Руководство по установке программы для ЭВМ «Платформа Тайга Дайнемикс» (Taiga Dynamics)

Россия, г. Москва, наб Пресненская, д. 12, помещ. 14/5

Назначение документа

Данный документ описывает пример развертывания Программы для ЭВМ «Платформа Тайга Дайнемикс» (далее - “платформа” или “Taiga Dynamics”). Данный пример подразумевает запуск платформы в однонодовом kubernetes кластере с использованием Rancher k3s. Хранилища данных согласно данной инструкции запускаются с использованием технологии docker и плагина docker compose. При наличии на площадке развернутого kubernetes кластера и требуемых хранилищ данных платформа может быть настроена для их использования. Конкретный вариант размещения должен быть заранее согласован и описан в соответствующих проектных документах.

Термины и определения

Платформа Taiga Dynamics — Программа для ЭВМ «Платформа Тайга Дайнемикс», совокупность приложений и технических решений, предназначенная для обработки промышленных данных с целью оптимизации технологических процессов на основе алгоритмов машинного обучения.

Инфраструктурные компоненты —- совокупность элементов программного обеспечение (далее ПО), необходимого для функционирования и/или установки Платформы, включая Shell скрипты, базы данны (далее БД) хранилища и прочее.

Сервер — виртуальная машина или физический сервер, где будет установлена Платформа.

АРМ администратора — автоматизированное рабочее место (далее АРМ) администратора с которого осуществляется доступ к серверу Платформы.

Условные обозначения и имена переменных

${}	Переменные используемые в командных сценариях (далее скриптах), которые должны быть определены заранее. Могут быть заданы как переменные среды и/или заменены на корректные значения перед выполнением скрипта

Подготовка к установке

Ключ для распаковки архива с дистрибутивом платформы: Taiga2019&

Необходимые условия

- Сервер с установленной операционной системой Ubuntu Server 20.04, AstraLinux 1.7.2 или аналогичной;
- SSH доступ к серверу с АРМ администратора;
- HTTP/HTTPS серверу (TCP/443, TCP/80, TCP/9007);
- Пользователь с возможностью исполнения команд с административными привилегиями;
- Установленные пакеты:
    - openssl,
    - keytool,
    - update-ca-certificates.

Доставка архива с инфраструктурными компонентами

Используемые переменные:
${user} - Имя пользователя с правами доступа root
${server_name} - полное доменное имя Сервера Платформы

Скачать архив infra.tar.gz на АРМ администратора. Ссылка на скачивание Архива предоставляется сотрудниками ООО “Рокет Контрол” по запросу.

Загружаем скачанный архив infra.tar.gz на сервер используя scp. Запускаем терминал (CMD, PowerShell, etc.) и выполняем следующие команды:

```
cd ${directory_with_infra.tar.gz}
scp ./infra.tar.gz ${user}@${server_name}:~/
```

Подключение к Серверу.

Используемые переменные:
${user} - Имя пользователя с правами доступа root
${server_name} - полное доменное имя Сервера Платформы

Запусткаем терминал (CMD, PowerShell, etc.) и выполнить следующие команды:

```
ssh ${user}@${server_name}
```

Создание отдельного раздела для хранения данных Платформы

Примечание: Данный раздел носит ознакомительный характер т.к. конкретные команды, имена и размеры разделов могут отличаться от указанных в Руководстве. В случае возникновения сомнений или затруднений при создании выделенного раздела следует обратиться к сотрудникам ООО “Рокет Контрол” для консультации.

Примечание: Подключение физического/виртуального диска и создание файловой системы не входит в данное руководство.

Создаём директорию /datadrive в корне файловой системы, которая будет использована для хранения данных Платформы:

```
sudo mkdir /datadrive
```

Нужно идентифицировать UUID блочного устройства (диска) который будет использован. Для этого используем утилиту blkid:

```
sudo blkid
/dev/sda1: LABEL="cloudimg-rootfs" UUID="0029655f-5fe9-4742-ba81-acd4af8e1377" TYPE="ext4"
PARTUUID="6c79905f-1ae0-45dc-b129-bf65ab8ac6e3"
/dev/sda15: LABEL_FATBOOT="UEFI" LABEL="UEFI" UUID="1691-0908" TYPE="vfat"
PARTUUID="bac4f723-3e59-4be2-a124-dcb731025abe"
/dev/sdb1: UUID="769d17c3-a5e5-4455-b378-81aa6c6add6f" TYPE="ext4" PARTUUID="d1df9560-01"
/dev/sdc: UUID="3d47ad26-9fef-4da9-aeb9-da945ced6860" TYPE="ext4"
/dev/loop0: TYPE="squashfs"
/dev/loop1: TYPE="squashfs"
/dev/loop2: TYPE="squashfs"
/dev/sda14: PARTUUID="4c6e93d7-a073-409a-b063-0ad327552722"
```

Предположим что “наш” диск это /dev/sdc (имя блочного устройства (диска) может быть другим).

В файл /etc/fstab добавляем в конец файла строку содержащую UUID диска и mount point /datadrive:

```
 sudo nano /etc/fstab
# CLOUD_IMG: This file was created/modified by the Cloud Image build process
UUID=0029655f-5fe9-4742-ba81-acd4af8e1377       /        ext4   defaults,discard        0 1
UUID=1691-0908  /boot/efi       vfat    umask=0077      0 1
UUID=3d47ad26-9fef-4da9-aeb9-da945ced6860 /datadrive ext4 defaults 0 0
```

Выполняем команду по монтированию раздела:

```
sudo mount -a
```

Создание структуры директорий

Структура директорий будет выглядеть следующим образом:

```
datadrive
├──certs
│    ├──ck-server.key # ClickHouse: private self-signed private key
│    ├──ck-server.crt # ClickHouse: self-signed certificate
│    ├──client.password # Kafka: client password for access to keystore and truststore
│    ├──dhparam.pem # Clickhouse: diffie hellman group definition
│    ├──kafka-client.key # Kafka: self-singed private key
│    ├──kafka.client.keystore.jsk # Kafka: client keystore
│    ├──kafka-client-pub.pem # Kafka: self-singed certificate
│    ├──kafka.client.truststore.jsk # Kafka: client truststore
│    ├──kafka.server.keystore.jsk # Kafka: server keystore
│    ├──kafka.server.truststore.jsk # Kafka: server truststore
│    ├──ldap-cert.crt # LDAP: certificate for integration with LDAP (provided by Customer)
│    ├──minio-server.key # Mino: private self-signed private key
│    ├──minio-server.crt # Mino: self-signed certificate
│    ├──pg-server.key # postgres: private self-signed private key
│    ├──pg-server.crt # postgres:  self-signed certificate
│    ├──registry.key # registry: private self-signed private key
│    ├──registry.crt # registry:  self-signed certificate
│    ├──server.password # Kafka: server password for access to keystore and truststore
│    ├──ca.conf # configuration file for CA
│    ├──ca-key # private CA key
│    ├──ca-cert # public CA certificate
│    ├──serial.txt
│    ├──web-cert.crt # Domain: certificate (provided by Customer)
│    ├──web-cert.key # Domain: private key (provided by Customer)
│    └──pm.truststore.jks # trustore for PM component configuration
├──clickhouse # Clickhouse: data
├──docker # Docker: data
├──k3s # k3s: PVC
├──k3s-data # k3s: data
├──kafka # kafka: data
├──minio # minio: data
├──postgres # postgres: data
├──zookeeper-data # zookeeper: data
├──zookeeper-log # zookeeper: logs
└──platform # platform
    ├──docker-compose
    │   ├──clickhouse_console_config.xml
    │   ├──clickhouse_ssl_config.xml
    │   ├──clickhouse_users.xml
    │   ├──postgresql.conf
    │   └──docker-compose.yaml
    └──helm
        ├──custom-values.yaml
        ├──platform-application
```

```
        │   └──platform-application-1.0.0__{release_version}.tgz
        └──releases
            └──releases-{release_version}.tar
```

Создаем необходимые директории и будем последовательно наполнять их, по мере выполнения данного руководства:

```
cd /datadrive
mkdir certs clickhouse docker k3s k3s-data kafka minio postgres zookeeper-data zookeeper-logs
platform/docker-compose platform/helm
```

Изменяем владельцев следующих директорий на /datadrive. Данное действие необходимо для нормального функционирования приложений запущенных в контейнерах docker.

```
sudo chown -R 101:103/datadrive/clickhouse
sudo chown -R 1000:1000 /datadrive/kafka
sudo chown -R 1000:1000 /datadrive/zookeeper-log
sudo chown -R 1000:1000 /datadrive/zookeeper-data
sudo chown -R 999:999 /datadrive/postgres
```

Распаковка архива infra.tar.gz

Перемещаем infra.tar.gz и распакуем его в /datadrive:

```
cd ~
mv infta.tar.gz /datadrive
cd /datadrive
tar -xvf infta.tar.gz
```

Выпуск сертификатов

Используемые переменные:
${platform_password} - пароль для защиты целостности сертификатов
${pm_trustore_password} - пароль от trustore для platform management компонента
${server_name} - полное доменное имя Сервера Платформы

