ПРОГРАММНЫЙ ПРОДУКТ
ЦИФРОВОЙ ШТАБ

Руководство по развертыванию

2024

АННОТАЦИЯ

Данный документ предназначен для специалистов, выполняющих администрирование программного продукта «Цифровой штаб» (далее – **Программный продукт**), и включает описание действий по установке и настройке **Программного продукта** в операционных системах, поддерживающих систему управления контейнерами Docker (Astra Linux, РЕД ОС, Ubuntu, Debian и т. д.).

СОДЕРЖАНИЕ

1 ТРЕБОВАНИЯ К ПРОГРАММНО-АППАРАТНОЙ ЧАСТИ И ПЕРСОНАЛУ

1.1 Требования к серверной части

Все компоненты **Программного продукта** устанавливаются на один виртуальный либо физический сервер под управлением ОС, поддерживающей систему управления контейнерами Docker.

Минимальные требования к аппаратной части:

- Процессор: не менее 2 ГГц, 22 ядра.
- Оперативная память: не менее 44 Гб.
- Дисковое пространство: не менее 6544 Гб свободного дискового пространства.

Требования к программной части:

- ОС, поддерживающие систему управления контейнерами Docker (Astra Linux, РЕД ОС, Ubuntu, Debian и т. д.).
- СУБД: PostgreSQL версии 15.2.

Список программных комплексов, доступ к которым **Программный продукт** должен иметь монопольный доступ:

- инфраструктурный домен **Программного продукта**;
- платформа виртуализации;
- система учета сетевых конфигураций;
- система управления конфигурациями;
- хранилище секретов.

Под монопольным доступом подразумевается, что никакие другие системы или пользователи не используют указанные системы.

1.2 Требования к квалификации персонала

Администратор **Программного продукта** должен обладать квалификацией, обеспечивающей:

- базовые навыки администрирования ОС семейства Linux (настройка репозиториев, системные настройки);
- базовые навыки работы с Docker, Docker Compose;
- базовые навыки работы с СУБД PostgreSQL.
- базовые навыки работы со средствами автоматизации Ansible;
- базовые навыки работы со средствами виртуализации;
- базовые навыки работы с сетевой инфраструктурой;
- базовые навыки работы со средствами мониторинга ИБ;
- базовые навыки работы со средствами мониторинга ИТ;
- базовые навыки работы со службами каталогов.

2 УСТАНОВКА И РАЗВЕРТЫВАНИЕ ПРОГРАММНОГО ПРОДУКТА

2.1 Установка кластера kubernetes

2.1.1 Настройка мастер нод

Переключаемся на root

```
su -
```

отключаем swap и selinux

```
swapoff -a && sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
setenforce && sed -I 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
```

создайте файл для автозагрузки модулей ядра, необходимых для работы сервиса crio:

```
cat <<EOF> /etc/modules-load.d/crio.conf
overlay
br_netfilter
EOF
```

Загрузите модули в ядро

```
cat <<EOF> /etc/modules-load.d/crio.conf
overlay
br_netfilter
EOF
```

Проверьте, что данные модули работают:

```
lsmod | egrep "br_netfilter|overlay"
```

Создайте конфигурационный файл для работы сети внутри kubernetes:

```
cat <<EOF> /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
```

Примените параметры командой:

```
sysctl --system
```

Установите необходимые пакеты:

```
dnf install kubernetes kubernetes-kubeadm cri-o cri-tools -y
```

Настройте проброс портов в iptables:

```
iptables -P FORWARD ACCEPT
```

Установите настройки по умолчанию для конфигурации контейнера:

```
sed -i '/^\[crio\.runtime\]/a seccomp_profile = "\/etc\/containers\/policy.json"' /etc/crio/crio.conf
```

Запустите службу crio и добавьте ее в автозагрузку:

```
systemctl enable --now crio
```

Добавьте службу kubelet в автозагрузку:

```
systemctl enable kubelet.service
```

Загрузите образы контейнеров, необходимых kubeadm для инициализации ноды кластера:

```
kubeadm config images pull
```

Запустите инициализацию мастер-ноды в одноранговом кластере. Данная команда выполнит начальную настройку и подготовку основного узла кластера. Ключ --pod-network-cidr задает адрес внутренней подсети для вашего кластера.

```
kubeadm init --pod-network-cidr=10.244.0.0/16
```

В случае успешной инициализации, в конце вывода команды будет отображаться примерно следующее:

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 10.81.186.86:6443 --token sj5u6t.m8wl18o26fhj4gml \
 --discovery-token-ca-cert-hash
sha256:53cbbbaf46036dadd55d348eea7b7585d7e7e048a554f7b684a5e5322f9ba498
```

Настройте параметры управления кластером. Настройку можно выполнить как для локального пользователя, так и для суперпользователя root.

Для управления кластером от имени локального пользователя выполните команды:

```
mkdir /home/$USER/.kube
cp -i /etc/kubernetes/admin.conf /home/$USER/.kube/config
chown $USER. /home/$USER/.kube /home/$USER/.kube/config
```

Для управления кластером от имени суперпользователя root выполните команды:

```
echo "export KUBECONFIG=/etc/kubernetes/admin.conf" >> /root/.bashrc
source .bashrc
export KUBECONFIG=/etc/kubernetes/admin.conf
```

Установите внутреннюю конфигурацию сети в кластере (в примере используется calico).

