type	Descri
getBuild	.ru.vsk.marlin.grpc.api.common.BuildId	Build	

Method Name	Request Type	Response Type	Descri
createBuild	Build	Build	
deleteBuild	.ru.vsk.marlin.grpc.api.common.BuildId	.google.protobuf.Empty	
restartBuild	.ru.vsk.marlin.grpc.api.common.BuildId	Build	
getBuildUrls	BuildUrlsRequest	BuildUrls	
getBuildLogs	BuildLogsRequest	BuildLog	

Top

release/release_service.proto

ReleaseTestingMarkRequest

Field	Type	Label	Description
releaseId	string		
environmentId	string		

ReleaseService

Method Name	Request Type	Response Type	Desc
markReleaseIsBeingTested	ReleaseTestingMarkRequest	.google.protobuf.Empty	
markReleaseIsTested	ReleaseTestingMarkRequest	.google.protobuf.Empty	

Top

keycloak_service.proto

KeycloakCredentials

Field	Type	Label	Description
clientId	string		
clientSecret	string		
url	string		
realm	string		

UpdateUserAuthRequest

Field	Type	Label	Description
serviceDeploymentId	string		
keycloakCode	string		

KeycloakService

Method Name	Request Type	Response Type	Descrip
updateUserAuth	UpdateUserAuthRequest	.google.protobuf.Empty	
updateServiceAuth	common.PaasDeploymentId	.google.protobuf.Empty	
getKeycloakCredentials	common.PaasDeploymentId	KeycloakCredentials	

Top

config_service.proto

Configurations

Field	Type	Label	Description
mapdata	Configurations.MapdataEntry	repeated	

Configurations.MapdataEntry

Field	Type	Label	Description
key	string		
value	bool		

ConfigService

Method Name	Request Type	Response Type	Description
getConfigurations	.google.protobuf.Empty	Configurations	

Top

service_service.proto

Service

Field	Type	Label	Description
id	string		
projectId	string		
acceleratorPackId	string		

Field	Type	Label	Description
code	string		
name	string		
description	string		
kindId	string		
jiraCmdb	string		
releaseDate	common.Date		
apiUrl	string		
codeUrl	string		
processUrl	string		
createdAt	google.protobuf.Timestamp		
updatedAt	google.protobuf.Timestamp		

ServiceCode

Field	Type	Label	Description
code	string		

ServiceCreationRequest

Field	Type	Label	Description
projectId	string		

Field	Type	Label	Description
acceleratorPackId	string		
code	string		
name	string		
description	string		
kindId	string		
jiraCmdb	string		
releaseDate	common.Date		

ServiceService

Method Name	Request Type	Response Type	Description
createService	ServiceCreationRequest	Service	
deleteService	common.ServiceId	.google.protobuf.Empty	
getServiceByCode	ServiceCode	Service	

Top

autobuild/auto_build_service.proto

AutoBuild

Field	Type	Label	Description
id	google.protobuf.StringValue		

Field	Type	Label	Description
serviceId	string		
name	string		
branch	string		
tag	string		
validationsCodes	string	repeated	
autoReleaseId	google.protobuf.StringValue		
updatedAt	google.protobuf.Timestamp		

AutoBuilds

Field	Type	Label	Description
builds	AutoBuild	repeated	

AutoBuildService

Method Name	Request Type	Response Type
getAutoBuilds	.ru.vsk.marlin.grpc.api.common.ServiceId	AutoBuilds
createAutoBuild	AutoBuild	AutoBuild
updateAutoBuild	AutoBuild	AutoBuild
deleteAutoBuild	.ru.vsk.marlin.grpc.api.common.AutoBuildId	.google.protobuf.Empty

Top

template/template_service.proto

Address

Field	Type	Label	Description
code	string		
name	string		
host	string		
id	string		
port	google.protobuf.Int32Value		

MappingValue

Field	Type	Label	Description
type	MappingValue.MappingType		
keyName	string		
mappingName	string		
id	google.protobuf.StringValue		
sourceType	MappingValue.SourceType		
sourceId	string		

Monitor

Field	Type	Label	Description
path	string		
port	uint32		
interval	string		
type	MonitorType		

PaasSubscription

Field	Type	Label	Description
id	string		
platformServiceDeploymentId	string		
platformServiceId	string		
options	string		

Parameter

Field	Type	Label	Description
secret	bool		
value	string		
key	string		
envVarId	google.protobuf.StringValue		
paramId	google.protobuf.StringValue		

ServiceTemplatesRequest

Field	Type	Label	Description
serviceId	string		
pagination	ru.vsk.marlin.grpc.api.common.Pagination		
orderDirection	ru.vsk.marlin.grpc.api.common.OrderDirection		
active	bool		
search	string		
orderBy	ServiceTemplatesRequest.OrderBy		

Subscription

Field	Type	Label	Description
serviceId	string		
templateId	string		

Template

Field	Type	Label	Description
name	string		
environmentId	string		
active	bool		
description	string		

Field	Type	Label	Description
replicas	int32		
addresses	Address	repeated	
subscriptions	Subscription	repeated	
paasSubscriptions	PaasSubscription	repeated	
parameters	Parameter	repeated	
serviceId	string		
containerParams	string		
mappingValue	MappingValue	repeated	
id	google.protobuf.StringValue		
busy	bool		
createdAt	google.protobuf.Timestamp		
updatedAt	google.protobuf.Timestamp		
validity	bool		
monitors	Monitor	repeated	

TemplateId

Field	Type	Label	Description
id	string		

Templates

Field	Type	Label	Description
templates	Template	repeated	
pagination	ru.vsk.marlin.grpc.api.common.Pagination		

MappingValue.MappingType

Name	Number	Description
ENVIRONMENT_VARIABLE	0	
FILE	1	

MappingValue.SourceType

Name	Number	Description
PLATFORM_SERVICE	0	
SERVICE	1	
PARAMETER	2	

MonitorType

Name	Number	Description
MONITOR_TYPE_UNSPECIFIED	0	
POD_MONITOR	1	
SERVICE_MONITOR	2	

Name	Number	Description

ServiceTemplatesRequest.OrderBy

Name	Number	Description
CREATED_AT	0	
NAME	1	

TemplateService

Method Name	Request Type	Response Type	Description
getTemplate	TemplateId	Template	
getTemplates	ServiceTemplatesRequest	Templates	
createTemplate	Template	Template	
updateTemplate	Template	Template	
deleteTemplate	TemplateId	.google.protobuf.Empty	

Top

template/template_address_service.proto

AddressValidationRequest

Field	Type	Label	Description
serviceId	string		

Field	Type	Label	Description
environmentId	string		
host	string		
templateId	string		

AddressValidationResponse

Field	Type	Label	Description
isValid	bool		
isUnique	bool		
dnsIsReady	bool		
tlsReadiness	TlsReadiness		

DefaultAddressesRequest

Field	Type	Label	Description
serviceId	string		
environmentId	string		

GenerateAddressRequest

Field	Type	Label	Description
serviceId	string		
environmentId	string		

Field	Type	Label	Description
addressCode	string		
unsavedAddresses	GeneratedTemplateAddress	repeated	

GeneratedAddresses

Field	Type	Label	Description
addresses	GeneratedTemplateAddress	repeated	

GeneratedTemplateAddress

Field	Type	Label	Description
name	string		
code	string		
host	string		

ServiceKindAddress

Field	Type	Label	Description
code	string		
name	string		

ServiceKindAddresses

Field	Type	Label	Description
addresses	ServiceKindAddress	repeated	

TlsReadiness

Name	Number	Description
TLS_UNSPECIFIED	0	
CERTIFICATE_FOUND	1	
CERTIFICATE_IS_ISSUABLE	2	
CERTIFICATE_IS_UNISSUABLE	3	

TemplateAddressService

Method Name	Request Type	Response
generateAddress	GenerateAddressRequest	GeneratedTempl
validateAddress	AddressValidationRequest	AddressValidatio
generateDefaultAddresses	DefaultAddressesRequest	GeneratedAddre
fillAllTemplateAddressHosts	.google.protobuf.Empty	.google.protobuf.
getServiceKindAddresses	.ru.vsk.marlin.grpc.api.common.ServiceId	ServiceKindAddr

Top

common/paas_deployment_id.proto

PaasDeploymentId

Field	Type	Label	Description
id	string		

Top

common/auto_build_id.proto

AutoBuildId

Field	Type	Label	Description
id	string		

Top

common/date.proto

Date

Field	Type	Label	Description
year	int32		
month	int32		
day	int32		

Top

common/order_direction.proto

OrderDirection

Name	Number	Description
DESC	0	
ASC	1	

Top

common/service_id.proto

ServiceId

Field	Type	Label	Description
id	string		

Top

common/build_id.proto

BuildId

Field	Type	Label	Description
id	string		

Top

common/pagination.proto

Pagination

Field	Type	Label	Description
currentPage	int32		
itemsPerPage	int32		
totalPages	int32		
totalItems	int32		

Top

common/deployment_id.proto

DeploymentId

Field	Type	Label	Description
id	string		

Top

common/auto_release_id.proto

AutoReleaseId

Field	Type	Label	Description
id	string		

Top

subscription/paas_subscription_service.proto

PaasDeploymentRequest

Field	Type	Label	Description
servicePaasSubscriptionId	string		
environmentId	string		

ServicePaasSubscription

Field	Type	Label	Description
id	string		
serviceId	string		
paasId	string		

ServicePaasSubscriptionId

Field	Type	Label	Description
id	string		

ServicePaasSubscriptionRequest

Field	Type	Label	Description
serviceId	string		
paasId	string		

PaasSubscriptionService

Method Name	Request Type	
createPaasSubscription	ServicePaasSubscriptionRequest	ServicePaas
deletePaasSubscription	ServicePaasSubscriptionId	.google.prot
createPaasDeployment	PaasDeploymentRequest	.ru.vsk.mar
deletePaasDeployment	.ru.vsk.marlin.grpc.api.common.PaasDeploymentId	.google.prot

Top

subscription/subscription_service.proto

ServiceSubscription

Field	Type	Label	Description
id	string		
receiverId	string		
initiatorId	string		

ServiceSubscriptionId

Field	Type	Label	Description
id	string		

ServiceSubscriptionRequest

Field	Type	Label	Description
receiverId	string		
initiatorId	string		
serviceSubscriptionType	ServiceSubscriptionType		

ServiceSubscriptionType

Name	Number	Description
ST_UNSPECIFIED	0	
PRODUCER	1	
CONSUMER	2	

SubscriptionService

Method Name	Request Type	Response Type	Desc
createServiceSubscription	ServiceSubscriptionRequest	ServiceSubscription	
deleteServiceSubscription	ServiceSubscriptionId	.google.protobuf.Empty	

Top

kubernetes/kubernetes_service.proto

DeploymentStatus

Field	Type	Label	Description
namespace	string		
deployment	string		
conditions	V1DeploymentCondition	repeated	

V1DeploymentCondition

Field	Type	Label	Description
lastTransitionTime	google.protobuf.Timestamp		
lastUpdateTime	google.protobuf.Timestamp		
message	string		
reason	string		
status	string		
type	string		

KubernetesService

Method Name	Request Type	Response Type	Descrip
getStatus	.ru.vsk.marlin.grpc.api.common.DeploymentId	DeploymentStatus	

Top

service_deployment_service.proto

GetDeploymentUrlsResponse

Field	Type	Label	Description
kubernetes	string		
kibana	string		
apis	GetDeploymentUrlsResponse.ServiceAddress	repeated	