Создаем файл конфигурации ca.conf

```
cd /datadrive/certs
nano ./ca.conf
####################################################################
[ ca ]
default_ca    = CA_default      # The default ca section

[ CA_default ]

default_days     = 3650         # How long to certify for
default_crl_days = 30           # How long before next CRL
```

```
default_md       = sha256       # Use public key default MD
preserve         = no           # Keep passed DN ordering

x509_extensions = ca_extensions # The extensions to add to the cert

email_in_dn     = no            # Don't concat the email in the DN
copy_extensions = copy          # Required to copy SANs from CSR to cert

base_dir      = .
certificate   = $base_dir/ca-cert   # The CA certificate
private_key   = $base_dir/ca-key    # The CA private key
new_certs_dir = $base_dir/             # Location for new certs after signing
database      = $base_dir/index.txt    # Database index file
serial        = $base_dir/serial.txt   # The current serial number

unique_subject = no  # Set to 'no' to allow creation of
                     # several certificates with same subject.

####################################################################
[ req ]
default_bits       = 4096
default_keyfile    = cakey.pem
distinguished_name = server_distinguished_name
x509_extensions    = ca_extensions
string_mask        = utf8only

####################################################################
[ server_distinguished_name ]

commonName         = Common Name (e.g. server FQDN or YOUR name)
commonName_default = platform.ai

countryName         = Country Name (2 letter code)
countryName_default = RU

organizationName            = Organization Name (eg, company)
organizationName_default    = platform LTD

stateOrProvinceName         = State or Province Name (full name)
stateOrProvinceName_default = Moscow

localityName         = Locality Name (eg, city)
localityName_default = Moscow

organizationName            = Organization Name (eg, company)
organizationName_default    = RocketControl

####################################################################
[ ca_extensions ]

subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always, issuer
```

```
basicConstraints       = critical, CA:true
keyUsage               = keyCertSign, cRLSign

####################################################################
[ signing_policy ]
countryName            = optional
stateOrProvinceName    = optional
localityName           = optional
organizationName       = optional
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional

####################################################################
[ signing_req ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid,issuer
basicConstraints       = CA:FALSE
keyUsage               = digitalSignature, keyEncipherment
####################################################################
```

Создаем лог файл для учёта выпуска сертификатов:

```
# Prepare log files for signing with root CA
touch index.txt
echo '01' > serial.txt
```

Генерируем приватный ключ и сертификат CA:

```
# Generate self-signed root CA on 3650 days
openssl req -new -config ca.conf -newkey rsa:4096 -days 3650 -x509 -subj '/CN=platform.ai/O=OOO
Rocket Control/C=RU' -keyout ./ca-key -out ./ca-cert -nodes
```

Генерируем server keystore and truststore для Kafka:

```
# Generate server keystore and truststore
# Create Kafka Server Certificate and store in KeyStore:
keytool -genkey -keyalg RSA -alias platform-kafka-server -keystore kafka.server.keystore.jks
-validity 1000 -storepass ${platform_password} -keypass ${platform_password} -dname
"cn=platform-kafka-server, o=OOO Rocket Control, c=RU" -storetype pkcs12
```

Генерируем запрос на подпись сертификата (CSR):

```
# Create Certificate Signing Request (CSR):
keytool -keystore kafka.server.keystore.jks -alias platform-kafka-server -certreq -file
cert-file -storepass ${platform_password} -keypass ${platform_password} -ext
SAN=dns:${server_name},dns:broker,dns:zookeeper,ip:127.0.0.1,ip:0.0.0.0
```

Подписываем сертификат:

```
# Sign CSR with the CA:
openssl ca -batch -config ca.conf -policy signing_policy -extensions signing_req -out
cert-file-signed -infiles cert-file
```

Импортируем CA сертификат и сертификат для сервера Kafka в server truststore и keystore:

```
# Import CA certificate in KeyStore:
keytool -keystore kafka.server.keystore.jks -alias CA-Root-platform -import -file ca-cert
-storepass ${platform_password} -keypass ${platform_password} -noprompt
# Import Signed CSR In KeyStore:
keytool -keystore kafka.server.keystore.jks -alias platform-kafka-server -import -file
cert-file-signed -storepass ${platform_password} -keypass ${platform_password} -noprompt
# Import CA certificate In TrustStore:
keytool -keystore kafka.server.truststore.jks -alias CARoot -import -file ca-cert -storepass
${platform_password} -keypass ${platform_password} -noprompt
```

Выполняем аналогичные операции для client truststore и keystore:

```
# Generate client keystore and truststore
# Create Kafka client Certificate and store in KeyStore:
keytool -genkey -keyalg RSA -alias platform-kafka-client -keystore kafka.client.keystore.jks
-validity 1000 -storepass ${platform_password} -keypass ${platform_password} -dname
"cn=platform-kafka-client, o=OOO Rocket Control, c=RU" -storetype pkcs12
# Create Certificate signed request (CSR):
keytool -keystore kafka.client.keystore.jks -alias platform-kafka-client -certreq -file
cert-file -storepass ${platform_password} -keypass ${platform_password}
# Sign CSR with the CA:
openssl ca -batch -config ca.conf -policy signing_policy -extensions signing_req -out
cert-file-signed -infiles cert-file
# Import CA certificate in KeyStore:
keytool -keystore kafka.client.keystore.jks -alias CA-Root-platform -import -file ca-cert
-storepass ${platform_password} -keypass ${platform_password} -noprompt
# Import Signed CSR In KeyStore:
keytool -keystore kafka.client.keystore.jks -alias platform-kafka-client -import -file
cert-file-signed -storepass ${platform_password} -keypass ${platform_password} -noprompt
# Import CA certificate In TrustStore:
keytool -keystore kafka.client.truststore.jks -alias CARoot -import -file ca-cert -storepass
${platform_password} -keypass ${platform_password} -noprompt
```

Извлекаем сертификаты из keystore для дальнейшего использования:

```
#Extract private certificate from keystore
openssl pkcs12 -in kafka.client.keystore.jks -nocerts -out kafka-client.key -passin
pass:${platform_password} -passout pass:${platform_password}
#Extract public certificate from keystore
openssl pkcs12 -in kafka.client.keystore.jks -nokeys -out kafka-client-pub.pem -passin
pass:${platform_password} -passout pass:${platform_password}
```

Генерируем сертификат для ClickHouse:

```
# Create certificate for CLICKHOUSE
# Generate certificate and key for Clickhouse on 3560 days
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -out ./ck-server.crt -keyout
./ck-server.key -subj "/CN=${server_name}/O=OOO Rocket Control" -addext "subjectAltName =
DNS:${server_name}"
# Generate dhparam for Clickhouse
openssl dhparam -out dhparam.pem 4096
```

Записываем пароль в файл:

```
echo "${platform_password}" > client.password
echo "${platform_password}" > server.password
```

Генерируем сертификат для Postgres:

```
# Create certificate for Postgres
# Generate certificate and key for PG on 3650 days
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -out ./pg-server.crt -keyout
./pg-server.key -subj "/CN=${server_name}/O=OOO Rocket Control" -addext "subjectAltName =
DNS:${server_name}"
```

Изменяем права доступа для сертификата и ключа, чтобы в последствии Postgres мог получить доступ к сертификатам:

```
# change own amd mod for certificate Postgres
chmod 600 pg-server.key
chmod 640 pg-server.crt
chown systemd-coredump:root pg-server.key
chown systemd-coredump:root pg-server.crt
```

Генерируем сертификат для Minio:

```
# Create certificate for MINIO
# Generate certificate and key for Minio on 3650 days
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -out ./minio-server.crt -keyout
./minio-server.key -subj "/CN=${server_name}/O=OOO Rocket Control" -addext "subjectAltName =
DNS:${server_name}"
```

Генерируем сертификат для docker registry:

```
# Create certificate for docker registry
# Generate certificate and key for docker registry on 3650 days
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -out ./registry.crt -keyout ./registry.key
-subj "/CN=${server_name}/O=OOO Rocket Control" -addext "subjectAltName = DNS:${server_name}"
```

Генерируем truststore для platform management компонента и импортируем сертификаты для postgres и minio:

```
keytool -genkeypair -noprompt -alias tmpCertToBeDeleted -keyalg RSA -keysize 2048
-storepass "${pm_truststore_password}" -keystore truststore.jks -dname "${server_name},
O=OOO Rocket Control"
keytool -delete -noprompt -alias tmpCertToBeDeleted -storepass
"${pm_truststore_password}" -keystore truststore.jks
keytool -import -noprompt -keystore truststore.jks -storepass "${pm_truststore_password}"
-alias postgres -file pg-server.crt
keytool -import -noprompt -keystore truststore.jks -storepass "${pm_truststore_password}"
-alias minio -file minio-server.crt
```

Загрузка сертификатов в доверенное хранилище сервера

Выполнить следующие команды:

```
cp /datadrive/certs/ /usr/local/share/ca-certificates/
update-ca-certificates
```

Установка инфраструктурных компонент

Переходим в директорию с инсталляционными пакетами (появится после распаковки архива infra.tar.gz):

```
cd /datadrive/platform_infra
```

Примечание: Все дальнейшие команды выполняются в директории /datadrive/platform_infra, если не обозначено другое.

Установка и настройка Docker

Используемые переменные:
${user} - Имя пользователя с правами доступа root.

Переходим в директорию docker и устанавливаем все *.deb пакеты:

```
# Docker install
cd /datadrive/platform_infra/docker
sudo dpkg -i  containerd.io.deb
sudo dpkg -i  docker-ce-cli.deb
sudo dpkg -i  docker-ce.deb
```

Создаём файл /etc/docker/daemon.json и вносим следующую конфигурацию:

```
sudo nano /etc/docker/daemon.json
{
    "graph": "/datadrive/docker"
}
```

Перезапускаем Docker для того чтобы применить конфигурацию:

```
sudo systemctl restart docker
```

Добавляем пользователя **${user}** в группу докер, чтобы не использовать sudo для запуска докер команд:

```
sudo usermod -aG docker ${user}
```

```
newgrp docker
```

Примечание: Возможно, потребуется перезапустить SSH сессию, чтобы настройки вступили в силу.

Установка docker-compose

Выполняем следующие команды:

```
cd /datadrive/platform_infra/docker
sudo cp "./docker/docker-compose-linux-x86_64" /usr/local/bin/docker-compose
sudo chmod +x  /usr/local/bin/docker-compose
```

Установка Helm.