```
kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
```

Проверьте список и статус всех подов в кластере:

```
kubectl get pod -n kube-system
```

Выполните дополнительную настройку:

```
sudo hostnamectl set-hostname masternode
```

Проверьте изменения:

```
hostname && hostname -I
masternode10.81.186.86 10.81.186.67 10.85.0.1 1100:200::1
```

В файл /etc/hosts внесите данные о master и worker:

```
sudo nano /etc/hosts
```

```
10.81.186.86 masternode
10.81.186.106 worker
```

Для вывода команды присоединения worker к кластеру выполните:

```
kubeadm token create --print-join-command

kubeadm join 10.81.186.86:6443 --token nxuiy0.xt5ts9u4ynhdoowr --discovery-token-ca-cert-hash sha256:53cbbbaf46036dadd55d348eea7b7585d7e7e048a554f7b684a5e5322f9ba498
```

2.1.2 Настройка рабочих нод

Переключаемся на root

```
su -
```

отключаем swap и selinux

```
swapoff -a && sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
setenforce && sed -I 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
```

создайте файл для автозагрузки модулей ядра, необходимых для работы сервиса crio:

```
cat <<EOF> /etc/modules-load.d/crio.conf
overlay
br_netfilter
EOF
```

Загрузите модули в ядро

```
cat <<EOF> /etc/modules-load.d/crio.conf
overlay
br_netfilter
EOF
```

Проверьте, что данные модули работают:

```
lsmod | egrep "br_netfilter|overlay"
```

Создайте конфигурационный файл для работы сети внутри kubernetes:

```
cat <<EOF> /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
```

Примените параметры командой:

```
sysctl --system
```

Установите необходимые пакеты:

```
dnf install kubernetes kubernetes-kubeadm cri-o cri-tools -y
```

Настройте проброс портов в iptables:

```
iptables -P FORWARD ACCEPT
```

Установите настройки по умолчанию для конфигурации контейнера:

```
sed -i '/^\[crio\.runtime\]/a seccomp_profile = "\/etc\/containers\/policy.json"' /etc/crio/crio.conf
```

Запустите службу crio и добавьте ее в автозагрузку:

```
systemctl enable --now crio
```

Добавьте службу kubelet в автозагрузку:

```
systemctl enable kubelet.service
```

Выполните дополнительную настройку:

```
sudo hostnamectl set-hostname worker
```

Проверьте изменения:

```
hostname && hostname -I
 worker10.81.186.106 10.81.186.142
```

В файл /etc/hosts внесите данные о master и worker:

```
sudo nano /etc/hosts
```

```
10.81.186.86 masternode
10.81.186.106 worker
```

2.1.3 Подключение рабочей ноды к кластеру

На master-ноде получите команду присоединения worker:

```
kubeadm token create --print-join-command

kubeadm join 10.81.186.86:6443 --token nxuiy0.xt5ts9u4ynhdoowr --discovery-token-ca-cert-hash sha256:53cbbbaf46036dadd55d348eea7b7585d7e7e048a554f7b684a5e5322f9ba498
```

На worker выполните полученную команду:

```
kubeadm join 10.81.186.86:6443 --token nxuiy0.xt5ts9u4ynhdoowr --discovery-token-ca-cert-hash sha256:53cbbbaf46036dadd55d348eea7b7585d7e7e048a554f7b684a5e5322f9ba498

[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

2.2 Установка Kafka

2.2.1 Установка zookeeper

Установите Java в системе:

```
dnf install java-11-openjdk -y
```

Скачайте архив zookeeper и распакуйте

```
sudo wget https://www.apache.org/dist/zookeeper/zookeeper-3.6.2/apache-zookeeper-3.6.2-bin.tar.gz
sudo tar -xvzf apache-zookeeper-3.6.2-bin.tar.gz -C /opt
sudo mv /opt/apache-zookeeper-3.6.2-bin /opt/zookeeper
```

Создайте конфигурационный файл Zookeeper:

```
sudo cp /opt/zookeeper/conf/zoo_sample.cfg /opt/zookeeper/conf/zoo.cfg
```

Добавьте переменные окружения в файл /etc/environment:

```
echo "export ZOOKEEPER_HOME=/opt/zookeeper" | sudo tee -a /etc/environment
echo "export PATH=\$ZOOKEEPER_HOME/bin:\$PATH" | sudo tee -a /etc/environment
```

Откройте порты:

```
sudo firewall-cmd --zone=public --add-port=2181/tcp --add-port=2888/tcp --add-port=3888/tcp –permanent
sudo firewall-cmd --reload
```

Создайте пользователя и смените владельца

```
sudo useradd zookeeper --system  --no-create-home
sudo chown -R zookeeper:zookeeper /opt/zookeeper
```

Создайте юнит для запуска сервиса

```
cat <<EOF> /etc/systemd/system/zookeeper.service
[Unit]
Description=Apache Zookeeper
Documentation=http://zookeeper.apache.org
Requires=network.target
After=network.target

[Service]
Type=forking
EnvironmentFile=/etc/default/zookeeper
ExecStart=/usr/share/zookeeper/bin/zkServer.sh start
ExecStop=/usr/share/zookeeper/bin/zkServer.sh stop
SuccessExitStatus=143
Restart=on-failure
User=zookeeper
Group=zookeeper

[Install]
WantedBy=multi-user.target
Alias=zookeeper.service
EOF
```

Запустите сервис

```
sudo systemctl enable –now zookeeper
```

2.2.2 Установка kafka

Установите Java в системе:

```
dnf install java-11-openjdk -y
```

Скачайте архив zookeeper и распакуйте

```
sudo wget https://archive.apache.org/dist/kafka/3.2.1/kafka_2.13-3.2.1.tgz
sudo tar -xvzf kafka_2.13-3.2.1.tgz -C /opt
sudo mv /opt/kafka_2.13-3.2.1 /opt/kafka
```