GetDeploymentUrlsResponse.ServiceAddress

Field	Type	Label	Description
title	string		
url	string		

ServiceDeploymentService

Method Name	Request Type	Response Type	Descriptic
getDeploymentUrls	common.DeploymentId	GetDeploymentUrlsResponse	

Top

project_member_service.proto

MemberCreationRequest

Field	Type	Label	Description
projectId	string		

Field	Type	Label	Description
userId	int64		

MemberId

Field	Type	Label	Description
id	string		

MembersRequest

Field	Type	Label	Description
projectId	string		
orderBy	MembersRequest.OrderBy		
orderDirection	common.OrderDirection		
pagination	common.Pagination		

ProjectMember

Field	Type	Label	Description
id	string		
projectId	string		
userId	int64		
accessLevel	ProjectMember.AccessLevel		

ProjectMembers

Field	Type	Label	Description
ProjectMembers	ProjectMember	repeated	
pagination	common.Pagination		

MembersRequest.OrderBy

Name	Number	Description
CREATED_AT	0	
USER_FULL_NAME	1	

ProjectMember.AccessLevel

Name	Number	Description
OWNER	0	
USER	1	

ProjectMemberService

Method Name	Request Type	Response Type	Description
addMember	MemberCreationRequest	ProjectMember	
deleteMember	MemberId	.google.protobuf.Empty	
getMember	MemberId	ProjectMember	
getMembers	MembersRequest	ProjectMembers	

Method Name	Request Type	Response Type	Description

Top

integration/integration_service.proto

Integration

Field	Type	Label	Description
id	string		
integrableType	IntegrableType		
integrableId	string		
kind	IntegrationKind		
status	IntegrationStatus		

IntegrationId

Field	Type	Label	Description
id	string		

Integrations

Field	Type	Label	Description
integrations	Integration	repeated	

IntegrationsRequest

Field	Type	Label	Description
integrableId	string		
integrableType	IntegrableType		

IntegrableType

Name	Number	Description
INTEGRABLE_TYPE_UNSPECIFIED	0	
PROJECT	1	
SERVICE	2	
ACCELERATOR_PACK	3	
PLATFORM_SERVICE	4	
MEMBER	5	

IntegrationKind

Name	Number	Description
KIND_UNSPECIFIED	0	
CONFLUENCE	1	
GITLAB	2	
JENKINS	3	

IntegrationStatus

Name	Number	Description
STATUS_INTEGRATION_UNSPECIFIED	0	
PENDING	1	
RUNNING	2	
FAILED	3	
SUCCESS	4	

IntegrationService

Method Name	Request Type	Response Type	Description
getIntegrations	IntegrationsRequest	Integrations	
restartFailedIntegration	IntegrationId	Integration	

Top

test/test_settings_service.proto

IntegrationTestSettings

Field	Type	Label	Description
environmentId	string		
testingType	string		
threshold	int32		

ServiceId

Field	Type	Label	Description
id	string		

TestSettings

Field	Type	Label	Description
serviceId	string		
integrationTestEnabled	bool		
intTestManualSkipAllowed	bool		
integrationTestSettings	IntegrationTestSettings	repeated	
unitTestsCoverage	int32		

TestSettingsService

Method Name	Request Type	Response Type	Des
findOrCreateTestSettings	.ru.vsk.marlin.grpc.api.common.ServiceId	TestSettings	
updateTestSettings	TestSettings	TestSettings	

Top

test/integration_test_service.proto

GetTestsLaunchesRequest

Field	Type	Label	Description
deploymentId	string		
limit	int32		

MessageDetailsResult

Field	Type	Label	Description
id	string		
deploymentId	string		
testingType	string		

ServiceTest

Field	Type	Label	Description
id	string		
testingType	string		
status	TestingStatus		
logUrls	string	repeated	
successTestCount	int32		
failedTestCount	int32		
totalTestCount	int32		
manualApproved	bool		

Field	Type	Label	Description
message	string		
stacktrace	string		
updatedAt	google.protobuf.Timestamp		
createdAt	google.protobuf.Timestamp		

ServiceTestsLaunch

Field	Type	Label	Description
id	string		
deploymentId	string		
tests	ServiceTest	repeated	
createdAt	google.protobuf.Timestamp		

ServiceTestsLaunches

Field	Type	Label	Description
launches	ServiceTestsLaunch	repeated	

TestingStatusMessage

Field	Type	Label	Description
marlinId	google.protobuf.StringValue		
status	TestingStatus		

Field	Type	Label	Description
message	string		
stacktrace	string		
success	int32		
failed	int32		
total	int32		
urls	string	repeated	

TestingStatus

Name	Number	Description
TS_UNDEFINED	0	
IN_PROGRESS	1	
FINISHED	2	
ERROR	3	
STOPPING	4	
STOPPED	5	

IntegrationTestService

Method Name	Request Type	Response Type
startTest	.ru.vsk.marlin.grpc.api.common.DeploymentId	MessageDetailsResult

Method Name	Request Type	Response Type
stopTest	.ru.vsk.marlin.grpc.api.common.DeploymentId	MessageDetailsResult
getTestsLaunches	GetTestsLaunchesRequest	ServiceTestsLaunches
updateTestStatus	TestingStatusMessage	.google.protobuf.Empt

Top

project_service.proto

Project

Field	Type	Label	Description
id	string		
name	string		
code	string		
description	string		
jiraCmdb	string		
apiUrl	string		
codeUrl	string		
processUrl	string		
ownerId	int64		
createdAt	google.protobuf.Timestamp		

Field	Type	Label	Description
updatedAt	google.protobuf.Timestamp		

ProjectCreationRequest

Field	Type	Label	Description
name	string		
code	string		
description	string		
jiraCmdb	string		
ownerId	int64		

ProjectId

Field	Type	Label	Description
id	string		

Projects

Field	Type	Label	Description
projects	ReducedProject	repeated	
metaInf	common.Pagination		

ProjectsRequest

Field	Type	Label	Description
pagination	common.Pagination		
search	string		
filterByCurrentUser	bool		
orderBy	string		

ReducedProject

Field	Type	Label	Description
id	string		
name	string		
createdAt	google.protobuf.Timestamp		
ownerId	int64		
iconUrl	string		

ProjectService

Method Name	Request Type	Response Type	Description
getProjects	ProjectsRequest	Projects	
getProject	ProjectId	Project	
createProject	ProjectCreationRequest	Project	
deleteProject	ProjectId	.google.protobuf.Empty	

Method Name	Request Type	Response Type	Description
updateProject	Project	Project	updates only 3 fields: name, description, jiraCmdb.

Top

rolesync/role_sync.proto

FullSyncRequest

Field	Type	Label	Description
syncComponent	SyncComponent		

SyncComponent

Name	Number	Description
ALL	0	
SONAR	1	
LDAP	2	
KUBERNETES	3	

RoleSyncService

Method Name	Request Type	Response Type	Description
fullSync	FullSyncRequest	.google.protobuf.Empty	

Top

autorelease/auto_release_service.proto

AutoDeployment

Field	Type	Label	Description
id	google.protobuf.StringValue		
environmentId	string		
templateId	string		
createdAt	google.protobuf.Timestamp		
updatedAt	google.protobuf.Timestamp		

AutoRelease

Field	Type	Label	Description
id	google.protobuf.StringValue		
name	string		
enabled	bool		
majorVersion	int32		
minorVersion	int32		

Field	Type	Label	Description
bugFixVersion	int32		
buildVersion	int32		
versionIncrement	int32		
serviceId	string		
commitApi	string		
autoDeployments	AutoDeployment	repeated	
validations	string	repeated	
createdAt	google.protobuf.Timestamp		
updatedAt	google.protobuf.Timestamp		

AutoReleases

Field	Type	Label	Description
autoReleases	AutoRelease	repeated	

AutoReleaseService

Method Name	Request Type	Response T
getAutoRelease	.ru.vsk.marlin.grpc.api.common.AutoReleaseId	AutoRelease
getAutoReleases	.ru.vsk.marlin.grpc.api.common.ServiceId	AutoReleases
createAutoRelease	AutoRelease	AutoRelease

Method Name	Request Type	Response T
updateAutoRelease	AutoRelease	AutoRelease
deleteAutoRelease	.ru.vsk.marlin.grpc.api.common.AutoReleaseId	.google.protobu
startAutoRelease	.ru.vsk.marlin.grpc.api.common.BuildId	.google.protobu
handleAutoTestSuccess	.ru.vsk.marlin.grpc.api.common.DeploymentId	.google.protobu

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	P
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use	int32	int	int	int32	int	intege

.proto Type	Notes	C++	Java	Python	Go	C#	P
	sint32 instead.						
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	intege
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	intege
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	intege
sint32	Uses variable-	int32	int	int	int32	int	intege

.proto Type	Notes	C++	Java	Python	Go	C#	P
	length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.						
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	intege
fixed32	Always four bytes. More	uint32	int	int	uint32	uint	intege

.proto Type	Notes	C++	Java	Python	Go	C#	P
	efficient than uint32 if values are often greater than 2^28.						
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	intege
sfixed32	Always four bytes.	int32	int	int	int32	int	intege
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	intege
bool		bool	boolean	boolean	bool	bool	boolea
string	A string must	string	String	str/unicode	string	string	string

.proto Type	Notes	C++	Java	Python	Go	C#	P
	always contain UTF-8 encoded or 7-bit ASCII text.						
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Функциональные характеристики Платформы "Marlin"

1. Введение

Настоящий документ описывает функциональные характеристики программного обеспечения Платформы "Marlin" (далее - Marlin, Платформа) в части ее базового функционала, а также содержит информацию, необходимую для её эксплуатации.

Документ с общим описанием Платформы состоит из следующих разделов:

1. Раздел «Назначение Системы» содержит сведения о назначении Платформы и ее функциональных возможностях.
2. Раздел «Требования к программному обеспечению компьютера пользователя» минимальные требования к программному обеспечению, необходимые для корректной работы Платформы.
3. Раздел «Выполнение Системы» указана последовательность действий, обеспечивающих загрузку, запуск, выполнение и завершение Платформы, приведено описание функций, формата и возможных вариантов команд, с помощью которых осуществляется загрузка и управление выполнением Платформы, а также ответы Платформы на эти команды.

2. Назначение системы

Платформа "Marlin" (далее - Платформа) предназначена для разработки прикладного программного обеспечения с предоставлением необходимых в ходе разработки, тестирования и эксплуатации вычислительных ресурсов по модели «платформа как сервис».

Платформа предназначена для автоматизации деятельности в области:

1. Сборки, тестирования и развертывания ПО;
2. Регистрации приложений, сервисов, процессов;
3. Реализации доступа разработчика к репозиториям исходного кода и библиотек;
4. Выделения ресурсов для управления сервисами;

5. Реализации зон разработки, тестирования и зоны постоянной эксплуатации Платформы;
6. Реализации общих инфраструктурных сервисов (развертывание системных компонент, подготовка скриптов развертывания, оказание консультаций для развертывания).

Процесс разработки состоит из повторяющегося набора подпроцессов в Платформе:

1. Анализ требований и проектирование;
2. Реализация проекта в программном коде и документации приложения;
3. Проверка корректности реализации и соответствия требованиям;
4. Размещение приложения в продуктивной среде;
5. Анализ результатов.