Выполняем следующие команды:

```
tar -xvf "./helm/helm.tar.gz"
sudo cp "./linux-amd64/helm" /usr/local/bin/helm
sudo chmod +x  /usr/local/bin/helm
```

Установка private docker registry

Выполняем следующие команды:

```
docker load -i ./images/registry.tar
docker run -d \
  --restart=always \
  --name registry \
  -v /datadrive/certs:/certs \
  -e REGISTRY_HTTP_ADDR=0.0.0.0:5001 \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/datadrive/certs/registry.crt \
  -e REGISTRY_HTTP_TLS_KEY=/datadrive/certs/registry.key \
  -p 5001:5001 \
  registry:2.8.0
```

Установка и настройка k3s Rancher.

Используемые переменные:
${server_name} -доменное имя сервера.

Выполняем следующие команды:

```
cd /datadrive/platform_infra/k3s/
docker load -i k3s-airgap-images-amd64.tar

for dockerimage in $(docker images --format "{{.Repository}}:{{.Tag}}"); do echo "$dockerimage";
dockerimagename=$(echo "$dockerimage" | grep -oP '\/.+') ; docker tag $dockerimage
"${server_name}:5001$dockerimagename"; done
```

```
for image in $(docker images --format "{{.Repository}}:{{.Tag}}" | grep ${server_name}); do
docker push $image; done


cp "./k3s/k3s" /usr/local/bin/k3s
chmod +x  /usr/local/bin/k3s

K3S_KUBECONFIG_MODE="644" \
INSTALL_K3S_VERSION="v1.23.2+k3s1" \
INSTALL_K3S_EXEC='server --disable traefik --data-dir /datadrive/k3s-data
--default-local-storage-path /datadrive/k3s' \
./k3s/install.sh
```

Добавляем файл /etc/rancher/k3s/registries.yaml

```
nano /etc/rancher/k3s/registries.yaml

mirrors:
  docker.io:
    endpoint:
      - ${server_name}:5001
  quay.io:
    endpoint:
      - ${server_name}:5001
  k8s.gcr.io:
    endpoint:
      - ${server_name}:5001
```

Перезапускаем k3s:

```
sudo systemctl restart k3s
```

Установка и настройка cert-manager и ingress-nginx

Используемые переменные:
${server_name} -доменное имя сервера.

Выполняем следующие команды:

```
# Deploy cert-manager and ingress-nginx
cd /datadrive/platform_infra/images
docker load -i cert-manager.tar
docker load -i ingress-nginx.tar

for dockerimage in $(docker images --format "{{.Repository}}:{{.Tag}}"); do echo "$dockerimage";
dockerimagename=$(echo "$dockerimage" | grep -oP '\/.+') ; docker tag $dockerimage
"${server_name}:5001$dockerimagename"; done
for image in $(docker images --format "{{.Repository}}:{{.Tag}}" | grep ${server_name}); do
docker push $image; done

CONTROLLER_DIGEST=$(docker inspect --format='{{ .RepoDigests }}'
"${server_name}:5001/ingress-nginx/controller:v1.1.1" | awk -F '[@\\[\\]]' '{print $3}')
```

```
ADMISSSION_DIGEST=$(docker inspect --format='{{ .RepoDigests }}'
"${server_name}:5001/ingress-nginx/kube-webhook-certgen:v1.1.1" | awk -F '[@\\[\\]]' '{print
$3}')

# Check variables
echo $CONTROLLER_DIGEST
echo $ADMISSSION_DIGEST

cd  /datadrive/platform_infra/helm_charts
helm upgrade --install cert-manager cert-manager-v1.7.1.tgz -n cert-manager --set
installCRDs=true
helm upgrade --install ingress-nginx ./helm_charts/ingress-nginx-4.0.17.tgz -n ingress-nginx
--set-string
"controller.config.ssl-reject-handshake"="true","controller.image.digest"="$CONTROLLER_DIGEST","
controller.admissionWebhooks.patch.image.digest"="$ADMISSSION_DIGEST"
```

Установка компонентов Платформы

Конфигурация docker-compose

Переходим в директорию /datadrive/platform/docker-compose/

```
cd /datadrive/platform/docker-compose/
```

Примечание: Все дальнейшие команды выполняются в директории /datadrive/platform_infra, если не обозначено другое.

Создание конфигурационного файла для clickhouse_console_config

```
nano clickhouse_console_config.xml
```

```
<?xml version="1.0"?>
<clickhouse>
    <logger>
        <console>1</console>
    </logger>
</clickhouse>
```

Создание конфигурационного файла для clickhouse_ssl_config

```
nano  clickhouse_ssl_config.xml
```

Используемые переменные:
${client_password} - пароль находится в файле /datadrive/certs/client.password

```
<clickhouse>
    <https_port>9999</https_port>

    <openSSL>
        <server>
            <certificateFile>/tmp/server.crt</certificateFile>
            <privateKeyFile>/tmp/server.key</privateKeyFile>
            <dhParamsFile>/tmp/dhparam.pem</dhParamsFile>
            <verificationMode>none</verificationMode>
            <loadDefaultCAFile>true</loadDefaultCAFile>
            <cacheSessions>true</cacheSessions>
            <disableProtocols>sslv2,sslv3</disableProtocols>
            <preferServerCiphers>true</preferServerCiphers>
        </server>
```

```
        <client>
            <loadDefaultCAFile>true</loadDefaultCAFile>
            <cacheSessions>true</cacheSessions>
            <disableProtocols>sslv2,sslv3</disableProtocols>
            <preferServerCiphers>true</preferServerCiphers>
            <invalidCertificateHandler>
                <name>RejectCertificateHandler</name>
            </invalidCertificateHandler>
        </client>
    </openSSL>

    <kafka>
        <security_protocol>ssl</security_protocol>
        <ssl_key_location>/tmp/kafka-client.key</ssl_key_location>
        <ssl_key_password>${client_password}</ssl_key_password>
        <ssl_certificate_location>/tmp/kafka-client-pub.pem</ssl_certificate_location>
        <ssl_ca_location>/tmp/platform-ca.pem</ssl_ca_location>
    </kafka>

    <profiles>
        <default>
            <stream_flush_interval_ms>1000</stream_flush_interval_ms>
        </default>
    </profiles>

    <logger>
        <console>1</console>
    </logger>
</clickhouse>
```

Создание конфигурационного файла clickhouse_users.xml

```
nano clickhouse_users.xml
```

Используемые переменные:
${clickhouseUser} - имя пользователя для сервера clickhouse.
${clickhousePassword}** - пароль от пользователя **${clickhouseUser}.

```
<?xml version="1.0"?>
<clickhouse>
    <profiles>
        <default>
            <max_memory_usage>10000000000</max_memory_usage>
            <load_balancing>random</load_balancing>
            <max_partitions_per_insert_block>100000</max_partitions_per_insert_block>
        </default>
```

```
        <readonly>
            <readonly>1</readonly>
        </readonly>
    </profiles>

    <users>
        <${clickhouseUser}>
            <password>${clickhousePassword}</password>
            <networks>
                <ip>::/0</ip>
            </networks>
            <profile>default</profile>
            <quota>default</quota>
            <access_management>1</access_management>
        </${clickhouseUser}>
    </users>

    <quotas>
        <default>
            <interval>
                <duration>3600</duration>
                <queries>0</queries>
                <errors>0</errors>
                <result_rows>0</result_rows>
                <read_rows>0</read_rows>
                <execution_time>0</execution_time>
            </interval>
        </default>
    </quotas>
</clickhouse>
```

Создание конфигурационного файла для postgres

```
nano postgresql.conf
```

```
log_destination = 'csvlog'
logging_collector = 'on'
pgaudit.log = 'write, ddl, role'
pgaudit.log_relation = 'on'
log_connections = 'on'
log_disconnections = 'on'
listen_addresses = '*'
port = 5432
idle_in_transaction_session_timeout = 60000

#Connectivity
max_connections = 500
superuser_reserved_cennections = 3
```

```
#Memory settings
shared_buffers = '2048 MB'
work_mem = '64 MB'
maintenance_work_mem = '320 MB'
huge_pages = off
effective_cache_size = '6 GB'

#Monitoring
shared_preload_libraries = 'pg_stat_statements,pgaudit'
track_io_timing=on
track_functions=p1
pg_stat_statement.max = 10000
pg_stat_statements.track = all

#Checkpointing
checkpoint_timeout = '15 min'
checkpoint_complection_target = 0.9
max_wal_size = '1024 MB'
min_wal_size = '512 MB'

#WAL writing
wal_buffer = 1
wal_writer_delay = '200 ms'
wal_writer_flush_after = '1 MB'

#Background writer
bgwriter_delay = '200 ms'
bgwriter_lru_maxpages = 100
bgwriter_lru_multiplier = 2.0
bgwriter_flush_after = 0

#Parallel queries
max_worker_processes = 4
max_parallel_workers_per_gather = 2
max_parallel_maintenance_workers = 2
max_parallel_workers = 4
parallel_leader_participation = on

#Advanced features
enable_partitionwise_join = on
enable_partitionwise_aggregate = on

#ssl
ssl = true
ssl_cert_file = '/etc/postgresql-certs/server.key'
ssl_key_file = '/etc/postgresql-certs/server.crt'
```

Создание конфигурационного файла для zookeeper

```
nano zoo.cfg
```

```
autopurge.purgeInterval=1
autopurge.snapRetainCount=5
```

Создание конфигурационного файла docker-compose.yaml

Используемые переменные:
${server_name} - полное доменное имя сервера (пример: my-awesome-server.example.com)

${images_pgaudit}** - **${server_name}:5001:pg13_audit:v2022-03-30

${postgresUser} - администратор базы данных в postgres

${postgresPassword}** - пароль для пользователя **${postgresUser}

${keystore_password} - пароль используемый для генерации /etc/kafka/secrets/kafka.server.keystore.jks (пароль находится в файле /datadrive/certs/server.password)

${truststore_password} - пароль используемый для генерации /etc/kafka/secrets/kafka.server.truststore.jks (пароль находится в файле /datadrive/server/client.password)