Отредактируйте файл конфигурации Kafka config/server.properties, укажите свой broker.id, listeners и advertised.listeners.

```
nano config/server.properties
broker.id=0
listeners=PLAINTEXT://localhost:9092
advertised.listeners=PLAINTEXT://your-hostname:9092
```

Создайте юнит для запуска сервиса

```
cat <<EOF> /etc/systemd/system/kafka.service
[Unit]
Description=Apache Kafka
Documentation=http://kafka.apache.org/documentation.html
Requires=network.target
After=network.target

[Service]
Type=simple
StandardOutput=null
Environment="KAFKA_HEAP_OPTS=-Xms1G -Xmx1G"
Environment="LOG_DIR=/var/log/kafka"
ExecStart=/opt/kafka/bin/kafka-server-start.sh /etc/kafka/server.properties
ExecStop=/opt/kafka/bin/kafka-server-stop.sh
User=kafka
Group=kafka
Restart=on-failure
LimitNOFILE=infinity
SuccessExitStatus=143

[Install]
WantedBy=multi-user.target
Alias=kafka.service
EOF
```

Откройте порты:

```
sudo firewall-cmd --zone=public --add-port=9092/tcp –permanent
sudo firewall-cmd --reload
```

Создайте пользователя и смените владельца

```
sudo useradd kafka --system  --no-create-home
sudo chown -R kafka:kafka /opt/kafka
```

Запустите сервис

```
sudo systemctl enable –now kafka
```

2.3 Установка Schema Registry

Установите Java в системе:

```
dnf install java-11-openjdk -y
```

Скачайте архив schema_registry и распакуйте

```
sudo wget https://packages.confluent.io/archive/6.0/confluent-6.0.1.tar.gz
sudo tar -xvzf confluent-6.0.1.tar.gz -C /opt
```

Отредактируйте /etc/schema-registry/log4j.properties

```
log4j.rootLogger=INFO, stdout, file

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%d] %p %m (%c:%L)%n

log4j.logger.kafka=ERROR, stdout
log4j.logger.org.apache.zookeeper=ERROR, stdout
log4j.logger.org.apache.kafka=ERROR, stdout
log4j.additivity.kafka.server=false

log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.maxBackupIndex=10
log4j.appender.file.maxFileSize=100MB
log4j.appender.file.File=${schema-registry.log.dir}/schema-registry.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=[%d] %p %m (%c)%n
```

Отредактируйте /etc/schema-registry/schema-registry.properties

```
listeners= http://0.0.0.0:8081

kafkastore.bootstrap.servers=PLAINTEXT://<HOST-WITH-KAFKA>:9092
kafkastore.topic=_schemas
debug=False
schema.compatibility.level=full_transitive
```

Создайте юнит для запуска сервиса

```
cat <<EOF> /etc/systemd/system/schema_registry.service
[Unit]
Description=schema registry
After=network.target

[Service]
PrivateTmp=True
User=schema_registry
Group=schema_registry
LimitNOFILE=32768
Environment='SCHEMA_REGISTRY_LOG4J_OPTS=-Dlog4j.configuration=file:/etc/schema-registry/log4j.properties -Dschema-registry.log.dir=/var/log/schema_registry'
Environment='SCHEMA_REGISTRY_HEAP_OPTS=-Xmx256M'
Environment='SCHEMA_REGISTRY_JVM_PERFORMANCE_OPTS=-server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:+ExplicitGCInvokesConcurrent -Djava.awt.headless=true'
Environment='SCHEMA_REGISTRY_JMX_OPTS=-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false'
Environment='JMX_PORT=5555'


ExecStart=/opt/confluent-6.0.1/bin/schema-registry-start /etc/schema-registry/schema-registry.properties
ExecStop=/opt/confluent-6.0.1/bin/schema-registry-stop

ExecReload=/bin/kill -HUP $MAINPID
KillSignal=SIGTERM
Restart=on-failure

[Install]
WantedBy=multi-user.target
EOF
```

Откройте порты:

```
sudo firewall-cmd --zone=public --add-port=8081/tcp –permanent
sudo firewall-cmd --reload
```

Создайте пользователя и смените владельца

```
sudo useradd schema_registry --system  --no-create-home
sudo chown -R schema_registry:schema_registry  /opt/confluent*
```

Запустите сервис

```
sudo systemctl enable –now schema_registry
```

2.4 Установка Postgresql

Переключаемся на root

```
su -
```

Для установки postgresql выполните команду:

```
dnf install postgresql15-server
```

Далее необходимо произвести инициализацию базы данных postgresql:

```
postgresql-15-setup initdb
```

После успешной инициализации запустите службу postgresql и добавьте ее в автозагрузку:

```
systemctl enable postgresql-15.service --now
```

Убедитесь, что служба запущена:

```
systemctl status postgresql-15.service
```

В статусе должно отображаться active (running).