Для разработки выбрана парадигма API First, которая определяет последовательность, в которой создаются публичные API системы, реализуются заявленные функции и происходит ввод системы в эксплуатацию. На этом этапе определяется компонентный состав системы, интерфейсы взаимодействия частей системы между собой, внешними зависимостями и т.д. После процесса проектирования производится реализация спроектированных компонентов/функций, производится документирование, покрытие тестами и первоначальное тестирование разработанных компонентов/функций. После реализации требований должен быть произведен контроль качества реализации. Контроль включает в себя как ручные, так и автоматизированные процедуры. После успешного прохождения контроля качества, приложение готово к развертыванию в продуктивной среде. Для обеспечения непрерывности функционирования, в среде могут выполняться несколько версий приложения, что накладывает дополнительные требования к разработке.

3. Характеристика функциональной структуры

Платформа представляет собой программный продукт, адаптированный для работы в различных операционных системах (ОС). Компоненты, входящие в состав Платформы, основаны на клиент-серверной архитектуре и современные (последние) версии всех браузеров и клиентских ОС. Платформа адаптирована для функционирования внутри защищенной сети без доступа (или с ограниченным доступом) в информационно-телекоммуникационную сеть Интернет.

3.1 Базовый функционал

В Платформе реализована следующая функциональность:

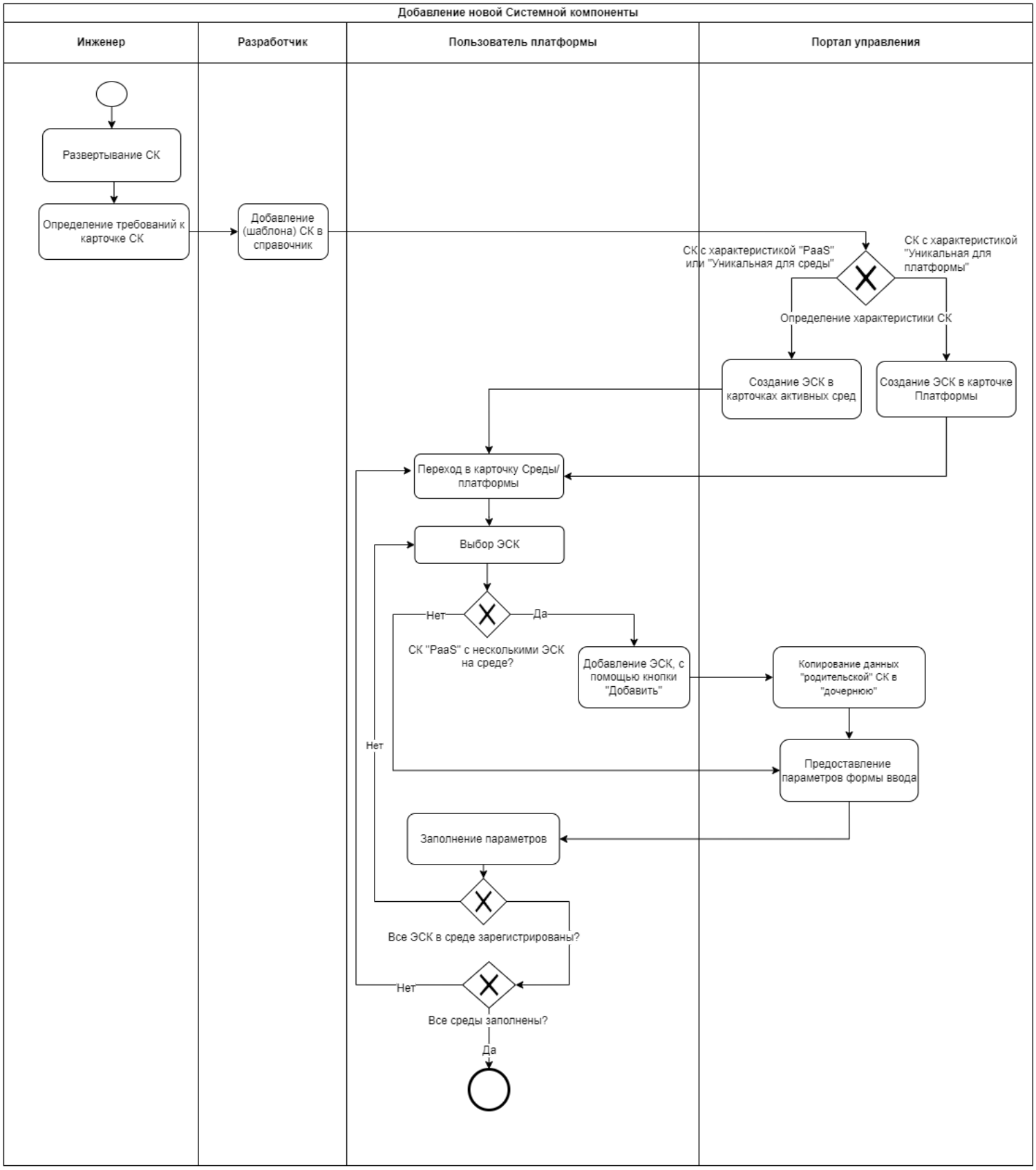
1. Авторизация пользователя;
2. Регистрация приложения;
3. Управления участниками команды;
4. Регистрация сервиса;
5. Запуск интеграции с информационной системой;
6. Управление поставкой сервиса;
7. Управление подписками сервиса;
8. Создание и запуск сборки на среде разработки;
9. Запуск развертывания;
10. Хранение секретов в HashiCorp Vault.

3.2 Интеграция Платформы с системами

Платформа интегрирована со многими сторонними системами и имеет возможность гибко перенастраивать как имеющиеся подключения к системам, так и подключать новые. Системы, которые предоставляются Платформой в виде услуг, заводятся как системные компоненты (далее - СК).

Для создания СК в Платформе необходимо выполнение инженерных работ по развертыванию/установке новой (версии) СК на вычислительных ресурсах, выделенных для Платформы. После этого её необходимо зарегистрировать в ПУ: добавить в справочник СК ПУ и добавить параметры подключения в карточке Среды. В карточке СК указываются параметры системной компоненты (обще и средозависимые), в том числе секретные. Секретные параметры должны маскироваться в ПУ в соответствии с правами доступа пользователей.

Процесс управления СК в Платформе представлен ниже:



СК может быть в единственном экземпляре на всю Платформу. Чтобы настроить каждую систему под свою среду, необходимо создать экземпляр системной компоненты (далее - ЭСК). В Платформе 4 среды, и для каждой из них может создано по одной (или несколько) ЭСК. Для каждой ЭСК на среде заполняются параметры

подключения и вносятся все необходимые данные для корректной работы Системы-Платформы.

Системы, которые предоставляют услуги в Платформе (такие как MinIO, Keycloak и т.д.), отображаются в качестве платформенных сервисов (далее - PaaS-сервисов). При добавлении подписки на PaaS-сервис, параметры его развертывания передаются в шаблон развертывания сервиса-подписчика, при этом возможность добавления развертывания для сервиса-подписчика контролируется наличием/отсутствием развертывания платформенного сервиса-подписки на соответствующей среде. Для добавления развертывания сервиса, имеющего подписку на PaaS, необходимо добавить развертывание PaaS для среды, в которой должен быть развернут сервис. Для сервисов PaaS реализовано ведение реестра сервисов-подписчиков.

3.2.1 Взаимодействие Платформы с Jenkins

Jenkins позволяет автоматизировать часть процесса разработки программного обеспечения, в котором не обязательно участие человека, обеспечивая функции непрерывной интеграции. Jenkins в Платформе представлен как отдельная СК, также в каждой среде создана ЭСК Jenkins. В карточке ЭСК Jenkins необходимы следующие параметры подключения для корректной работы симбиоза Платформа-Jenkins:

1. Учетная запись.
2. Токен - токен для доступа к API Jenkins.
3. Ссылка - ссылка на web-интерфейс Jenkins.
4. Порт - порт для доступа к API Jenkins.
5. Репозиторий - путь до git-репозитория с pipeline для сборки приложения.
6. Имя ветки репозитория - имя ветки в git-репозитории.
7. CredentialID - credentialID для доступа из Jenkins в GitLab.
8. Build jenkinsfile - путь в репозитории git до build_pipeline (включая название файла).
9. Deploy jenkinsfile - путь в репозитории git до deploy_pipeline (включая название файла).
10. Stop_start jenkinsfile - путь в репозитории git до stop_start_pipeline (включая название файла).
11. Destroy jenkinsfile - путь в репозитории git до stop_start_pipeline (включая название файла).

12. Upgrade jenkinsfile - путь в репозитории git до upgrade_pipeline (включая название файла).
13. Put_vars jenkinsfile - путь в репозитории git до put_vars_pipeline (включая название файла).
14. Delete_vars jenkinsfile - путь в репозитории git до delete_vars_pipeline (включая название файла).

При запуске развертывания/сборки в ПУ автоматически формируется JSON с параметрами, необходимыми для выполнения pipeline (указаны в переменной DATA, этот JSON передается Jenkins.

На данный момент реализованы pipeline Jenkins, которые позволяют автоматизировать следующие процессы:

1. Сборка прикладного сервиса:
    - проверок:
        - проверка кода сервиса на качество (SonarQube);
        - проверка план-факт API для протоколов:
            - gRPC;
            - OpenAPI.
        - проверка корректности межсервисной авторизации;
        - юнит-тесты;
        - сбор зависимостей;
        - сборка из открытого или закрытого репозитория;
        - выкладывание собранных артефактов (сервис и его Docker образ) в Nexus Repository:
            - среда разработки;
            - среда интеграционного тестирования.
2. Создание развертывания прикладного сервиса;
    - выбор среды исполнения:
        - OpenShift;
    - Проливки в заказанные PaaS:

```
+ Keycloak межсервисная авторизация;
+ Keycloak пользовательская авторизация;
+ MinIO;
+ PostgreSQL.
```

- Остановка развертывания прикладного сервиса;
- Запуск развертывания прикладного сервиса (после остановки);
- Удаление развертывания прикладного сервиса;

3. Управление подписками.

Также в Платформе реализована возможность управления конвейерами Jenkins для разных языков программирования прикладных сервисов с помощью Accelerator pack. Параметры для передачи в Jenkins определяются в зависимости от указанного Accelerator pack.

3.2.2 Взаимодействие Платформы с GitLab

GitLab представляет собой веб-приложение и систему управления репозиториями программного кода для распределенной системы контроля версий Git. GitLab, как правило, используется с Git, что позволяет разработчикам сохранять написанный код в онлайн-формате и работать с другими разработчиками над разными проектами. GitLab позволяет взаимодействовать с репозиториями, управлять правами доступа и пользователями, отслеживать ошибки, автоматизировать процессы и выполнять многие другие операции.

GitLab в Платформе представлен как отдельная СК, с типом "Уникальная в Платформе". Ниже перечислены параметры необходимые для корректного подключения Платформы к GitLab:

1. Ссылка;
2. API URL;
3. Приватный токен.

3.2.3 Взаимодействие Платформы с MinIO

MinIO в Платформе представлен в качестве PaaS-сервиса, с помощью которого осуществляется выделение ресурсов для хранения неструктурированных данных. MinIO - реализация протокола AS3, в соответствии с которым передача файлов осуществляется по SSL. В рамках Платформы реализованы 4 кластера MinIO, по одному для каждой среды (т.е. 1 СК MiniO на Платформу и 4 ЭСК на каждую среду).

Ресурсы для прикладного сервиса в среде разработки выделяются Платформой автоматически при создании подписки на сервис PaaS MinIO.