${client_password} - пароль находится в файле /datadrive/certs/client.password

${minioAccessKey} - UUID version 4

${minioSecretKey} - UUID version 4

```
nano docker-compose.yaml
```

```
version: "3.4"

services:
  db:
    image: ${images_pgaudit}
    restart: always
    ports:
      - "5432:5432"
    volumes:
      - /datadrive/postgres:/var/lib/postgresql/data
      - ./postgresql.conf:/etc/postgresql/postgresql.conf
      - /datadrive/certs/pg-server.crt:/etc/postgresql-certs/server.crt
      - /datadrive/certs/pg-server.key:/etc/postgresql-certs/server.key
    environment:
      POSTGRES_DB: "main"
      POSTGRES_USER: ${postgresUser}
      POSTGRES_PASSWORD: ${postgresPassword}
    logging:
```

```
      options:
        max-size: 10m
    command: postgres -c 'config_file=/etc/postgresql/postgresql.conf'

  db-storage:
    image: clickhouse/clickhouse-server:22.4.3.3
    restart: always
    ports:
      - "8123:8123"
      - "9000:9000"
      - "9004:9004"
      - "9999:9999"
    volumes:
      - /datadrive/clickhouse:/var/lib/clickhouse
      -
./clickhouse_console_config.xml:/etc/clickhouse-server/config.d/clickhouse_console_config.xml
      - ./clickhouse_users.xml:/etc/clickhouse-server/users.xml
      - ./clickhouse_ssl_config.xml:/etc/clickhouse-server/config.d/clickhouse_ssl_config.xml
      - /datadrive/certs/ck-server.crt:/tmp/server.crt
      - /datadrive/certs/ck-server.key:/tmp/server.key
      - /datadrive/certs/ck-dhparam.pem:/tmp/dhparam.pem
      - /datadrive/certs/ca-cert:/tmp/platform-ca.pem:ro
      - /datadrive/certs/kafka-client.key:/tmp/kafka-client.key:ro
      - /datadrive/certs/kafka-client-pub.pem:/tmp/kafka-client-pub.pem:ro
    logging:
      options:
        max-size: 10m
    deploy:
      resources:
        limits:
          memory: 12G
        reservations:
          memory: 1G

  zookeeper:
    restart: always
    image: confluentinc/cp-zookeeper:6.2.0
    hostname: zookeeper
    container_name: zookeeper
    volumes:
      - /datadrive/platform/docker-compose/zoo.cfg:/etc/zookeeper/conf/zoo.cfg
      - /datadrive/zookeeper-data:/var/lib/zookeeper/data
      - /datadrive/zookeeper-log:/var/lib/zookeeper/log
      -
/datadrive/certs/kafka.server.keystore.jks:/etc/kafka/secrets/kafka.server.keystore.jks:ro
      -
datadrive/certs/kafka.server.truststore.jks:/etc/kafka/secrets/kafka.server.truststore.jks:ro
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2182
      ZOOKEEPER_TICK_TIME: 2000
```

```
      ZOOKEEPER_SECURE_CLIENT_PORT: 2181
      ZOOKEEPER_SERVER_CNXN_FACTORY: org.apache.zookeeper.server.NettyServerCnxnFactory
      ZOOKEEPER_SSL_KEYSTORE_LOCATION: '/etc/kafka/secrets/kafka.server.keystore.jks'
      ZOOKEEPER_SSL_KEYSTORE_PASSWORD: '${keystore_password}'
      ZOOKEEPER_SSL_TRUSTSTORE_LOCATION: '/etc/kafka/secrets/kafka.server.truststore.jks'
      ZOOKEEPER_SSL_TRUSTSTORE_PASSWORD: '${truststore_password}'
      ZOOKEEPER_SSL_CLIENT_AUTH: need
      ZOOKEEPER_AUTH_PROVIDER_X509: org.apache.zookeeper.server.auth.X509AuthenticationProvider
#use for ssl

  broker:
    restart: always
    image: confluentinc/cp-kafka:6.2.0
    hostname: broker
    container_name: broker
    depends_on:
      - zookeeper
    volumes:
      - /datadrive/kafka:/var/lib/kafka/data
      -
/datadrive/certs/kafka.client.keystore.jks:/etc/kafka/secrets/kafka.client.keystore.jks:ro
      -
/datadrive/certs/kafka.client.truststore.jks:/etc/kafka/secrets/kafka.client.truststore.jks:ro
      -
/datadrive/certs/kafka.server.keystore.jks:/etc/kafka/secrets/kafka.server.keystore.jks:ro
      -
/datadrive/certs/kafka.server.truststore.jks:/etc/kafka/secrets/kafka.server.truststore.jks:ro
      - /datadrive/certs/client.password:/etc/kafka/secrets/client.password:ro
      - /datadrive/certs/server.password:/etc/kafka/secrets/server.password:ro
      - /datadrive/certs/ck-server.crt:/tmp/server.crt
      - /datadrive/certs/ck-server.key:/tmp/server.key
    ports:
      - "29092:29092"
      - "19092:19092"
      - "9092:9092"
      - "9101:9101"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_JMX_PORT: 9101
      KAFKA_JMX_HOSTNAME: localhost
      KAFKA_LOG_RETENTION_BYTES: 10737418240
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ZOOKEEPER_SSL_CLIENT_ENABLE: 'true'
      KAFKA_ZOOKEEPER_CLIENT_CNXN_SOCKET: org.apache.zookeeper.ClientCnxnSocketNetty
      KAFKA_ZOOKEEPER_SSL_KEYSTORE_LOCATION: '/etc/kafka/secrets/kafka.client.keystore.jks'
      KAFKA_ZOOKEEPER_SSL_KEYSTORE_PASSWORD: '${client_password}'
      KAFKA_ZOOKEEPER_SSL_TRUSTSTORE_LOCATION: '/etc/kafka/sercerts/kafka.client.keystore.jks'
```

```
      KAFKA_ZOOKEEPER_SSL_TRUSTSTORE_PASSWORD: '${client_password}'
      KAFKA_ZOOKEEPER_SET_ACL: 'true'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: SSL:SSL,SSL_EXTERNAL:SSL
      KAFKA_ADVERTISED_LISTENER: SSL://broker:29092,SSL_EXTERNAL://${FQDN_host_server}:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: SSL
      KAFKA_SSL_KEYSTORE_FILENAME: 'kafka.server.keystore.jks'
      KAFKA_SSL_KEYSTORE_CREDENTIALS: 'server.password'
      KAFKA_SSL_KEY_CREDENTIALS: 'server.password'
      KAFKA_SSL_TRUSTSTORE_FOLENAME: 'kafka.server.truststore.jks'
      KAFKA_SSL_TRUSTSTORE_CREDENTIALS: 'server.password'
      KAFKA_SSL_CLIENT_AUTH: 'required'

  minio:
    image: minio/minio:RELEASE.2022-04-12T06-55-35Z
    restart: always
    command: server /data --address ":9005" --console-address ":9007"
    ports:
      - "9005:9005"
      - "9007:9007"
    environment:
      MINIO_ROOT_USER: ${minioAccessKey}
      MINIO_ROOT_PASSWORD: ${minioSecretKey}
    volumes:
      - /datadrive/minio:/data
      - /datadrive/certs/minio-server.crt:/root/.minio/certs/public.crt
      - /datadrive/certs/minio-server.key:/root/.minio/cert/public.key
" > docker-compose.yaml
```

Запуск docker-compose

Примечание: Запуск docker-compose происходит в директории /datadrive/platform/docker-compose/.

Выполняем следующую команду:

```
docker-compose up -d
```

Инициализация БД в Postgresql

Используемые переменные:
${postgresUser} - пользователь из файла /datadrive/platform/docker-compose/docker-compose.yaml
${keycloak_password} - сгенерированный пароль
${platformmanagement_password} - сгенерированный пароль
${streamstorage_password} - сгенерированный пароль

Выполняем подключение вовнутрь контейнера с postgres и создадим пользователей и БД:

```
docker exec -it docker-compose_db_1 bash
psql -U ${postgresUser} -h localhost -p 5432 -d postgres
```

```
CREATE USER platform_keycloak;
CREATE DATABASE keycloak;
ALTER USER platform_keycloak PASSWORD '${keycloak_password}';
ALTER DATABASE keycloak OWNER TO platform_keycloak;
COMMENT ON ROLE platform_keycloak IS 'Administration user for keycloack service';
COMMENT ON DATABASE keycloak IS 'keycloack service database';

CREATE USER platform_platformmanagement;
CREATE DATABASE platformmanagement;
ALTER USER platform_platformmanagement PASSWORD '${platformmanagement_password}';
ALTER DATABASE platformmanagement OWNER TO platform_platformmanagement;
COMMENT ON ROLE platform_platformmanagement IS 'Administration user for Platform Management
service';
COMMENT ON DATABASE platformmanagement IS 'platformmanagement service database';

CREATE USER platform_streamstorage;
CREATE DATABASE streamstorage;
ALTER USER platform_streamstorage PASSWORD '${streamstorage_password}';
ALTER DATABASE streamstorage OWNER TO platform_streamstorage;
COMMENT ON ROLE platform_streamstorage IS 'Administration user for Stream Storage service';
COMMENT ON DATABASE streamstorage IS 'streamstorage service database';

###
CREATE EXTENSION pgaudit;
CREATE EXTENSION pg_stat_statements;

### exit
\q
```

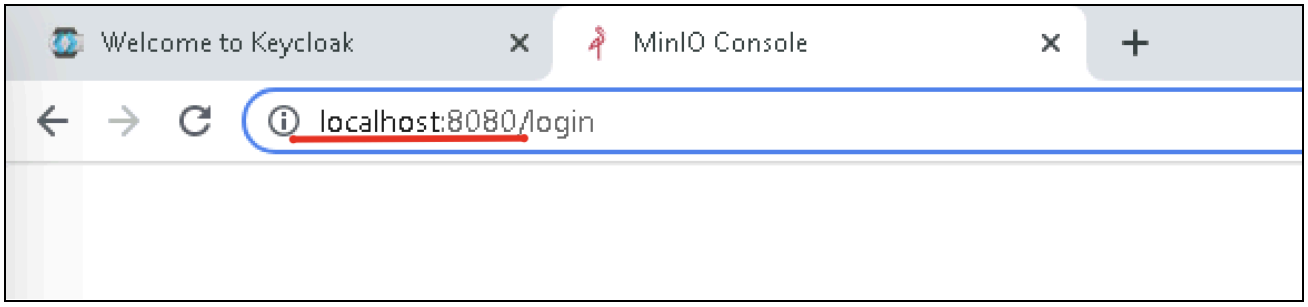
Создание bucket в minio

Переменные значения:
${host_name} - имя сервера к которому подключаются по ssh
${admin} - имя пользователя для подключения по ssh

Запускаем терминал (CMD, PowerShell, etc.)