2.5 Установка Mongodb

Установите mongodb:

```
sudo dnf install mongodb-org-server-6.0* -y
```

Создайте каталог для данных и установите владельцем mongod:

```
sudo mkdir -p /data/db
sudo chown -R mongod:mongod /data/db
```

Конфигурационный файл mongodb:

```
sudo cat <<EOF> /etc/mongod.conf
net:
  bindIp: 0.0.0.0
  ipv6: false
  maxIncomingConnections: 65536
  port: 27017

processManagement:
  timeZoneInfo: /usr/share/zoneinfo

systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

storage:
  dbPath: /data/db
  journal:
    enabled: true

security:
  authorization: enabled
EOF
```

Создайте пользователей СУБД.
Переключитесь на пользователя mongod

```
su - mongod
```

Запустите mongod без контроля доступа

```
mongod --port 27017 --dbpath /data/db
```

Подключаемся к инстансу и создаём пользователя

```
mongosh --port 27017
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: passwordPrompt(), // or cleartext password
    roles: [
      { role: "userAdminAnyDatabase", db: "admin" },
      { role: "readWriteAnyDatabase", db: "admin" }
    ]
  }
)
```

Завершите сеанс, остановите процесс mongod, а затем добавьте mongod в автозагрузку

```
sudo systemctl enable --now mongod
```

2.6 Установка Keycloak

Установите Java в системе:

```
sudo dnf install java-11-openjdk -y
```

Создайте системного пользователя keycloak

```
sudo useradd keycloak --system  --no-create-home
```

Скачайте и распакуйте архив с keycloak

```
curl -L https://github.com/keycloak/keycloak/releases/download/19.0.1/keycloak-19.0.1.tar.gz -o /opt/keycloak
```

Необходимо убедиться, что создан каталог для данных, логов и ssl сертификата

```
sudo mkdir -p /opt/keycloak/keycloak-10.0.1/{data,log}
```

Заполните необходимые параметры в конфигурационном файле

```
sudo cat <<EOF> /opt/keycloak/keycloak-19.0.1/conf/keycloak.conf
db-username=<db_user>
db-password=<db_passr>
db=postgres
health-enabled=True
metrics-enabled=True
proxy=edge
db-url=jdbc:postgresql://<db_ip>:<db_port>/<db_name>
EOF
```

Смените владельца на keycloak

```
sudo chown -R keycloak:keycloak /opt/keycloak
```

Создайте системный юнит для запуска keycloak

```
sudo cat <<EOF> /etc/systemd/system/keycloak.service
[Unit]
Description=Keycloak server
After=network-online.target
Wants=network-online.target systemd-networkd-wait-online.service

[Service]
User=keycloak
Group=keycloak
Environment="KEYCLOAK_ADMIN=<username>"
Environment="KEYCLOAK_ADMIN_PASSWORD=<password_for_username>"
Environment="KC_HTTP_RELATIVE_PATH=/auth"
Environment="PROXY_ADDRESS_FORWARDING=true"
KC_HOSTNAME="somedomain.ru"
ExecStart=/opt/keycloak/keycloak-19.0.1/bin/kc.sh start  --http-enabled=true \
            --http-port=8080 --hostname-strict=false --hostname-strict-https=false \
            --log=console,file --log-file=/opt/keycloak/log/server.log
WorkingDirectory=/opt/keycloak/keycloak-19.0.1
ReadWritePaths=/opt/keycloak/keycloak-19.0.1/conf /opt/keycloak/keycloak-19.0.1/data
/opt/keycloak/keycloak-19.0.1/lib/quarkus
SuccessExitStatus=0 143

# Hardening options
CapabilityBoundingSet=
AmbientCapabilities=
NoNewPrivileges=true
ProtectHome=true
ProtectSystem=strict
ProtectKernelTunables=true
ProtectKernelModules=true
ProtectControlGroups=true
PrivateTmp=true
PrivateDevices=true
LockPersonality=true

[Install]
WantedBy=multi-user.target
EOF
```

Добавьте в автозагрузку сервис

```
Sudo systemctl daemon-reload
sudo ssytemctl enable --now keycloak
```

2.7 Установка Minio

Создайте пользователя minio

```
sudo useradd minio --system  --no-create-home
```

Скачайте и установите пакет minio

```
go_arch=amd64
# RELEASE.2023-03-24T21-41-23Z
release=20230324214123.0.0.x86_64
curl -L https://dl.minio.io/server/minio/release/linux-amd64/archive/minio-
\${minio_server_release}.rpm \
-o /tmp/minio-\${minio_server_release}.rpm
rpm -iU /tmp/minio-\${minio_server_release}.rpm
```

Создайте каталог для данных

```
mkdir -p /var/lib/minio
chown minio:minio -R /var/lib/minio
```

Отредактируйте файл /etc/default/minio

```
cat <<EOF> /etc/default/minio
# Minio local/remote volumes.
MINIO_VOLUMES="/var/lib/minio"
# Minio cli options.
MINIO_OPTS="--address :9000 --console-address :9001 "

# Access Key of the server.
MINIO_ACCESS_KEY="root"
# Secret key of the server.
MINIO_SECRET_KEY="password"

MINIO_DOMAIN=minio.domain.ru
MINIO_SERVER_URL=https://minio.domain.ru
MINIO_BROWSER_REDIRECT_URL=https://minio.domain.ru/console
CONSOLE_SUBPATH=/console
MINIO_ROOT_USER=root
MINIO_ROOT_PASSWORD=P@ssw0rd
EOF
```