Выделение ресурсов на последующих средах необходимо произвести самостоятельно. При добавлении развертывания для сервиса PaaS MinIO формируются следующие переменные окружения:

1. USERACCESSKEY - наименование пользователя;
2. USERSECRETKEY - секретный;
3. URL - адрес подключения;
4. BUCKET - наименование хранилища.

При развертывании прикладного сервиса в выделенных ресурсах автоматически проводится настройка и заполнение данными (т.е. проливка). Пользователю может быть доступно несколько bucket. По умолчанию для каждого bucket выделен определенный объем памяти, который не регулируется в рамках Платформы.

3.2.4 Взаимодействие Платформы с Keycloak

Keycloak предоставляется Платформой, как платформенный сервис (PaaS) и является средством управления:

1. Аутентификацией и идентификацией пользователей;
2. Авторизацией в сервисе:
    - пользователей;
    - других сервисов.

Ресурсы для прикладного сервиса в среде разработки выделяются Платформой автоматически при создании подписки на сервис PaaS Keycloak. Выделение ресурсов на последующих средах производится вручную. При развертывании прикладного сервиса в выделенных ресурсах автоматически проводится настройка и заполнение данными (проливка).

3.2.5 Взаимодействие Платформы с PostgreSQL

PostgreSQL - система управления реляционными базами данными, которая предоставляется Платформой, как платформенный сервис (PaaS).

Ресурсы для прикладного сервиса в среде разработки выделяются Платформой автоматически при создании подписки на сервис PaaS PostgreSQL. Выделение ресурсов на последующих средах необходимо произвести самостоятельно.При развертывании

прикладного сервиса в выделенных ресурсах автоматически проводится настройка и заполнение данными (проливка).

При использовании PaaS, в чарт в values.yml и в необходимые шаблоны (т.е., например, шаблон для deployment) необходимо добавить переменные окружения в следующем формате:

1. Host
2. User
3. Password

3.3 Средства мониторинга и управления

3.3.1 Системный мониторинг (мониторинг элементов)

Основой модели уровня мониторинга является системный мониторинг, который измеряет все элементы приложения и инфраструктуры. Мониторинг системы отличается от других видов мониторинга через элементарный уровень, поэтому этот мониторинг также называется мониторингом элементов. Измеряются только отдельные компоненты приложения и инфраструктуры без их активации. Этот мониторинг необходим для технического управления инфраструктурой. Системный мониторинг состоит из следующих функций мониторинга.

Функция	Определение
Услуга мониторинга	Мониторинг как сервисов инфраструктуры, так и сервисов приложений. Услуги измеряются с помощью протокола управления, обычно это простой протокол управления сетью (SNMP)
Ресурс мониторинга	Мониторинг одного компонента за поведением ресурсов, таких как центральный процессор (ЦП), память и пропускная способность сети
Buildin мониторинга	Встроенный мониторинг - это средство мониторинга, которое запрограммировано как часть приложения. Дополнительным достоинством этого средства мониторинга является то, что

Функция	Определение
	измеряются счетчики времени выполнения приложения, которые не видны за пределами приложения
События мониторинга	Мониторинг событий включает в себя все компоненты инфраструктуры и службы инфраструктуры, а также события приложений. События обычно можно прочитать из локальных файлов журнала, таких как журнал приложения, журнал безопасности и файл журнала ошибок

3.3.2 Мониторинг приложений

Мониторинг приложений предоставляет информацию об образе отдельного приложения (подсистемы) или сервиса, измеряя базовую систему (мониторинг системы), сервисы инфраструктуры и интерфейсы приложений. Этот уровень важен для управления приложениями, поскольку он отображает всю необходимую информацию из одного приложения в одном представлении. Мониторинг приложения состоит из функции системного монитора, а также следующих функций монитора.

Функция	Определение
Мониторинг интерфейса приложения	Мониторинг интерфейса приложения специально предназначен для администраторов приложений, чтобы они могли с одного взгляда определить, что приложение работает в соответствии с ожиданиями. Следовательно, необходимо для каждого приложения определять, какие компоненты инфраструктуры, службы инфраструктуры и службы приложений являются важными. Кроме того, измерения выполняются на уровне приложения, которое проверяет взаимодействие между инфраструктурой и кодом приложения
Инфраструктура Услуга мониторинга	Даже если услуга инфраструктуры «живая», это не означает, что она также работает должным образом или даже доступна. Чтобы установить, что служба работает, можно выполнить динамический тест на уровне инфраструктуры, на которой вызывается служба

3.3.3 Мониторинг информационной системы

Для выполнения пользовательской функции требуется больше приложений. Они не всегда могут быть идентифицированы пользователем отдельно. Чтобы сделать пользовательские соглашения на этом уровне, уровень мониторинга информационной системы также определяется в приложении на уровне мониторинга приложения. Это, по сути, цепочка приложений. Измеряя информационную систему E2E, можно наблюдать как можно ближе от пользователя. Эти измерения выполняются путем прохода по пути транзакций. Поэтому мониторинг на этом уровне также называется мониторингом транзакций.

Для поддержки этого измерения также проводятся измерения инфраструктуры E2E. Мониторинг инфраструктуры EXNUMEXE показывает, что между различными компонентами инфраструктуры существует связь. Это особенно справедливо в географически разделенных местах и крупных вычислительных центрах с более логическими сетями (локальная сеть (LAN)), глобальной сетью (WAN) и т. Д.

Мониторинг информационной системы был построен в дополнение к мониторингу приложений со следующими функциями.

Функция	Определение
Информация Система Мониторинга E2E	Наличие приложения еще не означает, что пользователь может работать. Часто приложения объединяются в полные цепочки (информационные системы).
Инфраструктура Мониторинга E2E	Инфраструктура мониторинга E2E включает измерение всего уровня инфраструктуры. Этот мониторинг часто используется в географически разделенных единицах (WAN).
Домен инфраструктуры Мониторинга	В случае, если есть несколько поставщиков или сеть разделена на физически / логически разделенные домены, часто желательно контролировать эти домены отдельно с точки зрения доступности и т. Д. Также могут быть другие стандарты для этих доменов, такие как демилитаризованная зона (DMZ).) по сравнению с сетью автоматизации делопроизводства.

3.3.4 Контроль цепей

Цепной мониторинг нацелен на то, чтобы дать перспективу управления цепочками.

- Пользователь - это восприятие;
- Производительность, требующая бизнес-процесса;
- Обработка информации по цепочке.

В дополнение к мониторингу информационной системы мониторинг цепи включает в себя следующие функции.

Функция	Определение
Информация Поток Мониторинг E2E	На этом уровне признается мониторинг информационных потоков. При измерении информационного потока EXNUMEXE ни одна услуга ИКТ не измеряется, а бизнес-процессы
Мониторинг бизнес-процессов E2E	Эта функция измеряет бизнес-процесс от начала до конца на основе связанных информационных систем
Монитор конечного пользователя E2E	На этом уровне оценивается предоставление услуг с точки зрения пользователя. Изображение предоставления услуг получается с точки зрения конечного пользователя

3.4 Интерфейс Платформы

3.4.1 Описание интерфейса платформы "Marlin"

Интерфейс Платформы поддерживается большинством браузеров. Адаптивная верстка позволяет использовать различные устройства для работы Платформы без потери информации. Язык интерфейса – русский.

1. Раздел "Обзор": отображает состояние используемых компонентов в системе для сотрудника с ролью "Администратор"
    - Вкладка "Компоненты и ПО"
2. Раздел "Панель Развертываний": отображает все существующие развёртывания, а также связь с поставками сервиса, репозиторием кода, сборками и,или

автосборками сервиса приложения.
    - Номер
    - Среда
    - Среда исполнения
    - Протокол
    - Валидность
    - Шаблон
    - Сервис
    - Приложение
3. Раздел "Приложения": отображает все существующие приложения в Платформе, в рамках которых создаются сервисы приложений.
    - Название
    - Бизнес-владелец
    - Дата создания
4. Раздел "Сервисы": включает в себя информацию о приложении. поставки приложения, зависимости приложения, процессы, сборки, автосборки, шаблоны, развертывания, настройки и интеграции, которые необходимы для работы с приложениями на портале Marlin
    - вкладка "Сервисы Приложений": отображает все созданные Сервисы приложений
        - Название
        - Родительское приложение
        - Дата создания
        - Тип
        - Accelerator Pack
    - вкладка "Сервисы PaaS": отображает PaaS сервисы
        - Название
        - Системные компоненты
        - Дата создания
        - Runtime
5. Раздел "Среды": отображает используемые на платформе среды. Каждая Среда имеет параметры и набор Систем, установленных в ней. Сотруднику с ролью "Администратор" имеет возможность добавить новую среду.
    - DEV

- TEST
- STAGE
- PROD

6. Раздел "Аудит": отображает временную последовательность действий совершаемых пользователями на портале Marlin.
   - Дата события
   - Название
   - Инициатор
   - Email
   - Тип объекта
   - Объект
7. Раздел "Настройки":
   - Настройки СК - Отображают ссылки на Confluence и GitLab
   - Настройки ПУ -Отображают среды исполнения
8. Раздел "Справочники":
   - вкладка "Системные компоненты": отображает СК, применяемые пользователями в работе на портале
   - вкладка "Репозитории":
     - Docker
     - Maven
     - APK
     - Npm
   - вкладка "Типы PaaS": отображает предоставляемые услуги в зависимости от типа. Также, при создании СК, в карточке указывается "Тип PaaS" (при условии, если новая СК является PaaS-услугой)
     - Среда исполнения
     - Тип платформенного сервиса
     - Управление доступом
   - вкладка "Типы сервисов"
     - External
     - Kafka
     - Front
     - Backend

- вкладка "Accelerator Packs": отображает перечень стеков со своими pipeline в Jenkins -Accelerator pack for kafka -Angular -JAVA11 -NodeJS -Docker -DOTNET3 -PHP
- вкладка "Типы событий": отображает связь между событием и его возможной автоматической реализацией.

3.4.2 Ограничения в интерфейсе платформы "Marlin"

Пользователю с ролью "Администратор" доступны все действия в Платформе:

1. Создание приложений.
2. Удаление приложений.
3. Создание PaaS сервисов.
4. Добавление новых Сред.
5. Изменение настроек Сервисов приложений.
6. Сохранение настроек Сервисов приложений.
7. Редактирование Настроек портала управления.
8. Редактирование настроек Системных компонент.
9. Добавление Репозиториев.
10. Удаление Репозиториев.
11. Добавление типов PaaS.
12. Редактирование типов PaaS.
13. Добавление Accelerator Pack.
14. Редактирование Accelerator Pack.
15. Создание, редактирование, удаление карточки сервиса.
16. Просмотр, редактирование, добавление исходного кода в репозиторий GitLab.
17. Создание, редактирование, удаление сборки.
18. Создание, редактирование, удаление автосборки.
19. Создание, редактирование, удаление развертывания.
20. Создание, редактирование, удаление авторазвертывания.
21. Создание, редактирование, удаление карточки поставки.
22. Подписка/ удаление подписки на платформенные сервисы.
23. Подписка/ удаление подписки на прикладные сервисы.
24. Перезапуск интеграции с Jenkins.