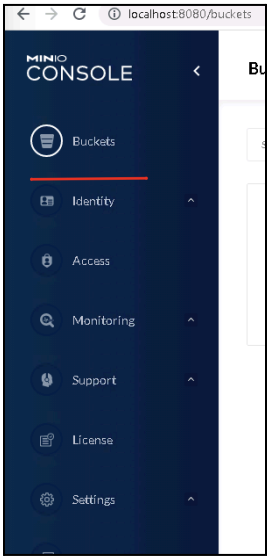
```
ssh -L 8080:localhost:9007 ${admin}@${host_name}
```

Заходим в браузер (chrome, edge, IE, etc.) в адресную строку вставляем localhost:8080

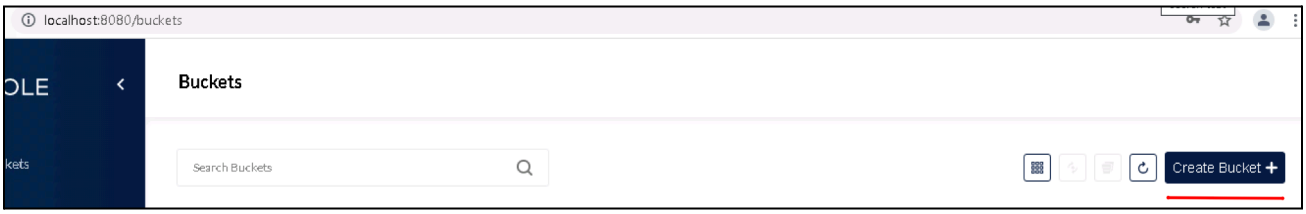


На стартовой странице minio вводим значения
MINIO_ROOT_USER: **${minioAccessKey}**
MINIO_ROOT_PASSWORD: **${minioSecretKey}**
из файла /datadrive/platform/docker-compose/docker-compose.yaml

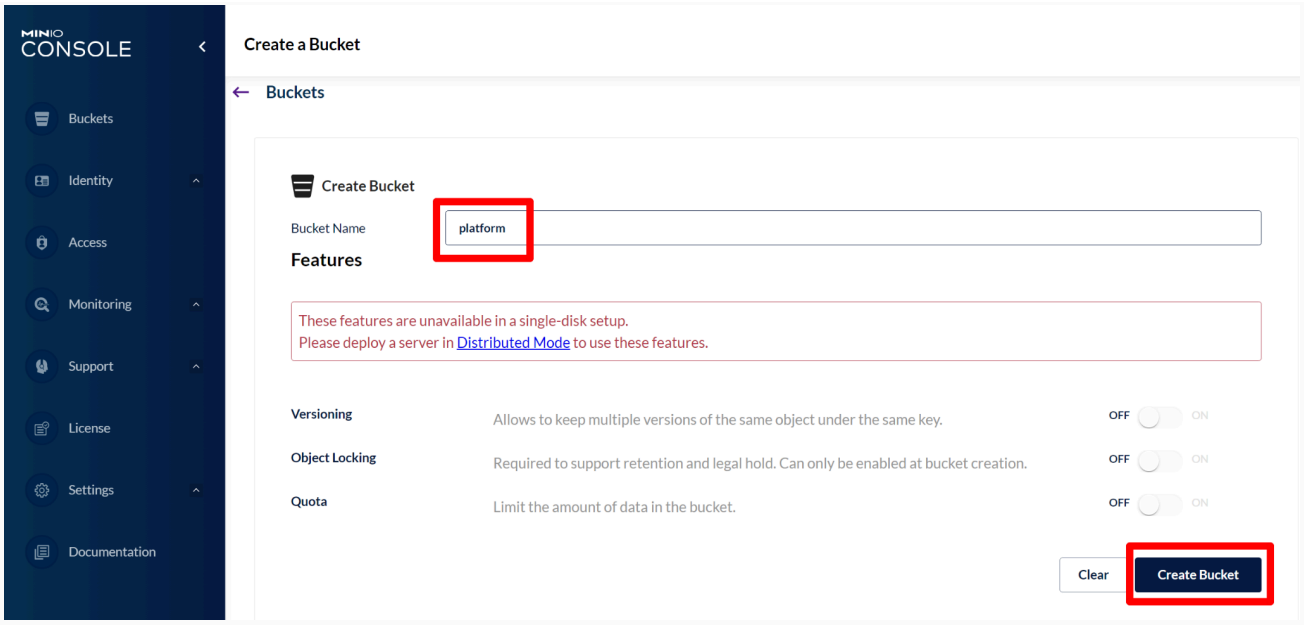
В левом вертикальном меню выбираем Buckets:



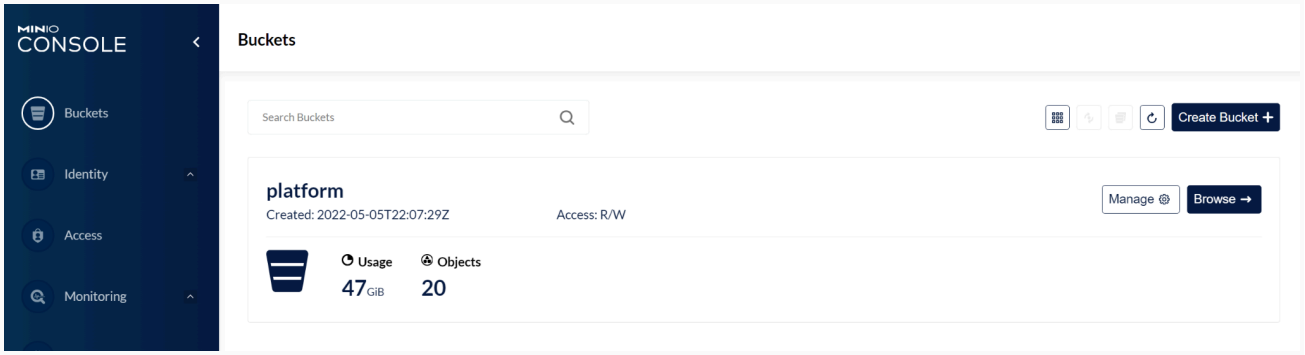
В правом верхнем углу нажимаем на Create Bucket:



В поле Bucket Name вводим platform после чего нажимаем Create Bucket:



Убеждаемся, что bucket создан:



После создания Bucket необходимо закрыть терминал открытый в начале пункта.

Создание пользователя reader в Clickhouse

Используемые переменные:
${click_password} - пароль из файла /datadrive/platform/docker-compose/docker-compose.yaml
${random_password} - сгенерированный пароль
https://www.lastpass.com/features/password-generator

Подключаемся к контейнеру:

```
docker exec -it platform-db-storage-1 bash
```

После того как подключились к контейнеру вводим следующие команды:

```
clickhouse-client --user platform --password ${click_password}
ALTER USER reader IDENTIFIED WITH plaintext_password BY ${random_password};
```

```
exit;
exit
```

Переходим в директорию директорию /datadrive/platform/helm/

```
cd /datadrive/platform/helm/
```

Запуск Платформы

Создание файл custom-values.yaml

```
nano custom-values.yaml
```

Используемые переменные:
${web-cert.key} - ключ от сертификата сгенерированного командой заказчика в PEM формате для https. Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/web-cert.key | base64 | tr -d '\n'
```

${web-cert.crt} - сертификат сгенерированный командой заказчика в PEM формате для https. Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/web-cert.crt | base64 | tr -d '\n'
```

${server_name} - полное доменное имя сервера

${FQDN_host_server:port_registry}** - **${server_name}:5001

${cert_docker_registry} - сертификат для docker registry. Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/registry.crt | base64 | tr -d '\n'
```

${cert_for_ldaps} - сертификат сгенерированный командой заказчика в PEM формате для ldaps. Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/cert_for_ldaps.crt | base64 | tr -d '\n'
```

${UUID_auth} - UUID version 4

${UUID_metric} - UUID version 4

${UUID_log} - UUID version 4

**${your_password_for_keycloack_UI}** - пароль для пользователя Taiga для входа в UI keycloack https://${server_name}/auth/

${platform_keycloak} - пользователь **platform_keycloak** созданный во время инициализации базы в postgres

${keycloak_password} - пароль от пользователя **platform_keycloak** созданного во время инициализации базы в postgres

${pg-public.crt} - сертификат ssl для postgres. Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/pg-public.crt | base64 | tr -d '\n'
```

${UUID_stream} - UUID version 4

${your_clickhouse_user_from_clickhouse_users.xml} - пользователь из файла /datadrive/platform/docker-compose/clickhouse_users.xml

${your_clickhouse_user_password_from_clickhouse_users.xml} - пароль от пользователя из файла /datadrive/platform/docker-compose/clickhouse_users.xml

${reader_password} - пароль созданный в пункте “Создание пользователя reader в Clickhouse”

${ck-server.crt} - сертификат ssl для clickhouse. Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/ck-server.crt | base64 | tr -d '\n'
```

${server_password} - пароль из файла /datadrive/certs/server.password

${base64_keystore.jks} - значение репозитория сертификатов kafka.client.keystore.jks. Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/kafka.client.keystore.jks | base64 | tr -d '\n'
```

${base64_truststore.jks} - значение репозитория сертификатов kafka.client.truststore.jks.

Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/kafka.client.truststore.jks | base64 | tr -d '\n'
```