Создайте системный юнит minio

```
cat <<EOF> /etc/systemd/system/minio.service
[Unit]
Description=MinIO
Documentation=https://docs.min.io
Wants=network-online.target
After=network-online.target
AssertFileIsExecutable=/usr/local/bin/minio

[Service]
WorkingDirectory=/usr/local

User=minio
Group=minio
ProtectProc=invisible

EnvironmentFile=-/etc/default/minio
ExecStartPre=/bin/bash -c "if [ -z \"\${MINIO_VOLUMES}\" ]; then echo \"Variable
MINIO_VOLUMES not set in /etc/default/minio\"; exit 1; fi"
ExecStart=/usr/local/bin/minio server \$MINIO_OPTS $MINIO_VOLUMES

# Let systemd restart this service always
Restart=always

# Specifies the maximum file descriptor number that can be opened by this process
LimitNOFILE=1048576

# Specifies the maximum number of threads this process can create
TasksMax=infinity

# Disable timeout logic and wait until process is stopped
TimeoutStopSec=infinity
SendSIGKILL=no

[Install]
WantedBy=multi-user.target
EOF
```

Запустите сервис

```
systemctl daemon-reload
systemctl enable --now minio
```

2.8 Установка mediamtx

Создайте пользователя mediamtx

```
sudo useradd mediamtx --system  --no-create-home
```

Скачайте архив с mediamtx и распакуйте

```
cd /tmp

mkdir -p /etc/mediamtx/
#useradd mediamtx --system  --no-create-home -g mediamtx
useradd mediamtx --system  --no-create-home
curl -L
https://github.com/bluenviron/mediamtx/releases/download/v0.22.0/mediamtx_v0.22.0_linux_amd64.tar.gz \
-o /tmp/mediamtx_v0.22.0_linux_amd64.tar.gz
tar -xzvf mediamtx_v0.22.0_linux_amd64.tar.gz
mv mediamtx /usr/local/bin/
chown -R mediamtx:mediamtx /usr/local/bin/mediamtx
```

Отредактируйте файл /etc/mediamtx/mediamtx.yml

```
cat <<EOF> /etc/mediamtx/mediamtx.yml
logLevel: info
logDestinations: [syslog]
logFile: mediamtx.log

readTimeout: 10s
writeTimeout: 10s
readBufferCount: 512
udpMaxPayloadSize: 1472

externalAuthenticationURL:

api: yes
apiAddress: 0.0.0.0:9997

metrics: yes
metricsAddress: 0.0.0.0:9998

pprof: no
pprofAddress: 127.0.0.1:9999

runOnConnect:
runOnConnectRestart: no


rtspDisable: no
protocols: [udp, multicast, tcp]
encryption: "no"
rtspAddress: :8554
rtspsAddress: :8322
rtpAddress: :8000
rtcpAddress: :8001
multicastIPRange: 224.1.0.0/16
multicastRTPPort: 8002
multicastRTCPPort: 8003
serverKey: server.key
serverCert: server.crt
authMethods: [basic, digest]


rtmpDisable: no
rtmpAddress: :1935
rtmpEncryption: "no"
rtmpsAddress: :1936
rtmpServerKey: server.key
rtmpServerCert: server.crt


hlsDisable: no
hlsAddress: :8888
hlsEncryption: no
hlsServerKey: server.key
hlsServerCert: server.crt
hlsAlwaysRemux: no
hlsVariant: lowLatency
hlsSegmentCount: 7
hlsSegmentDuration: 1s
hlsPartDuration: 200ms
hlsSegmentMaxSize: 50M
hlsAllowOrigin: '*'
hlsTrustedProxies: []
hlsDirectory: ''


webrtcDisable: no
webrtcAddress: :8889
```

```
webrtcEncryption: no
webrtcServerKey: server.key
webrtcServerCert: server.crt
webrtcAllowOrigin: '*'
webrtcTrustedProxies: []
webrtcICEServers: [stun:stun.l.google.com:19302]
webrtcICEHostNAT1To1IPs: []
webrtcICEUDPMuxAddress:
webrtcICETCPMuxAddress:


paths:
  all:
    source: publisher

    sourceProtocol: automatic

    sourceAnyPortEnable: no

    sourceFingerprint:

    sourceOnDemand: no
    sourceOnDemandStartTimeout: 10s
    sourceOnDemandCloseAfter: 10s

    sourceRedirect:

    disablePublisherOverride: no

    fallback:

    rpiCameraCamID: 0
    rpiCameraWidth: 1920
    rpiCameraHeight: 1080
    rpiCameraHFlip: false
    rpiCameraVFlip: false
    rpiCameraBrightness: 0
    rpiCameraContrast: 1
    rpiCameraSaturation: 1
    rpiCameraSharpness: 1
    rpiCameraExposure: normal
    rpiCameraAWB: auto
    rpiCameraDenoise: "off"
    rpiCameraShutter: 0
    rpiCameraMetering: centre
    rpiCameraGain: 0
    rpiCameraEV: 0
    rpiCameraROI:
    rpiCameraTuningFile:
    rpiCameraMode:
    rpiCameraFPS: 30
    rpiCameraIDRPeriod: 60
    rpiCameraBitrate: 1000000
    rpiCameraProfile: main
    rpiCameraLevel: '4.1'
    rpiCameraAfMode: auto
    rpiCameraAfRange: normal
    rpiCameraAfSpeed: normal
    rpiCameraLensPosition: 0.0
    rpiCameraAfWindow:
    rpiCameraTextOverlayEnable: false
    rpiCameraTextOverlay: '%Y-%m-%d %H:%M:%S - MediaMTX'

    publishUser: video-processing-ms
    publishPass: PASSWORD
    publishIPs: []
```

```
133:
134:      readUser: video-processing-ms
135:      readPass: PASSWORD
136:      readIPs: []
137:
138:      runOnInit:
139:      runOnInitRestart: no
140:
141:      runOnDemand:
142:      runOnDemandRestart: no
143:      runOnDemandStartTimeout: 10s
144:      runOnDemandCloseAfter: 10s
145:
146:      runOnReady:
147:      runOnReadyRestart: no
148:
149:      runOnRead:
150:      runOnReadRestart: no
151:  EOF
```