Пользователю с ролью "Разработчик"/ "Сотрудник эксплуатации" доступны следующие действия в Платформе:

1. Создание, редактирование, удаление карточки сервиса.
2. Просмотр, редактирование, добавление исходного кода в репозиторий GitLab.
3. Создание, редактирование, удаление сборки.
4. Создание, редактирование, удаление автосборки.
5. Создание, редактирование, удаление развертывания.
6. Создание, редактирование, удаление авторазвертывания.
7. Создание, редактирование, удаление карточки поставки.
8. Подписка/ удаление подписки на платформенные сервисы.
9. Подписка/ удаление подписки на прикладные сервисы.
10. Перезапуск интеграции с Jenkins.

Пользователю с ролью "Базовый" доступен просмотр всех вкладок, приложений и сервис без возможности редактирования чего-либо.

4. Обеспечение защиты данных

4.1 Требования к шифрованию

1. На WAF терминируется шифрование с использованием сертификата подписанного доверенным УЦ. Соединение с данным типом шифрования доступно как внешним пользователям, так и внутренним через балансировщик.
2. На OpenShift Master Node терминируется шифрованием с использованием сертификата, подписанного сертификатом IntermediateCA и RootCA, который является доверенным только для тех операционных систем, на которых установлен сертификат RootCA компании.
3. Для каждого Ingress (в k8s) или Router (в OpenShift) во время разворачивания сервиса Jenkins с использованием OpenSSL генерирует новый сертификат, подписанный IntermediateCA.
4. Приватная часть сертификата IntermediateCA доступна платформе.
5. Приватная часть сертификата RootCA доступна только инженерам компании.

4.2 Требования к балансировки трафика

4.2.1 Схема балансировки трафика из Интернета на UI/REST

1. Строго внешний трафик (не межсервисный с gRPC/REST)
2. SSL терминируется на кластере WAF
3. Для каждого внешнего адреса на продуктивных средах (Stage, Prod, HotFix) вручную настраивается маппинг внешнего домена на внутренний фиксированный домен. Именно этот внешний домен и прописывается в шаблоне развертывания .
4. WAF балансируется по наименьшему числу открытых сессий.
5. WAF сохраняет X-FORWARDED-FOR.
6. Внутренние пользователи ходят в этот сервис через внешний WAF.

4.2.2 Схема балансировки трафика из внутренней сети на UI/REST

1. Строго пользовательский/legacy трафик (не межсервисный с gRPC/REST).
2. SSL терминируется на route/ingress.
3. DNS-балансировка нагрузки между OpenShift Infra.

4.2.3 Схема балансировки трафика gRP/REST между разными кластерами

1. Межсервисный трафик.
2. SSL терминируется на route/ingress.
3. Добавить сертификаты для промежуточных центров сертификации каждой из сред в базовый образ для сервисов

4.3 Требования к сетевой архитектуре

1. Сервера обрабатывающие подключения из внешних сетей отделены от основного сегмента системы.
2. Сервера БД размещены в выделенном сегменте БД.
3. Сервера отвечающие за формирование бизнес-логики системы, сервера хранящие информацию конфиденциального характера размещены во внутреннем сегменте LAN.
4. Все взаимодействия компонентов Платформы с системами внутри сети осуществляться через интеграционную шину, либо брокер сообщений (kafka).
5. При аутентификации систем под служебной(технологической) учетной записи для интеграционного взаимодействия вида «система-система» выполняется требование парольного Стандарта о парольной защите компании.

6. В интерфейсах (API) взаимодействия с внешними системами должны реализованы механизмы аутентификации и авторизации.
7. В случае интеграции компонентов Платформы с другими системами через интеграционную шину выполняются следующие требования:
    - Доступ к шинам ограничен посредством межсетевого экранирования;
    - Ограничен доступ систем, обращающихся к шинам, только системами, обладающими логином и паролем;
    - Использование систем, проксирующих запросы других систем к шинам, запрещено

4.4 Требования к разграничению прав доступа

В процессе функционирования Платформы должно осуществляться управление доступом (правами доступа) субъектов доступа, подразумевающее разграничение доступа субъектов к объектам доступа на основе совокупности установленных в Платформе правил (разграничения доступа), а также контроль соблюдения этих правил.

1. В Платформе предусмотрены 4 роли пользователей, каждая роль имеет одну соответствующую ей группу :

    ```
    * базовый доступ (marlin_basic) - доступ только на чтение ко всему, доступы остальным ролям выдаются как добавление прав к базовому доступу.
    ```

    пользователи, доступ которых ограничивается средой и вхождением в команду приложения:

    ```
    * разработчик (marlin_devel_user);
    * сотрудник эксплуатации (marlin_exploitation_user);
    * администратор Платформы (marlin_admin_user) - управляет не зависящими от среды сущностями ПУ.
    ```

2. Ограничения действуют по средам и командам.

Подробнее про возможности и ограничения ролевой модели смотрите в пункте 3.4.2.

4.5 Зонирование

Платформа реализуется с учетом работы в трех зонах: в зоне разработки, в зоне тестирования и в зоне постоянной эксплуатации.

Для разграничения областей обработки информации и в целях безопасности и изоляции влияния процессов разработки/тестирования и эксплуатации друг на друга в Платформе выделяются следующие зоны:

- зона разработки (ЗР) предназначена для проектирования, разработки и отладки разработчиком обновлений компонентов Ядра, новых или модернизируемых сервисные подсистемы, приложений;
- зона тестирования (ЗТ) предназначена для осуществления тестирования доработанных компонентов Ядра, вновь созданных или модернизированных сервисные подсистемы, приложений;
- зона постоянной эксплуатации (ЗПЭ) предназначена для осуществления постоянной эксплуатации компонентов Ядра, сервисные подсистемы и приложений на реальных данных.

В ЗР доступны все компоненты Ядра, сервисные подсистемы и приложения, а также средства управления требованиями/инцидентами и база знаний, предоставляемые в составе облачной Платформы. В ЗР производится разработка и отладка приложений, включая анализ качества исходного кода, анализ приложений на соответствие требованиям информационной безопасности.

В ЗТ выделяется три изолированных сегмента: сборки, тестирования на тестовых данных и тестирования на реальных данных. В сегменте сборки размещаются средства по управлению репозиториями и средства непрерывной интеграции в отдельных специализированных контейнерах, доступ к которым для пользователей и эксплуатационного персонала в штатном режиме запрещается. В сегментах тестирования размещаются остальные компоненты Ядра, СПС, приложения и сервисы. Для сегмента тестирования на реальных данных выделяются узлы с ПО исполнения контейнеров в выделенном сетевом сегменте.

4.6 Авторизация и аутентификация

В процессе функционирования Платформы должны осуществляться:

- присвоение субъектам (пользователи, процессы и т. д.) и объектам (устройства, объекты файловой системы, запускаемые и исполняемые модули, объекты СУБД и

т. п.) доступа уникальных признаков (идентификаторов);
- сравнение предъявляемых идентификаторов с присвоенными им ранее;
- проверка подлинности предъявленных идентификаторов (аутентификация).

Пользовательская и межсервисная авторизация может быть реализована с помощью PaaS Keycloak.

Далее аутентификация в приложениях и подсистемах осуществляется с помощью токенов, которые выдает сервер аутентификации. Процесс выдачи и обмена токенами скрыт от пользователя. Таким образом, войдя в Платформу пользователь автоматически имеет доступ ко всем ресурсам, для которых настроено SSO и к которым разрешен доступ, при этом на сервере не создается сессия для пользователя. Проверка подлинности токена и идентификация происходит при каждом обращении к сервису.

В качестве стандарта взаимодействия используется стандарт OpenID Connect.

4.6.1 Обработка входящих запросов

Пользовательский запрос

Обработка пользовательского запроса происходит по алгоритмам протокола OpenID Connect. Прикладной сервис обрабатывает пользовательские запросы следующим образом:

1. Неаутентифицированные пользователи перенаправляются на страницу Keycloak для аутентификации (ввод логина и пароля);
2. Для пользователей прошедших аутентификацию в Keycloak прикладной сервис получает access token, refresh token и id token;
3. Для дальнейших запросов пользователь использует access token;
4. Прикладной сервис должен валидирует access token при каждом запросе (проверяется срок действия и подпись);
5. При истечении срока действия access token, он может быть обновлен с помощью refresh token или пользователь становится неаутентифицированным;
6. Внутри access token и id token передается информация о пользователе и его ролях;
7. Прикладной сервис использует информацию о ролях для авторизации действий пользователя в сервисе.

Межсервисный запрос

Прикладной сервис обрабатывает запрос от сервисов следующим образом:

1. Неаутентифицированный запрос возвращается с ошибкой;
2. Для запросов с access token прикладной сервис валидирует access token при каждом запросе. Проверяется срок действия, подпись и наличие ACCESS роли клиента прикладного сервиса, принимающего запрос;
3. При истечении срока действия access token или отсутствия ACCESS роли, запрос возвращается с ошибкой.

4.6.2 Выполнение исходящих запросов

Для отправки запроса на авторизацию в связном сервисе:

1. Используя ClientId и ClientSecret прикладной сервис получает access token со списком ACCESS ролей прикладных сервисов, к которым может обращаться данный сервис;
2. Все исходящие запросы прикладной сервис сопровождает полученным access token;
3. При истечении срока действия, прикладной сервис перевыпускает access token.

4.7 Хранение секретной информации

Vault - система, которая должна хранить часть данных сервиса (секреты). является компонентом платформы Marlin.

Vault контролирует доступ к секретам и ключам шифрования, выполняя аутентификацию с помощью сервиса PaaS Keycloak. Секреты хранятся в значение key-value. Доступ к хранилищу осуществляется исключительно через API. Все данные хранятся в зашифрованном контейнере. Получение самого контейнера не раскрывает данные.

В платформе Marlin определены следующие типы секретов:

1. Доступ приложений из OpenShift
    - MinIO - Доступ к файлам
    - PostgreSQL - доступ к БД

- Keycloak - получение access token
- Kafka - доступ к событиям, очередям
- Секретные переменные окружения
- Nexus - получение образов приложений

2. Jenkins
   - Пароль дженкинса для проливки
     - MinIO - создание бакетов
     - PostgreSQL - Создание пользователей и схем, миграции
     - Keycloak - Создание клиентов, сервис аккаунтов, ролей, маппинга.
     - Kafka - СОздание продюсеров и консюмеров.
   - Генерация сертификатов (PrivateKey)
   - Пароли от компонентов платформы
     - OpenShift - доставка приложения
     - GitLab - Получение исходников приложений для сборки и получение кода пайплайнов
     - Nexus - получение и публикация образов (пока без пароля?)
     - ПУ - отправка callback
3. ПУ
   - Confluence - публикация документации
   - GitLab - управление репозиториями
   - Jenkins - запуск пайплайнов
   - Keycloak - авторизация пользователей
4. Доступ пользователей к PaaS (При использовании статических секретов HashiCorp не требуется)
   - MinIO
   - PostgreSQL

Процесс доставки секретов до приложения состоит из следующих этапов:

1. Секрет через ПУ Marlin размещается в Vault
2. В namespace приложения создается сущность ExternalSecret, описывающая способ получения и место доставки секрета. Разграничение политик доступа к секретам происходит на основе namespace сервиса. Один namespace – одна видимость секретов.
3. в сущности ExternalSecret создаётся Secret

4. Содержимое Secret монтируется в Pod как переменная окружения или файлult трVault строго контролирует доступ к секретам и ключам шифрования, выполняя аутентификацию в надежных источниках идентификации, таких как Active Directory, LDAP, Kubernetes, CloudFoundry и др.

4.7.1 Структура хранилища секретов

Используем одну инсталляцию Hashicorp Vault на среду.