${platform_streamstorage} - пользователь **platform_streamstorage** созданий во время инициализации базы в postgres

${streamstorage_password} - пароль от пользователя **platform_streamstorage** созданного во время инициализации базы в postgres

${pm_trustore_password} - пароль от trustore для platform management компонента

${pm_trustore_b64} - значение репозитория сертификатов truststore.jks. Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/truststore.jks | base64 | tr -d '\n'
```

${UUID_platform} - UUID version 4

${UUID_pod} - UUID version 4

${platform_platformmanagement} - пользователь **`platform_platformmanagement`** созданий во время инициализации базы в postgres

${platformmanagement_password} - пароль от пользователя **`platform_platformmanagement`** созданного во время инициализации базы в postgres

${minioAccessKey} - значение UUID из файла /datadrive/platform/docker-compose/docker-compose.yaml

${minioSecretKey} - значение UUID из файла /datadrive/platform/docker-compose/docker-compose.yaml

${minio-public.crt} - сертификат ssl для minio. Значение переменной должно быть передано в base64 кодировке. Для этого необходимо выполнить команду:

```
cat /datadrive/certs/minio-public.crt | base64 | tr -d '\n'
```

${AES_platform} - случайных 16, 24 или 32 символ конвертируемый в формат base64

```
printf '1234567890123456' | base64
```

${UUID_connectors} - UUID version 4

Файл конфигурации должен выглядеть следующим образом:

```
cert:
  enabled: true
  customeCa:
    enables: true
    key: ${nn.key}
    certificate: ${nn.crt}

gateway:
```

```
  ingressClass: "nginx"
  schema: "https"
  host: ${server_name}
  annotations:
    nginx.ingress.kubernetes.io/proxy-ssl-protocols: "TLSv1.2 TLSv1.3"

components:
  global:
    dockerImageRegistry: "${FQDN_host_server:port_registry}"
    privateRepository:
      data: ${cert_docker_registry}
    customCa:
      enabled: true
      certificate: ${cert_for_ldaps}
  router:
    auth:
      secret: ${UUID_auth}
    metricViewer:
      auth:
        secret: ${UUID_metric}
    auditCollector:
      enabled: true
    logViewer:
      auth:
        secret: ${UUID_log}

  Keycloak:
    auth:
      password: ${your_password_for_keycloack_UI}
    db:
      POSTGRES_HOST: ${server_name}
      POSTGRES_PORT: 5432
      POSTGRES_USER: ${platform_keycloak}
      POSTGRES_PASS: ${keycloak_password}
      POSTGRES_DB: keycloak
      POSTGRES_MAX_CONNECTIONS: "10"
      POSTGRES_USE_SSL: "true"
      POSTGRES_ROOT_CERT_B64: ${pg-public.crt}
    logging:
      application: "info"

  streamStorage3:
    auth:
      secret: ${UUID_stream}
    db:
      POSTGRES_HOST: ${server_name}
      POSTGRES_PORT: 5432
      POSTGRES_USER: ${platform_streamstorage}
      POSTGRES_PASS: ${streamstorage_password}
      POSTGRES_DB: streamstorage
      POSTGRES_USE_SSL: "true"
      POSTGRES_ROOT_CERT_B64: ${pg-public.crt}
```

```
    CLICKHOUSE_HOST: ${server_name}
    CLICKHOUSE_PORT: "9004"
    CLICKHOUSE_HTTP_PORT: "9999"
    CLICKHOUSE_USERNAME: ${your_clickhouse_user_from_clickhouse_users.xml}
    CLICKHOUSE_PASSWORD: ${your_clickhouse_user_password_from_clickhouse_users.xml}
    CLICKHOUSE_READER_PASSWORD: ${reader_password}
    CLICKHOUSE_KAFKA_BOOTSTRAP_SERVER_HOST: ${server_name}
    CLICKHOUSE_KAFKA_BOOTSTRAP_SERVER_PORT: "9092"
    CLICKHOUSE_IS_REPLICATED: "false"
    CLICKHOUSE_SSL_ROOT_CERT_B64: ${ck-server.crt}
    CLICKHOUSE_USE_SSL: true
    KAFKA_HOST: ${server_name}
    KAFKA_PORT: "9092"
    KRB5_CONF_DATA: ""
    KAFKA_SSL_PROPERTIES_DATA: |
            security.protocol=SSL
            ssl.truststore.location=/tmp/kafka/kafka.truststore.jks
            ssl.truststore.password=${server_password}
            ssl.keystore.location=/tmp/kafka//kafka.keystore.jks
            ssl.keystore.password=${server_password}
            ssl.client.auth=required
            ssl.enabled.protocols=TLSv1.2
    KAFKA_KEYSTORE_B64: ${base64_keystore.jks}
    KAFKA_TRUSTSTORE_B64: ${base64_truststore.jks}
    KAFKA_KEYTAB_B64: ""
  rateLimits:
    concurrentReaustsPerUsername: 20
  logging:
    application: "info"
    platform: "info"

platformManagement:
ssl:
   trustStoreBase64: ${pm_trustore_b64}
   trustStorePassword: ${pm_trustore_password}
  initData:
    username: ${platform_username}
    password: ${platform_password}
  auth:
    secret: ${UUID_platform}
    pod:
      secret: ${UUID_pod}
  db:
    POSTGRES_HOST: ${server_name}
    POSTGRES_PORT: 5432
    POSTGRES_USER: ${platform_platformmanagement}
    POSTGRES_PASS: ${platformmanagement_password}
    POSTGRES_DB: "platformmanagement"
    POSTGRES_MAX_CONNECTIONS: "20"
    POSTGRES_USE_SSL: "true"
  storage:
    backend: "MINIO"
```

```
      minio:
        bucket: "platform"
        username@servername
        accessKey: ${minioAccessKey}
        secretKey: ${minioSecretKey}
        endpoint: https://${server_name}:9005
        MINIO_SSL_ENABLED: true
    logging:
      application: "info"
      platform: "info"
    quartz:
      active: "true"
    secretAesKeyB64: ${AES_platform}

  connectors:
    auth:
      secret: ${UUID_connectors}
```

Развертывание приложения

Переменные значения:
${release_version} - актуальная версия релиза приложения

```
helm upgrade -f /datadrive/platform/helm/custom-values.yaml --install platform-platform
/datadrive/platform/helm/platform-platform-application/platform-platform-application_${release_v
ersion}.tgz
```

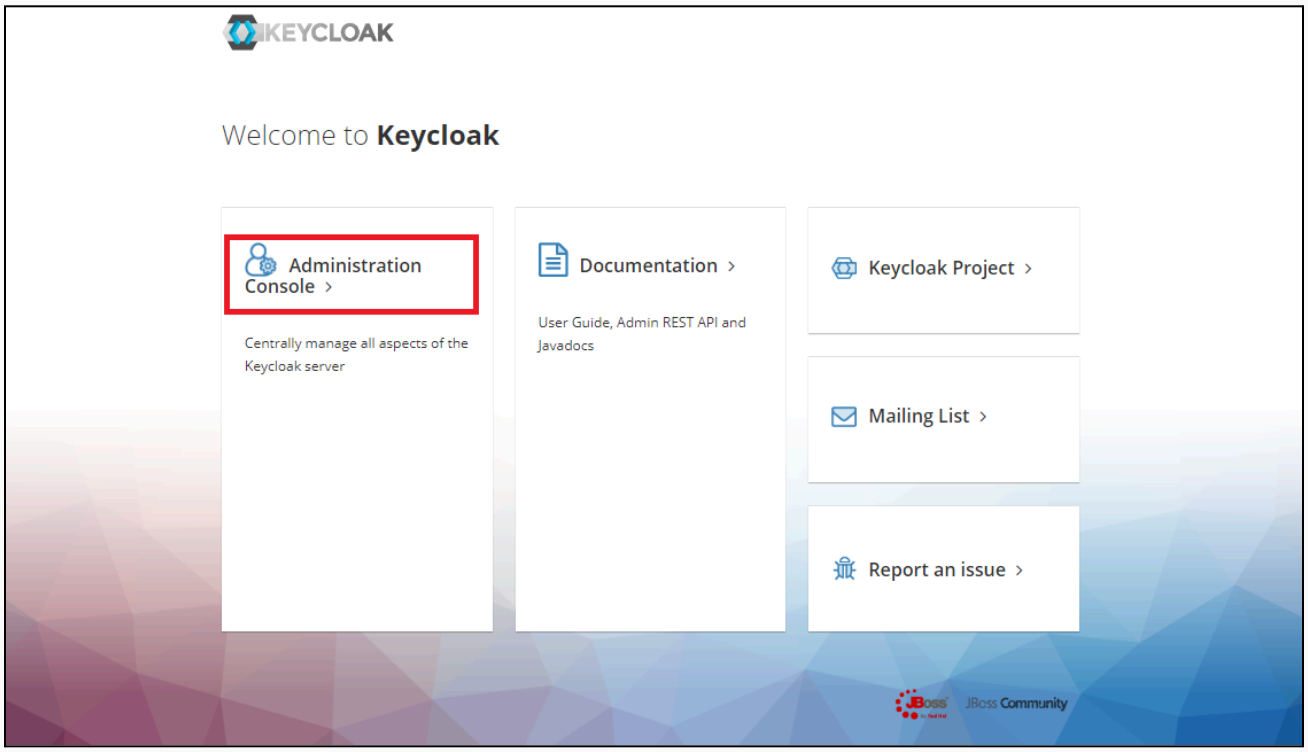
Работоспособность Платформы возможно проверить следующим образом. Все pods должны быть в состоянии Running.