Укажем владельца

```
1:  sudo chown -R mediamtx:mediamtx /etc/mediamtx/
```

Создайте системный юнит mediamtx

```
1:  cat <<EOF> /etc/systemd/system/mediamtx.service
2:  [Unit]
3:  Description=mediamtx
4:  After=network.target
5:
6:  [Service]
7:  User=mediamtx
8:  Group=mediamtx
9:  Type=simple
10: ExecStart=/usr/local/bin/mediamtx /etc/mediamtx/mediamtx.yml
11:
12: [Install]
13: WantedBy=multi-user.target
14: EOFЗапустите сервис
15: systemctl daemon-reload
16: systemctl enable --now minio
```

Запустите mediamtx

```
1:  sudo systemctl daemon-reload
2:  sudo systemctl enable mediamtx --now
```

2.9 Установка сервисов Цифрового Штаба

2.9.1 Установка реестра образов и загрузка образов

Создайте пользователя registry

```
sudo useradd registry --system  --no-create-home
```

Создайте каталог для данных /var/lib/registry

```
sudo mkdir -p /var/lib/registry
sudo chown -R registry:registry /var/lib/registry
```

Создайте каталог для конфигурационных файлов

```
sudo mkdir -p /etc/registry/
sudo cat <<EOF> /etc/registry/config.yml
version: 0.1
log:
  fields:
    service: registry
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /var/lib/registry
http:
  addr: :5000
  debug:
    addr: localhost:5001
    prometheus:
      enabled: true
      path: /metrics   headers:
    X-Content-Type-Options: [nosniff]
health:
  storagedriver:
    enabled: true
    interval: 10s
    threshold: 3
EOF
sudo chown -R registry:registry /etc/registry/
```

Создайте системный юнит для сервиса registry

```
sudo cat << EOF > /etc/systemd/system/registry.service
[Unit]
Description=registry.service

[Service]
User=registry
Group=registry
Restart=on-failure
ExecStartPre=-/usr/bin/docker rm -f registry
ExecStart=/usr/bin/docker run --name registry \
                              -p 5000:5000 -v /var/lib/registry:/var/lib/registry \
                              --restart=always registry:2
ExecStop=/usr/bin/docker stop -t 10 registry
ExecStopPost=-/usr/bin/docker rm -f registry
ExecReload=/usr/bin/docker restart registry

[Install]
WantedBy=multi-user.target
EOF
```

Добавьте в автозагрузку

```
sudo systemctl daemon-reload
sudo systemctl enable --now registry
```

Установите nginx

```
sudo dnf install nginx -y
```

Создайте каталог для сертификата и скопируйте обе части

```
sudo mkdir -p /etc/nginx/ssl
sudo chown -R nginx:nginx /etc/nginx
```

Добавьте конфигурационный файл

```
sudo cat <<EOF> /etc/nginx/conf.d/registry.conf
server {
        listen 443 ssl;
        server_name registry.somedomain.ru;

        # SSL
        ssl_certificate /etc/nginx/ssl/fullchain.pem;
        ssl_certificate_key /etc/nginx/ssl/privkey.pem;

        # Recommendations from https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html
        ssl_protocols TLSv1.1 TLSv1.2;
        ssl_ciphers 'EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH';
        ssl_prefer_server_ciphers on;
        ssl_session_cache shared:SSL:10m;

        # disable any limits to avoid HTTP 413 for large image uploads
        client_max_body_size 0;

        # required to avoid HTTP 411: see Issue #1486 (https://github.com/moby/moby/issues/1486)
        chunked_transfer_encoding on;

    location /v2/ {
        # Do not allow connections from docker 1.5 and earlier
        # docker pre-1.6.0 did not properly set the user agent on ping, catch "Go *" user agents
        if (\$http_user_agent ~ "^(docker\/1\.(3|4|5(?!\.[0-9]-dev))|Go ).*\$" ) {
          return 404;
        }

        # To add basic authentication to v2 use auth_basic setting.
        # auth_basic "Registry realm";
        # auth_basic_user_file /etc/nginx/conf.d/nginx.htpasswd;

        ## If \$docker_distribution_api_version is empty, the header is not added.
        ## See the map directive above where this variable is defined.
        add_header 'Docker-Distribution-Api-Version' \$docker_distribution_api_version always;
        proxy_pass                          http://localhost:5000;
        proxy_set_header  Host              \$http_host;   # required for docker client's sake
        proxy_set_header  X-Real-IP         \$remote_addr; # pass on real client's IP
        proxy_set_header  X-Forwarded-For   \$proxy_add_x_forwarded_for;
        proxy_set_header  X-Forwarded-Proto \$scheme;
        proxy_read_timeout                  900;
  }
}
EOF
sudo chown nginx:nginx -R /etc/nginx
```

Добавьте nginx в автозагрузку

```
sudo systemctl daemon-reload
sudo systemctl enable --now nginx
```

Скопируйте предоставленные архивы с образами в каталог /var/tmp и загрузите в реестр образов

```
 cd /var/tmp
ls | xargs -I {} docker load -i {}
```