Секреты прикладных сервисов:
secret_engine_path/apps/***stage***/***namespace***/***secret_name***

Секреты используемые в Jenkins: ***secret_engine_path***/jenkins/***stage***/***secret_name***

Где:

secret_engine_path - это путь до хранилища секретов

stage - это название среды

namespace - это namespace/project прикладного сервиса в OpenShift

secret_name - это название секрета

4.7.2 Управление доступом

Политика для jenkins:

- Чтение для *secret_engine_path*/apps/*stage*/*
- Создание, обновление для *secret_engine_path*/jenkins/*stage*/*

Политика для прикладного сервиса (выдается на один прикладной сервис):

- Чтение для *secret_engine_path*/apps/*stage*/*namespace*/*

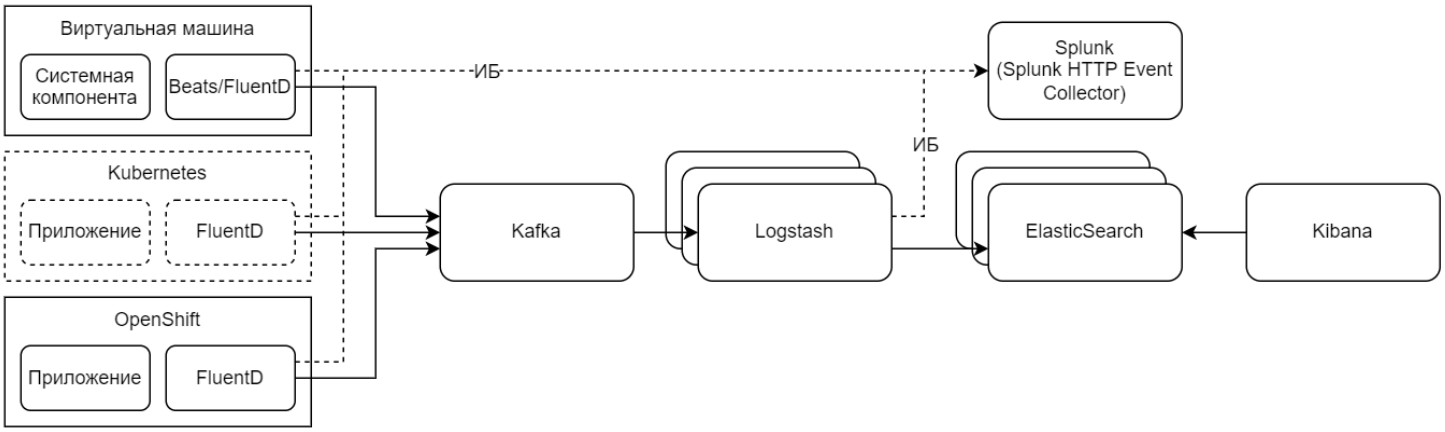
4.7.3 Процессы

1. Создание секрета
    - Сохранение секрета в Vault
    - Сохранение имени-секрета в ПУ
    - Передача необходимых имен в OpenShift для извлечения из хранилища

2. Обновление, ротация
    - Персистентные секреты
    - Секреты с ограниченным сроком жизни
3. Чтение секретов платформой
4. Чтение секретов прикладными сервисами
5. Выдача политик чтения/записи и привязка к токенам доступа
    - Авторизация платформы
    - Авторизация прикладных приложений

4.8 Требования к логированию

4.8.1 Компоненты Платформы Marlin



Название	Назначение
Beats	Сборщик логов для виртуальных машин не относящихся к средам исполнения.
FluentD	Сборщик логов интегрируемый в среды исполнения openshift и kubernetes. Кроме сбора логов так же занимается обогащением их информацией доступной из среды исполнения: идентификатор контейнера, сервиса, пода, неймспейса, итд.

Название	Назначение
Kafka	Брокер сообщений используемый для балансировки нагрузки потока логов.
Logstash	Утилита для сбора, обогащения, фильтрации и маршутизации потоков логов. Планируется использовать его для отделения событий ИБ от общего потока сообщений логов для перенаправления в отдельный кластер ElasticSearch.
ElasticSearch	Утилита для сбора, обогащения, фильтрации и маршутизации потоков логов. Планируется использовать его для отделения событий ИБ от общего потока сообщений логов для перенаправления в отдельный кластер ElasticSearch.
Kibana	Веб-интерфейс для просмотра логов, визуализации и агрегации информации из логов.
Splunk	Существующая система используемая в ВСК для хранения, обработки и доступа к событиям ИБ.

4.8.2 Принцип работы

После публикации сообщения прикладным сервисом в поток STDERR или STDOUT оно будет перехвачено компонентом FluentD. В компоненте FluentD сообщение будет обогащено информацией среды исполнения в том числе будет добавлен: идентификатор контейнера, идентификатор сервиса, идентификатор пода, идентификатор namespace. Это позволит точно определить источник сообщения.

Компонент FluentD отправляет все сообщения в брокер сообщений Kafka. Kafka предназначена для балансировки/сглаживания нагрузки в пиковые часы работы платформы. Таким образом предотвращается потеря сообщений из-за падения производительности или возможного отказа системы хранения логов вследствие перегрузки.

Компонент Logstash забирает из Kafka сообщения и перенаправляет их в хранилище - ElasticSearch. Logstash так же занимается маршрутизацией сообщений на основе

содержащейся в них информации, например, поля level. Это может использоваться для разделения потоков обычных событий и событий ИБ, для которых предназначена отдельная система Splunk. Так же Logstash маршрутизирует сообщения по различным индексам в зависимости от значений сообщения.

В ElasticSearch хранятся все события согласно жизненному циклу индекса. Жизненный цикл подразумевает ротацию старых сообщений для предотвращения переполнения диска и оптимизацию работы хранилища ElasticSearch во время операций записи и поиска.

Разработчики и администраторы прикладных сервисов платформы могут получить доступ к логам через компонент Kibana.

Splunk может получать все логи из коллекторов FluentD или Beats. Либо из logstash после балансировки.

Прикладные сервисы, разворачиваемые на Платформе Marlin, не зависимо от среды исполнения (Openshift, Kubernetes) выполняют следующие требования:

- Прикладной сервис осуществляет вывод логов в потоки STDOUT или STDERR. Вывод логов в файлы запрещен.
- Каждое сообщение в логе прикладного сервиса в формате JSON.
- Логи не содержат информацию, нарушающую безопасность прикладных сервисов или Платформы. Вывод сообщений, включающий подобную информацию, маскируется в местах критичных участков сообщения. Обязательно выполняется маскирование паролей, токенов, пин-кодов и т. п.

4.8.3 Принципы формирования логов по уровням

Уровень	Тип сообщений
TRACE	-
DEBUG	Отладочная информация.
INFO	Сообщения, носящие информационный характер: - системные сообщения от Microsoft.AspNetCore (пишутся автоматически) / Springboot

Уровень	Тип сообщений
	- сообщения вызова grpc сервиса, содержащие информацию по запросу: метод и данные (Request/Response) - сообщения этапов обработки запросов
WARN	Сообщения бизнес-логики, информирующие о неверных данных: - в запросе переданы логически некорректные данные (т.е. ошибки валидации) - в БД или смежных системах данные не соответствуют ожидаемым (например, не найдена заявка, которую необходимо утвердить и т.д.)
ERROR	Неожиданные ошибки, которые не обрабатываются согласно бизнес-логике: - исключения (exceptions), сгенерированные не нашим кодом и логическая обработка которых не предусмотрена.
FATAL	Сервис упал с непредвиденным исключением.

4.9 Требования к регистрации событий

Определение событий для вывода: необходимо выводить в разделе "Аудит" все события, совершаемые пользователем ("Автоматическое выполнение"=Нет), в соответствии с событиями, указанными в разделе "События безопасности"

Просмотр раздела "Аудит":

- Пользователь с ролью "Администратор": отображаются все записи в разделе
- Пользователь с ролью, отличной от "Администратор": отображаются только записи о приложениях, в команду которых длбавлен пользователь.

4.9.1 События безопасности:

Событие	Результат	Автоматическое выполнение
Авторизация		

Событие	Результат	Автоматическое выполнение
Вход/выход в систему (из системы)	осуществление входа/выхода	нет
Ошибка авторизации	Отображение ошибки	да
Просмотр панели развертываний (на всех средах)	отображение страницы "Активные развертывания"	нет
Среда		
Просмотр параметров среды	отображение карточки среды	нет
Добавление новой среды	доступ к форме создания	нет
Изменение порядка сред	отображение измененной последовательности сред	нет
Редактирование параметров среды	доступ к форме редактирования	нет
Настройка интеграций при создании прикладного сервиса	запуск интеграций со средой для нового сервиса	да
Настройка интеграций при создании севриса-PaaS	запуск интеграций со средой для нового сервиса	да
Перезапуск неуспешной интеграции для	повторный запуск интеграции для сервиса	нет
Панель развертываний на среде	отображение страницы "Активные развертывания"	нет

Событие	Результат	Автоматическое выполнение
Поиск по названию развертывания на среде	фильтрация результатов в зависимости от условий поиска	нет
Скрытие среды	среда не отображается в перечне по умолчанию	нет
Системная компонента		
Просмотр параметров СК	отображение карточки СК	нет
Редактирование параметров СК	доступ к форме редактирования	нет
Изменение характеристики СК	изменение иконки характеристики на странице с перечнем СК	нет
Просмотр настроек СК	отображение параметров СК	нет
Просмотр настроек ПУ	отображение настроек ПУ	нет
Редактирование настроек ПУ	отображение формы редактирования	нет
Тип PaaS		
Просмотр параметров типа PaaS	отображение карточки типа PaaS	нет
Добавление нового типа	доступ к форме создания	нет
Редактирование параметров типа PaaS	доступ к форме редактирования	нет

Событие	Результат	Автоматическое выполнение
Экземпляр системной компоненты		
Просмотр параметров ЭСК	отображение карточки ЭСК	нет
Изменение параметров ЭСК	доступ к форме редактирования	нет
Изменение статуса ЭСК	доступ к форме редактирования	нет
Добавление ЭСК	создание ЭСК при добавлении новой среды	да
Добавление ЭСК	добавление ЭСК при копировании существующей	нет
Репозиторий		
Просмотр параметров репозитория	отображение карточки репозитория	нет
Создание репозитория	доступ к форме создания	нет
Редактирование параметров репозитория	доступ к форме редактирования	нет
Удаление репозитория	удаление репозитория из списка	нет
Запуск синхронизации репозитория (редактирование справочника)	запуск пайплайна по синхронизации для репозитория Nexus после созранения изменений при	да

Событие	Результат	Автоматическое выполнение
	редактировании белого списка	
Запуск синхронизации репозитория в справочнике (кнопка "Синхронизировать")	запуск пайплайна по синхронизации для репозитория Nexus после нажатия на кнопку пользователем	нет
Запуск синхронизации репозиториев в карточке СК Nexus (кнопка "Синхронизировать")	запуск пайплайна по синхронизации для всех репозиториев Nexus после нажатия на кнопку пользователем	нет
Отображение протокола синхронизации	Вывод протокола с результатом выполнения синхронизации в Nexus	да
Сервис PaaS		
Просмотр параметров сервиса PaaS	отображение карточки сервиса PaaS	нет
Добавление сервиса PaaS	доступ к форме создания	нет
Редактирование параметров сервиса PaaS	доступ к форме редактирования	нет
Добавление новой ЭСК для сервиса PaaS	доступ к форме редактирования	нет
Запуск интеграции после добавления ЭСК для сервиса	автоматический запуск интеграции со средой при	да