```
kubectl get pods -n platform-platform
NAME                                    READY   STATUS    RESTARTS       AGE
feature-flags-74cc45dd5f-9l69h          1/1     Running   1 (6d1h ago)   16d
process-flow-diagrams-85477bb685-4gqd2  1/1     Running   1 (6d1h ago)   19d
sli-monitoring-78df7bff54-h6hx9         1/1     Running   1 (6d1h ago)   30d
liquibase-unlock-discovery-fbb4f9497-bgxrs  1/1     Running   2 (6d1h ago)   30d
stream-storage3-7bcdcd768b-97jfk        2/2     Running   7 (5d5h ago)   7d6h
incident-management-7b9486d48d-h9rbm    2/2     Running   7 (5d5h ago)   8d
keycloak-7f774cddd9-4h82r               2/2     Running   4 (5d5h ago)   16d
docs-544d9cb4cc-2c7zm                   1/1     Running   0              26h
connector-config-66b9db874f-76q2s       1/1     Running   0              26h
platform-management-9449cf489-dc9tm     2/2     Running   0              3h19m
router-656798b89d-cxxh4                 2/2     Running   0              50s
dashboard-667cf9dcf7-qdksv              1/1     Running   0              51s
```

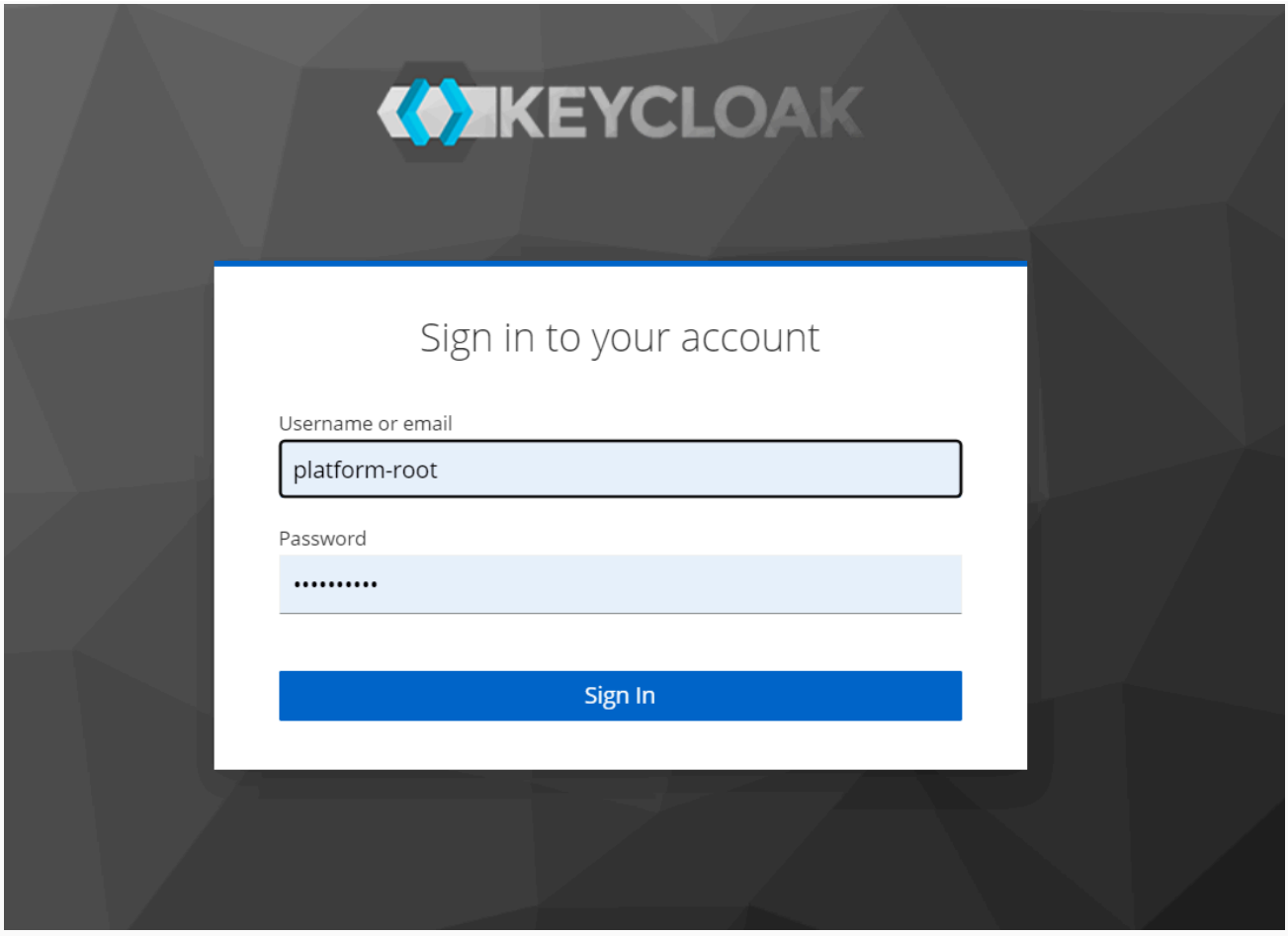
Так же должен быть доступен Web UI по адресу https://${server_name}/ui/login

Создание Пользователя Платформы.

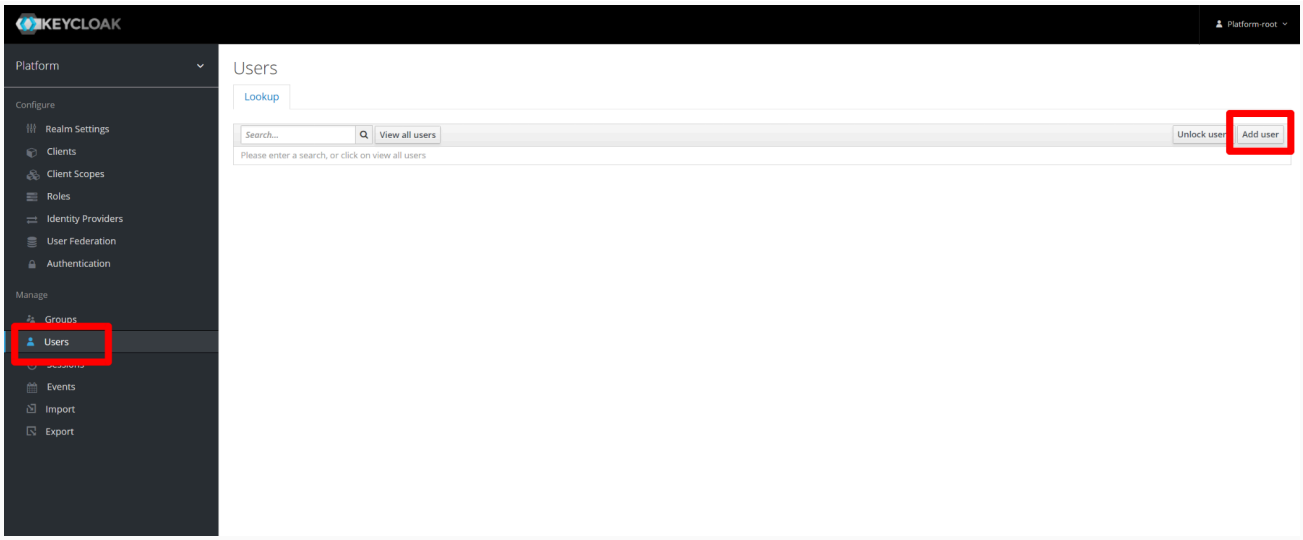
Подключится через WebUI к странице авторизации:
https://${server_name}/auth и перейти в Administration Console



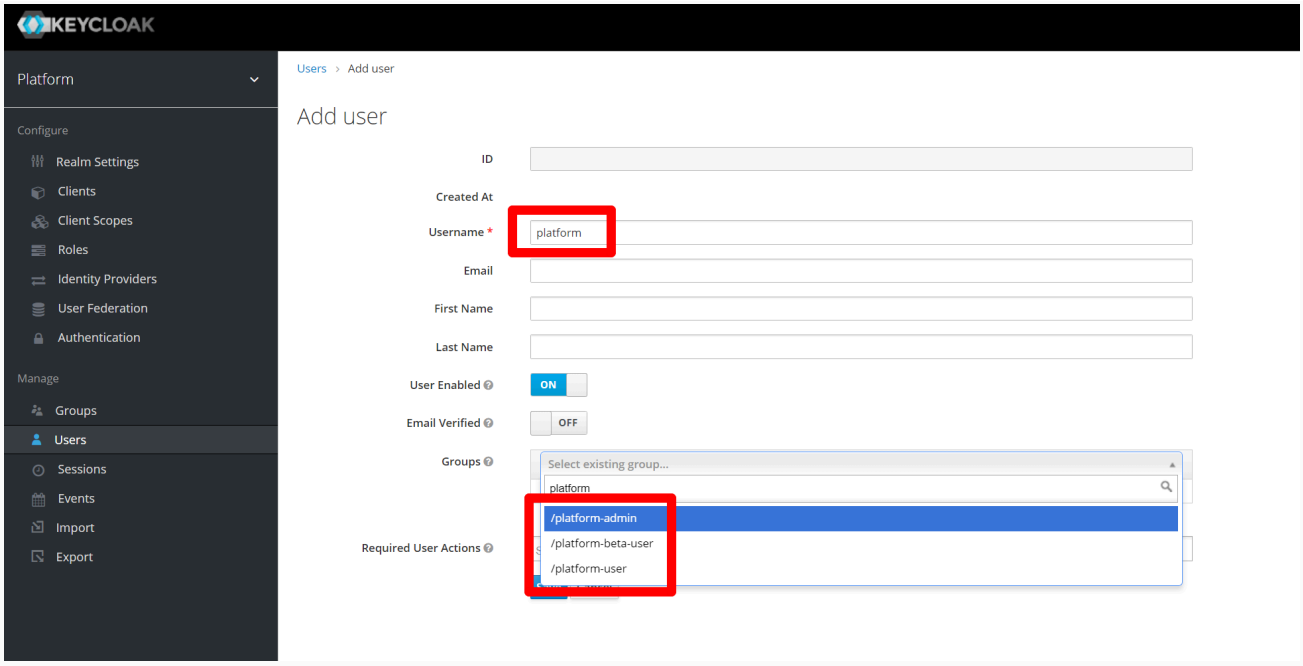
Авторизуйтесь используя имя пользователя platform-root и пароль ${your_password_for_keycloack_UI}, который можно посмотреть в helm-values.yaml



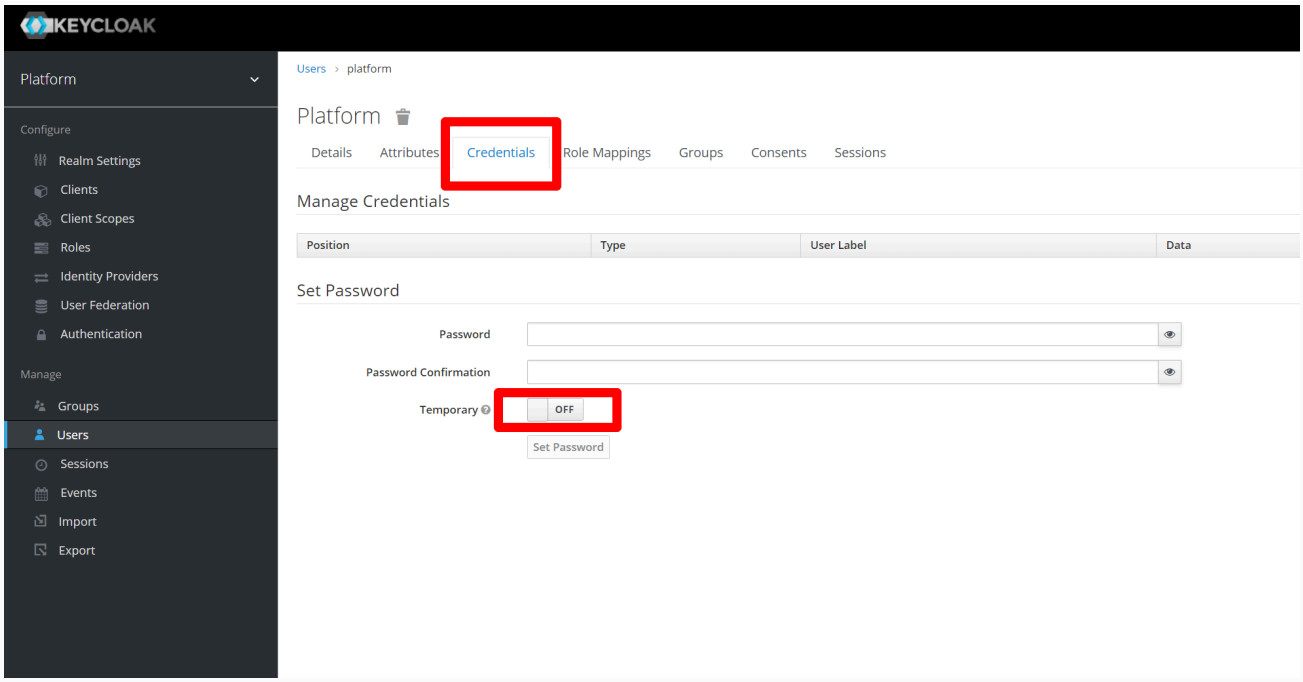
Перейдите во вкладку users на панели слева и затем нажмите Add user справа в открывшейся панели.



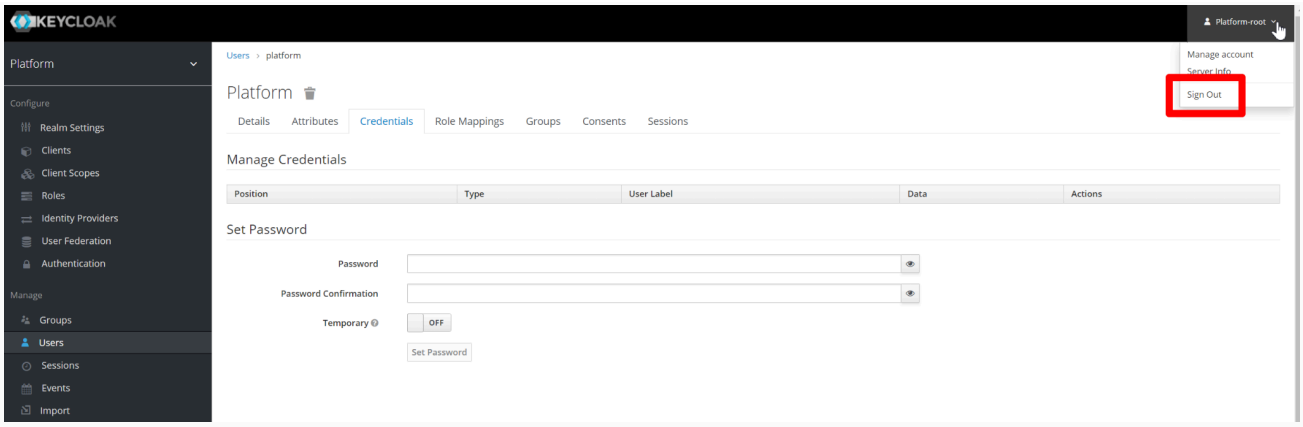
В открывшейся панели введите имя пользователя и назначьте соответствующие Groups, как показано на скриншоте и нажмите Save



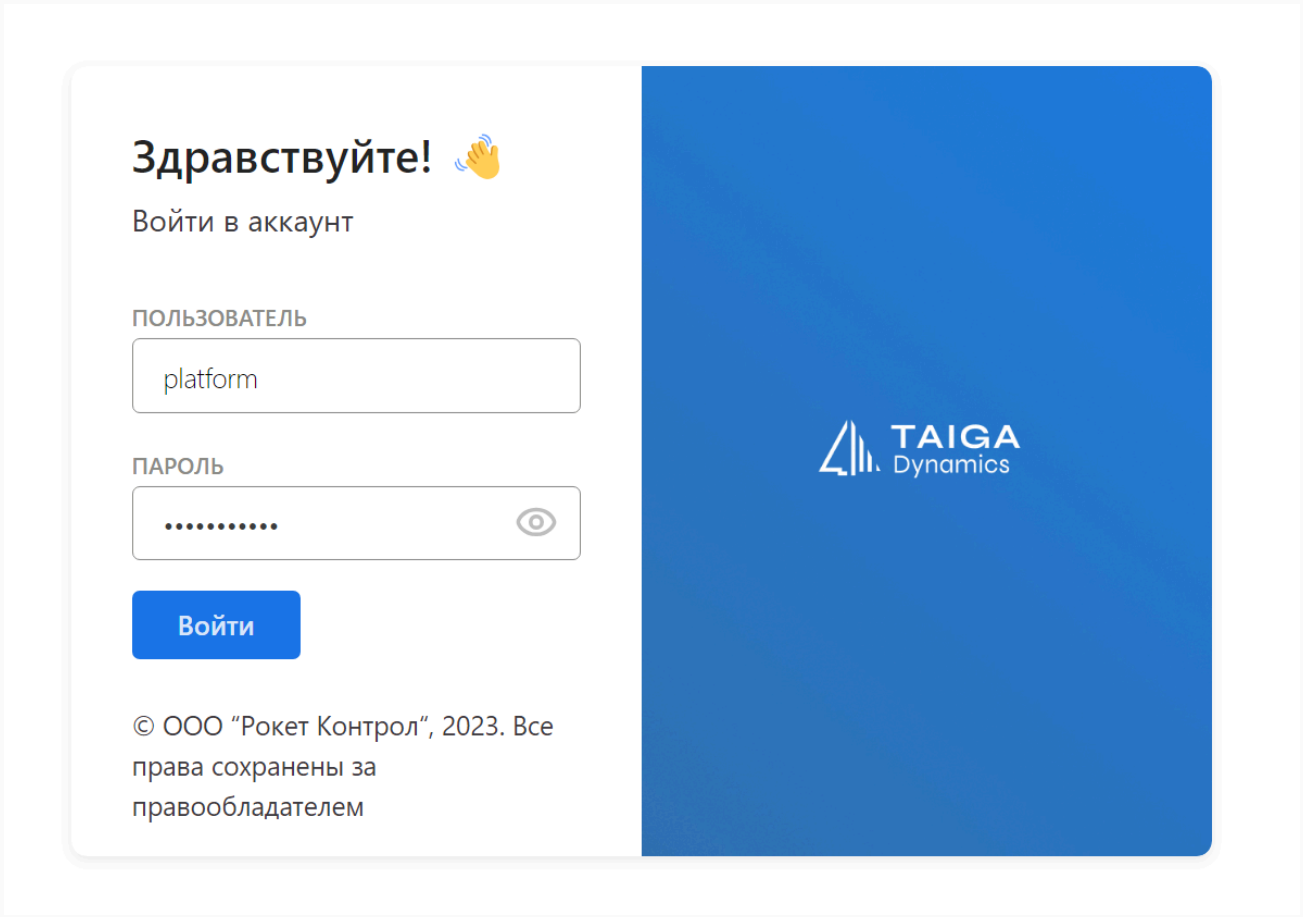
Затем перейдите во вкладку Credentials на верхней панели введите валидный пароль и повторите его, передвиньте toggle Temporary в положение OFF и затем нажмите Set Password



Затем выйдете из страницы авторизации через Sign Out



Залогиньтесь на страницу авторизации платформы https://${server_name}/ui/login , введите имя пользователя и пароль которые были указаны на предыдущем шаге и нажмите войти



В случае удачного логина - установка платформы окончена.

Контакты

Для помощи в установке Программы для ЭВМ «Платформа Тайга Дайнемикс» просьба писать техническим специалистам, используя электронную почту: a.osipov@rocketcontrol.ai, support@rocketcontrol.ai и a.gubarev@rocketcontrol.ai.