2.9.2 Сервис dhq-backend-for-frontend-ms

Configmap

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  APPLICATION_PORT: "8080"
  BPM_URL: http://dhq-bpm-ms:8080
  CORS_ALLOWED_ORIGINS: 'https://somedomain.ru'
  DATA_GATEWAY_URL: http://dhq-data-gateway-ms:8080
  EVENT_PROCESSING_URL: http://dhq-event-processing-ms:8080
  KEYCLOAK_CLIENT_SECRET: <KEYCLOAK_CLIENT_SECRET>
  KEYCLOAK_REALM: DHQ
  KEYCLOAK_RESOURCE: my-client
  KEYCLOAK_SERVER: https://somedomain.ru/auth
  LOGGING_LEVEL: INFO
  ORGANIZATION_ID: <ORGANIZATION_ID>
  TZ: Europe/Moscow
  VIDEO_PROCESSING_URL: http://dhq-video-processing-ms:8080
kind: ConfigMap
metadata:
  labels:
    project: dhq
  name: dhq-backend-for-frontend-ms-configmap-env
EOF
```

Ingress

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    meta.helm.sh/release-name: dhq-dev
    meta.helm.sh/release-namespace: dhq-dev
    nginx.ingress.kubernetes.io/cors-allow-credentials: "true"
    nginx.ingress.kubernetes.io/cors-allow-headers: Cache-Control, Content-Disposition,
      Content-Length, Content-Type, Date, Expires, Pragma, Strict-Transport-Security,
      Vary, X-Content-Type-Options, X-Frame-Options, X-Xss-Protection, Accept, Accept-
Encoding,
      Accept-Language, Authorization, Cache-Control, Cookie, Referer, Sec-Ch-Ua, Sec-Ch-
Ua-Mobile,
      Sec-Ch-Ua-Platform, Sec-Fetch-Dest, Sec-Fetch-Mode, Sec-Fetch-Site, User-Agent,
      X-Timezone
    nginx.ingress.kubernetes.io/cors-allow-methods: PATCH, PUT, GET, POST, OPTIONS,
      DELETE
    nginx.ingress.kubernetes.io/cors-allow-origin: https://somedomain.ru
    nginx.ingress.kubernetes.io/enable-cors: "true"
    nginx.ingress.kubernetes.io/proxy-body-size: 16m
    nginx.ingress.kubernetes.io/proxy-read-timeout: "1800"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "1800"
    nginx.org/websocket-services: dhq-backend-for-frontend-ms
  labels:
    app: dhq-backend-for-frontend-ms
    project: dhq
  name: dhq-backend-for-frontend-ms
spec:
  ingressClassName: nginx
  rules:
  - host: somedomain.ru
    http:
      paths:
      - backend:
          service:
            name: dhq-backend-for-frontend-ms
            port:
              number: 8080
        path: /api/(?!pass-requests)
        pathType: Prefix
      - backend:
          service:
            name: dhq-backend-for-frontend-ms
            port:
              number: 8080
        path: /[\d]/.*
        pathType: Prefix
      - backend:
          service:
            name: dhq-backend-for-frontend-ms
            port:
              number: 8080
        path: /info.*
        pathType: Prefix
  tls:
  - hosts:
    - somedomain.ru
    secretName: somedomain
EOF
```

Serviceaccount

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-backend-for-frontend-ms
    project: dhq
  name: dhq-backend-for-frontend-ms
EOF
```

Deployment

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-backend-for-frontend-ms
    project: dhq
  name: dhq-backend-for-frontend-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dhq-backend-for-frontend-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-backend-for-frontend-ms
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-backend-for-frontend-ms-configmap-env
        image: dhq-backend-for-frontend-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-backend-for-frontend-ms
        ports:
        - containerPort: 8080
          protocol: TCP
        resources:
          limits:
            cpu: 150m
            memory: 500Mi
          requests:
            cpu: 50m
            memory: 400Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: regcred
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-backend-for-frontend-ms
      serviceAccountName: dhq-backend-for-frontend-ms
      terminationGracePeriodSeconds: 30
      volumes:
      - configMap:
          defaultMode: 420
          name: dhq-backend-for-frontend-ms-configmap
        name: dhq-backend-for-frontend-ms-configmap
EOF
```

Service

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-backend-for-frontend-ms
    project: dhq
  name: dhq-backend-for-frontend-ms
  namespace: dhq-dev
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
    protocol: TCP
  selector:
    app: dhq-backend-for-frontend-ms
  type: ClusterIP
EOF
```

2.9.3 Сервис dhq-bastion-adapter-ms

Serviceaccount dhq-bastion-adapter-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-bastion-adapter-ms
    project: dhq
  name: dhq-bastion-adapter-ms
EOF
```

Configmap dhq-bastion-adapter-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  APP_PORT: "8080"
  BASTION_OPC_SERVER_URL: <BASTION_OPC_SERVER_URL>
  KAFKA_BOOTSTRAP_SERVERS: <kafka-host>:9092
  KAFKA_SCHEMA_REGISTRY_URL: http://<schema-registry-host>:8081
  SPLIT_SIZE: "20"
  TZ: Europe/Moscow
kind: ConfigMap
metadata:
  labels:
    app: dhq-bastion-adapter-ms
    project: dhq
  name: dhq-bastion-adapter-ms-configmap-env
EOF
```

Deploy dhq-bastion-adapter-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-bastion-adapter-ms
    project: dhq
  name: dhq-bastion-adapter-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dhq-bastion-adapter-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
      labels:
        app: dhq-bastion-adapter-ms
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-bastion-adapter-ms-configmap-env
        image: dhq-bastion-adapter-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-bastion-adapter-ms
        ports:
        - containerPort: 8080
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-bastion-adapter-ms
      serviceAccountName: dhq-bastion-adapter-ms
EOF
```