Событие	Результат	Автоматическое выполнение
	добавлении ЭСК	
Поиск по названию в перечне сервисов-PaaS	вывод результатов поиска	нет
Фильтрация по СК в перечне сервисов-PaaS	вывод результатов поиска	нет
Развертывание PaaS		
Добавление развертывания PaaS-сервиса	доступ к форме создания	нет
Добавление развертывания Runtime-сервиса	доступ к форме создания	нет
Уведомление о неуспешном развертывании Runtime-сервиса	отображается иконка с подсказкой при ошибке развертывания Runtime-сервиса	да
Удаление развертывания PaaS	удаление выделенных ресурсов	нет
Отображение протокола создания развертывания	Вывод протокола с результатом выполнения развертывания в Jenkins	да
Отображение протокола удаления развертывания	Вывод протокола с результатом удаления развертывания в Jenkins	да
Вывод статуса в зависимости от успешности развертывания	отображение статуса на блоке с развертыванием в	да

Событие	Результат	Автоматическое выполнение
	зависимости от статуса выполнения пайплайна Jenkins	
Ограничение по добавлению развертывания на среде для сервиса	недоступность добавления развертывания PaaS на среде, для которой не настроена интеграция	да
Подписка на PaaS		
Добавление подписки на PaaS-сервис	Создание подписки в разделе Зависимости	нет
Добавление подписки на Runtime-сервис	автоматическое добавление подписки после настройки интеграций прикладного сервиса	да
Удаление подписки на PaaS	Подписка удалена из раздела Зависимости	нет
Просмотр подписчиков сервиса PaaS	Отображение списка с подписчиками PaaS-сервиса	нет
Добавление подписчика в раздел "Подписчики" сервиса	автоматическое добавление сервиса, добавившего подписку, в раздел "Подписчики" сервиса-PaaS	да
Приложение		
Просмотр параметров приложения	отображение карточки приложения	нет
Создание приложения	доступ к форме создания	нет

Событие	Результат	Автоматическое выполнение
Изменение параметров приложения	доступ к форме редактирования	нет
поиск по названию в перечне приложений	вывод результатов поиска	нет
Отображение ссылок на репозитории кода, API и процессов для созданного приложения	автоматическое отображение ссылок на репозитории после установления интеграций для нового приложения	да
Запуск интеграции с Gitlab для приложения	запуск интеграции для нового приложения	да
Отображение статуса выполнения интеграции с Gitlab для приложения	вывод статуса интеграции для приложения	да
Перезапуск неуспешной интеграции для приложения	повторный запуск интеграции для приложения	нет
Команда приложения		
Добавление участника в команду приложения	доступ к форме изменения команды	нет
Добавление создателя приложения в команду	автоматическое добавление пользователя, создавшего приложение, в команду	да
Удаление участника из команды приложения	доступ к форме изменения команды	нет
Сервис		

Событие	Результат	Автоматическое выполнение
Просмотр параметров сервиса	отображение карточки сервиса	нет
Создание сервиса	доступ к форме создания	нет
Редактирование параметров сервиса	доступ к форме редактирования	нет
Поиск по названию в перечне сервисов	вывод результатов поиска	нет
Отображение ссылок на репозитории кода, API и процессов для созданного сервиса	автоматическое отображение ссылок на репозитории после установления интеграций для нового сервиса	да
Запуск интеграции с Gitlab для сервиса	запуск интеграций для нового сервиса	да
Отображение статусов выполнения интеграции с Gitlab для сервиса	вывод статусов интеграции для сервиса	да
Перезапуск неуспешной интеграции для сервиса	повторный запуск интеграции для сервиса	нет
Сборка сервиса		
Просмотр параметров сборки	отображение карточки сборки	нет
Создание сборки	доступ к форме создания	нет

Событие	Результат	Автоматическое выполнение
Редактирование параметров сборки	доступ к форме редактирования	нет
Запуск сборки после создания	автоматически запускается пайплайн сборки после ее создания	да
Перезапуск сборки	Изменение статуса выбранной сборки и запуск пайплайна	нет
Добавление развертывания сервиса в сборке	Создание развертывания	нет
Вывод статуса в зависимости от успешности развертывания в сборке	отображение статуса на блоке с развертыванием в зависимости от статуса выполнения пайплайна Jenkins	да
Отображение протокола запуска сборки	Вывод протокола с результатом выполнения сборки в Jenkins	да
Вывод статуса в зависимости от успешности сборки	отображение статуса на блоке со сборкой в зависимости от статуса выполнения пайплайна Jenkins	да
Поставка сервиса		
Просмотр параметров поставки	отображение карточки поставки	нет
Создание поставки	доступ к форме создания	нет

Событие	Результат	Автоматическое выполнение
Редактирование поставки	доступ к форме редактирования	нет
Изменение статуса поставки	Изменение статуса поставки	да
Добавление развертывания сервиса в поставке	Создание развертывания	нет
Перезапуск развертывания в поставке	запуск пайплайна дял развертывания сервиса после нажатия пользователем на кнопку "Перезапуск"	нет
Удаление развертывания в поставке	удаление развертывания из списка	нет
Перезапуск сборки в поставке	Изменение статуса выбранной сборки и запуск пайплайна	нет
Отображение версии сервиса-подписки в Зависимостях поставки	автоматическое отображение поставки сервиса-подписки в карточке поставки сервиса	да
Вывод статуса в зависимости от успешности развертывания в поставке	отображение статуса на блоке с развертыванием в зависимости от статуса выполнения пайплайна Jenkins	да
Скрытие поставки	перевод поставки в статус "Закрыта в статусе..."	нет
Тип сервиса		

Событие	Результат	Автоматическое выполнение
просмотр параметров типа сервиса	отображение параметров типа сервиса	нет
Поиск по названию в перечне типов сервиса	вывод результатов поиска	нет
Accelerator pack		
просмотр параметров Accelerator Pack	отображение карточки Accelerator Pack	нет
Создание Accelerator Pack	доступ к форме создания	нет
Изменение параметров Accelerator Pack	доступ к форме редактирования	нет
Поиск по названию в перечне Accelerator Pack	вывод результатов поиска	нет
Развертывание сервиса		
Создание развертывания сервиса	Отображается форма создания развертывания	нет
Остановка развертывания сервиса	запуск пайплайна stop-start	нет
Перезапуск развертывания сервиса	запуск пайплайна deploy	нет
Удаление развертывания	запуск пайплайна undeploy	нет
Вывод статуса в зависимости от успешности	отображение статуса на блоке с развертыванием в	да

Событие	Результат	Автоматическое выполнение
развертывания сервиса	зависимости от статуса выполнения пайплайна Jenkins	
Отображение протокола создания развертывания сервиса	Вывод протокола с результатом выполнения развертывания в Jenkins	да
Отображение протокола удаления развертывания сервиса	Вывод протокола с результатом удаления развертывания в Jenkins	да
Подписки на прикладной сервис		
Добавление подписки на прикладной сервис	отображение списка доступных для подписки сервисов	нет
Добавление подписчика в раздел "Подписчики" сервиса	автоматическое добавление сервиса, добавившего подписку, в раздел "Подписчики" прикладного сервиса	да
Удаление подписки на прикладной сервис	удаление подписки из перечня зависимостей сервиса	нет
Добавление подписки на сервис Kafka	форма для выбора типа подписки на сервис	нет
Шаблон развертывания		
Автоматическое создание шаблона развертывания для	автоматическое создание шаблона развертывания на	да

Событие	Результат	Автоматическое выполнение
сервиса	dev	
Добавление шаблона развертывания	отображение формы создания	нет
Редактирование шаблона развертывания	отображение формы редактирования	нет
Сохранение изменений в шаблоне развертывания	отображение сохраненного шаблона развертывания в перечне шаблонов	нет
Добавление развертывания Runtime-сервиса в шаблон развертывания прикладного сервиса	автоматическое добавление успешного развертывания Runtime-сервиса в параметры шаблона развертывания прикладного сервиса	да
Генерация адресов в шаблоне развертывания	автоматическая генерация адресов в шаблона развертывания при успешной подписке на среду исполнения	да
Поиск шаблона развертывания в перечне	отображение шаблона развертывания в соответствии с условиями поиска	нет
Просмотр параметров шаблона развертывания	отображаются параметры шаблона развертывания	нет
Отображение неактивных шаблонов в перечне	отображение в перечне неактивных шаблона развертывания	нет
Профиль/права доступа		

Событие	Результат	Автоматическое выполнение
Создание пользовательской учетной записи для доступа к развертываниям PaaS (PostgreSQL/MinIO)	запуск Jjob для создания пользовательской учетной записи после нажатия пользователем на кнопку для запроса доступа к развертыванию PaaS	нет
Удаление учетной записи для доступа к развертываниям PaaS (PostgreSQL/MinIO)	запуск удаления учетной записи после нажатия пользователем на кнопку "Удалить"	нет
Удаление учетной записи для доступа к развертываниям PaaS (PostgreSQL/MinIO) при удалении развертывания PaaS	автоматическое удаление учетной записи после успешного удаления развертывания PaaS	да
Просмотр профиля пользователя	отображение страницы "Профиль"	да
Изменение прав доступа при редактировании настроек среды	автоматическое изменение прав пользователя на управление объектами в ПУ на средах	да
Дополнительно логировать		
IP-адрес пользователя, выполнившего действие	-	-

Событие	Результат	Автоматическое выполнение
Идентификатор сессии, в рамках которой произошли изменения	-	-

4.10 Требования к обеспечению целостности

SonarQube - это инструмент автоматической проверки кода для обнаружения ошибок и уязвимостей в вашем коде. Он интегрируется с существующим рабочим процессом, чтобы обеспечить непрерывную проверку кода в ваших ветвях проекта и в запросах на интеграцию изменений из одной ветки в другую.

Проверка SonarQube осуществляется на выявление заданных параметров исходного кода на качество.

В интерфейсе SonarQube реализована возможность добавления правил проверки (Quality profile) и признаков, по которым оценивается качество кода (Quality gate). В SonarQube предусмотрены стандартные наборы правил и признаки для проверки кода (с пометкой "default"), которые применяются по умолчанию в случае, если не было добавлено других. Также предусмотрено создание наборов правил для любого языка (включая JavaScript и XML), перечень языков должен определяться автоматически.

Особенности реализации:

- управление настройками параметров проверок осуществляется при сборке сервиса посредством ручного выбора проверок.
- обеспечивается с помощью стандартных политик;
- успешное прохождение проверки необходимо для перевода на следующую среду.

Результат проверки качества своего проекта будет доступен по ссылке, предоставленной в конце анализа.

Параметры анализа могут быть настроены в нескольких местах:

- Глобальные свойства, определенные в пользовательском интерфейсе, применяются ко всем проектам (в верхней панели выберите Администрирование-

Конфигурация-Общие параметры).

- Свойства проекта, определенные в файле конфигурации анализа проекта или в файле конфигурации анализатора, имеют приоритет над параметрами, определенными в пользовательском интерфейсе.
- Параметры анализа проекта, определенные в файле конфигурации анализа проекта или файле конфигурации анализа, имеют приоритет над параметрами, определенными в пользовательском интерфейсе.
- Параметры анализа/ командной строки, определенные при запуске анализа переопределяют параметры анализа проекта.

4.10.1 Обязательные параметры:

1. Сервер.
2. Конфигурация проекта.

4.10.2 Необязательные параметры:

1. Идентичность проекта.
2. Аутентификация.
3. Веб сервисы.
4. Конфигурация проекта.
5. Дублирование.
6. Журнал анализа.

5. Аппаратные и программные требования предъявляемые к ПО

5.1 Перечень программного обеспечения

Резервное копирование

1. По данным:
    - Бекап данных
    - Бекап конфигурации
2. По уровню:
    - Инфраструктурные бекапы (бекапы системных компонент Платформы)

- Пользовательские бекапы (бекапы сервисов)

5.1.1 Требования к Порталу Управления

Назначение	Ядра CPU, шт	Память RAM, GB	SSD	Резервное копирование
Мониторинг (HAProxy, Prometheus, Alertmanager Grafana)	4	8	sda: 25 GB sdb: 25 GB sdc: 200 GB	ежедневно
Мониторинг (HAProxy, Prometheus, Alertmanager Grafana)	4	8	sda: 25 GB sdb: 25 GB sdc: 200 GB	ежедневно
Artifactory	4	8	sda: 50 GB sdb: 1 TB	ежедневно
Artifactory	4	8	sda: 50 GB sdb: 1 TB	ежедневно
Artifactory	4	8	sda: 50 GB sdb: 1 TB	ежедневно
X-Ray	4	8	300 GB	ежедневно
X-Ray	4	8	300 GB	ежедневно
HAProxy for PostgreSQL 13 Platform	2	2	56 GB	ежедневно
HAProxy for PostgreSQL 13 Platform	2	2	56 GB	ежедневно
PostgreSQL 13 Platform	6	24	500 GB	ежедневно

Назначение	Ядра CPU, шт	Память RAM, GB	SSD	Резервное копирование
PostgreSQL 13 Platform	6	24	500 GB	ежедневно
PostgreSQL 13 Platform	6	24	500 GB	ежедневно
Nexus	4	8	sda: 56.00 GB sdb: 640.00 GB	ежедневно
Nexus	4	8	• / - 20 Гб • /home – 10 Гб • /var – 20 Гб lvm • /tmp – 1 Гб • /data - 650 Гб	ежедневно
Portal	4	8	sda: 56.00 GB sdb: 40.00 GB	ежедневно
Portal	4	8	sda: 56.00 GB sdb: 40.00 GB	ежедневно
Portal	4	8	sda: 56.00 GB	ежедневно

Назначение	Ядра CPU, шт	Память RAM, GB	SSD	Резервное копирование
			sdb: 40.00 GB	
HAProxy for Portal	2	2	sda: 56.00 GB sdb: 40.00 GB	ежедневно
HAProxy for Portal	2	2	sda: 56.00 GB sdb: 40.00 GB	ежедневно
HAProxy for PostgreSQL Platform	2	2	sda: 56.00 GB sdb: 40.00 GB	ежедневно
HAProxy for PostgreSQL Platform	2	2	sda: 56.00 GB sdb: 40.00 GB	ежедневно
PostgreSQL Platform	4	8	sda: 56.00 GB sdb: 40.00 GB	ежедневно
PostgreSQL Platform	4	8	sda: 56.00 GB sdb: 40.00 GB	ежедневно

Назначение	Ядра CPU, шт	Память RAM, GB	SSD	Резервное копирование
PostgreSQL Platform	4	8	sda: 56.00 GB sdb: 40.00 GB	ежедневно
HAProxy for MinIO Platform	2	2	sda: 56.00 GB sdb: 40.00 GB	ежедневно
HAProxy for MinIO Platform	2	2	sda: 56.00 GB sdb: 40.00 GB	ежедневно
MinIO Platform	2	4	sda: 56.00 GB sdb: 450.00 GB	ежедневно
MinIO Platform	2	4	sda: 56.00 GB sdb: 450.00 GB	ежедневно
MinIO Platform	2	4	sda: 56.00 GB sdb: 450.00 GB	ежедневно
MinIO Platform	2	4	sda: 56.00 GB	ежедневно

Назначение	Ядра CPU, шт	Память RAM, GB	SSD	Резервное копирование
			sdb: 450.00 GB	
Redis Platform	4	16	sda: 56.00 GB sdb: 40.00 GB	ежедневно

5.1.2 Требования к клиентской части

Характеристика	Значение
Диагональ монитора	15` и больше
Дисковое хранилище	200 Гб
Операционная система	Windows 10 и выше
Память	4 Гб
Процессор	Intel Core i5
Сетевой адаптер	1 сетевой адаптер Ethernet 100 Мбит/с

6. Квалификация, численность, функции персонала, оказывающего поддержку Платформе

6.1. Квалификация персонала Платформы "Marlin"

6.1.1. Роль "Разработчик"

Разработчики приложений на Платформе MARLIN обладают следующими компетенциями:

1. Опыт разработки на Java, Java Script не менее двух лет;
2. Опыт разработки API, обеспечивающих взаимодействие по следующим протоколам: gRPC, REST, Kafka messaging;
3. Опыт написания тестов с использованием JUnit;
4. Знание фреймворков: Spring, Spring Boot, Angular 2, React:
5. Опыт работы с СУБД (PostgreSQL, MongoDB);
6. Уверенное знание принципов работы систем контроля версий (Git);
7. Опыт работы с системами контейнеризации (Docker);
8. Базовые знания в области автоматизации процесса разработки (CI/CD);
9. Опыт использования agile-методологии разработки с использованием: code-review, merge-request, feature-branch, регулярная поставка релизов и т.п.
10. Опыт использования программного обеспечения Jira;
11. Владение стеком технологий, необходимых для разработки конкретного приложения (определяется при проектировании приложения).

6.1.2. Роль "Аналитик"

Опыт работы системным аналитиком от 3-х лет. Опыт сбора, анализа и управления требованиями на разработку. Опыт работы с REST API и/или SOAP. Опыт работы с БД. Умение работать с SQL-запросами. Знания и опыт использования нотаций UML , BPMN и др. Опыт разработки функциональных требований, технических заданий, проектной документации. Опыт с xml, xsd схемами, json.

6.1.3. Роль "Специалист по тестированию"

Опыт работы от 2-х лет; Понимание архитектуры клиент-серверных и веб-приложений; Опыт проведения функционального тестирования; Опыт тестирования WEB (или мобильных) приложений; Понимание подходов тестирования frontend, backend и умение применять их; Опыт тестирования API; Умение составлять тестовую документацию; Умение анализировать бизнес и функциональные требования; Опыт использования баг-трекеров и написания подробных и понятных баг-репортов; Понимание методологий Scrum и Agile; Опыт работы с инструментами для проведения тестирования (Postman, SoapUI, Swagger и т.п.); Знание REST/SOAP/JSON/XML; Опыт использования Git, Docker, Kubernetes.

6.2. Численность персонала Платформы "Marlin"

Количество разработчиков, задействованных в проектировании платформы и написании исходного кода, в количестве 7 человек.

Количество аналитиков, задействованных в процессе разработки и проектировании платформы, в количестве 3 человек.

Количество специалистов по тестированию, задействованных в процессе разработки и проектировании платформы, в количестве 2 человек.

6.3 Функциональные обязанности персонала Платформы "Marlin"

6.3.1. Функциональные обязанности сотрудника с ролью "Разработчик"

В функциональные обязанности сотрудника с ролью "Разработчик" входят:

1. Выполнение задач, поставленных ведущим разработчиком или руководителем проекта.
2. Анализ, проектирование, разработка и тестирование программного обеспечения в соответствии со стандартами, принятыми в рамках проекта.
3. Планирование своей работы и ведение проектной документации.
4. Обеспечение высокого качества своих разработок.
5. Согласование действий с другими разработчиками при проведении совместных разработок.

6.3.2. Функциональные обязанности сотрудника с ролью "Аналитик"

В функциональные обязанности сотрудника с ролью "Аналитик" входят:

1. Сбор, анализ и документирование требований (интервьюирование заказчиков).
2. Взаимодействие с заказчиками и экспертами предметных областей.
3. Разработка технических заданий (ТЗ) и постановка задач.
4. Построение моделей, алгоритмов, прототипов.
5. Проектирование программного обеспечения и комплексных систем.
6. Координация разработки, тестирования, приемки и внедрения ПО.
7. Составление документации и обучение пользователей.

8. Тестирование и устранение ошибок при разработке ПО.

6.3.3. Функциональные обязанности сотрудника с ролью "Специалист по тестированию"

В функциональные обязанности сотрудника с ролью "Специалист по тестированию" входят:

1. Разрабатывать планы, графики, методики и описания тестирования.
2. Моделировать ситуации, которые могут возникнуть в условиях эксплуатации программного обеспечения.
3. Работать в связке с разработчиком.
4. Создавать тест-планы, тест-кейсы.
5. Выполнять тестирование программных продуктов.
6. Выполнять нагрузочные тестирования.
7. Анализировать результаты, полученные во время прохождения тестов.
8. Классифицировать выявленные ошибки и заносит их в базу данных для текущего программного продукта.
9. Контролировать процесс ликвидации выявленных ошибок разработчиком ПО.
10. Консультировать клиентов.
11. Составлять документацию для проведения функционального тестирования.
12. Участвовать в проведении опытных эксплуатаций программных продуктов.
13. Заполнять таблицы баз данных тестовыми данными.

7. Режим функционирования программного обеспечения

При выполнении всех условий эксплуатации, заложенных в документации, Система имеет следующие основные режимы функционирования:

1. Штатный — основной режим функционирования. В данном режиме Система выполняет свои функции в соответствии с техническими и организационными инструкциями.
2. Сервисный режим — режим, при котором производится пуск, остановка и перезапуск Системы, резервное копирование накопленных данных, обновление системного и прикладного программного обеспечения, изменение

конфигурационных параметров Системы. При переключении в данный режим допустимо непродолжительное снижение общей производительности Системы. Сервисный режим не требует приостановки работы пользователей Системы в целом.

3. Аварийный режим – режим, который позволяет использовать доступные ресурсы Системы для сохранения информации, правильного закрытия информационных массивов, работающих приложений и операционных систем. Аварийный режим используется для выполнения минимально необходимых операций в условиях аварийного энергоснабжения компонентов Системы или выхода из строя части оборудования.

При условии регулярного регламентного обслуживания и мониторинга параметров работы Система обеспечивает длительно-непрерывное, круглосуточное функционирование в штатном режиме и в сервисном режиме. Система обеспечивает круглосуточную бесперебойную работу режиме 24/7/365.

Причинами нарушения непрерывного режима функционирования Системы и перехода из штатного в аварийный режим функционирования могут являться:

1. Отключение электроэнергии;
2. Недоступность каналов передачи данных (авария).

Действия в аварийном режиме включают:

1. Диагностирование инцидентов или проблем, связанных со сбоями или нештатными ситуациями в работе Системы
2. Восстановление при необходимости программно-аппаратной конфигурации Системы (сетевого и серверного оборудования);
3. Восстановление информации при ее утере средствами системы резервного копирования и восстановления;
4. Расследование причин нештатной ситуации и определение причин инцидента или проблемы.

Реагирование на нештатные ситуации включает оповещение обслуживающего персонала, принятие контрмер, необходимое восстановление информации, выработку и проведение профилактических мероприятий. После проведения первичной диагностики и определения причин нештатной ситуаций нештатный режим переходит в сервисный режим, а затем, после тестирования, – в штатный режим.