Service dhq-bastion-adapter-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-bastion-adapter-ms
    project: dhq
  name: dhq-bastion-adapter-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-bastion-adapter-ms
  type: ClusterIP
EOF
```

2.9.4 Сервис dhq-bpm-ms

Serviceaccount dhq-bpm-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-bpm-ms
    project: dhq
  name: dhq-bpm-ms
EOF
```

Configmap dhq-bpm-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  ACTIVITI_HISTORY_LEVEL: full
  APPLICATION_PORT: "8080"
  DATA_GATEWAY_URL: http://dhq-data-gateway-ms:8080
  DB_CONNECTION: pgsql
  DB_HOST: <DB_HOST>
  DB_NAME: <DB_NAME>
  DB_PASSWORD: <DB_PASSWORD>
  DB_PORT: <DB_PORT>
  DB_USER: <DB_USER>
  EVENT_PROCESSING_URL: http://dhq-event-processing-ms:8080
  LOGGING_LEVEL: INFO
  PROCESSES_FOLDER_NAME: classpath:processes/***/**/*.bpmn
  TZ: Europe/Moscow
kind: ConfigMap
metadata:
  labels:
    app: dhq-bpm-ms
    project: dhq
  name: dhq-bpm-ms-configmap-env
EOF
```

Deployment dhq-bpm-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-bpm-ms
    project: dhq
  name: dhq-bpm-ms
spec:
  progressDeadlineSeconds: 600
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dhq-bpm-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-bpm-ms
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-bpm-ms-configmap-env
        image: dhq-bpm-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-bpm-ms
        ports:
        - containerPort: 8080
          protocol: TCP
        resources:
          limits:
            cpu: 150m
            memory: 500Mi
          requests:
            cpu: 50m
            memory: 400Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      initContainers:
      - args:
        - -c
        - until pg_isready -h $DB_HOST -p $DB_PORT; do echo waiting for database;
          sleep 2; done;
        command:
        - /bin/sh
        envFrom:
        - configMapRef:
            name: dhq-bpm-ms-configmap-env
        image: postgres:11.15
        imagePullPolicy: IfNotPresent
        name: check-db-ready
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-bpm-ms
      serviceAccountName: dhq-bpm-ms
```

```
EOF
```

Service dhq-bpm-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-bpm-ms
    project: dhq
  name: dhq-bpm-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-bpm-ms
  type: ClusterIP
EOF
```

2.9.5 Сервис dhq-data-gateway-ms

Serviceaccount dhq-data-gateway-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-data-gateway-ms
    project: dhq
  name: dhq-data-gateway-ms
EOF
```

Configmap dhq-data-gateway-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  TZ: Europe/Moscow
kind: ConfigMap
metadata:
  labels:
    app: dhq-data-gateway-ms
    project: dhq
  name: dhq-data-gateway-ms-configmap-env
EOF
```

Deployment dhq-data-gateway-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-data-gateway-ms
    project: dhq
  name: dhq-data-gateway-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dhq-data-gateway-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-data-gateway-ms
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-data-gateway-ms-configmap-env
        image: dhq-data-gateway-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-data-gateway-ms
        ports:
        - containerPort: 8080
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-data-gateway-ms
      serviceAccountName: dhq-data-gateway-ms
EOF
```

Service dhq-data-gateway-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-data-gateway-ms
    project: dhq
  name: dhq-data-gateway-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-data-gateway-ms
  type: ClusterIP
EOF
```

2.9.6 Сервис dhq-dictionaries-update-ms

Serviceaccount dhq-dictionaries-update-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-dictionaries-update-ms
    project: dhq
  name: dhq-dictionaries-update-ms
EOF
```

Configmap dhq-dictionaries-update-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  APPLICATION_PORT: "8080"
  DATA_GATEWAY_URL: http://dhq-data-gateway-ms:8080
  KAFKA_BOOTSTRAP_SERVERS: <kafka-host>:9092
  KAFKA_SCHEMA_REGISTRY_URL: http://<schema-registry-host>:8081
  TZ: Europe/Moscow
kind: ConfigMap
metadata:
  labels:
    app: dhq-dictionaries-update-ms
    project: dhq
  name: dhq-dictionaries-update-ms-configmap-env
EOF
```

Deployment dhq-dictionaries-update-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-dictionaries-update-ms
    project: dhq
  name: dhq-dictionaries-update-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dhq-dictionaries-update-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-dictionaries-update-ms
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-dictionaries-update-ms-configmap-env
        image: dhq-dictionaries-update-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-dictionaries-update-ms
        ports:
        - containerPort: 8080
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-dictionaries-update-ms
      serviceAccountName: dhq-dictionaries-update-ms
      terminationGracePeriodSeconds: 30
EOF
```

Service dhq-dictionaries-update-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-dictionaries-update-ms
    project: dhq
  name: dhq-dictionaries-update-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-dictionaries-update-ms
  type: ClusterIP
EOF
```

2.9.7 Сервис dhq-enrichment-ms

Serviceaccount dhq-enrichment-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-enrichmentg-ms
    project: dhq
  name: dhq-enrichment-ms
EOF
```

Configmap dhq-enrichment-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  APP_LOG_LEVEL: INFO
  DATA_GATEWAY_URL: http://dhq-data-gateway-ms:8080
  KAFKA_BOOTSTRAP_SERVERS: <kafka-host>:9092
  KAFKA_SCHEMA_REGISTRY_URL: http://<schema-registry-host>:8081
  TZ: Europe/Moscow
kind: ConfigMap
metadata:
  labels:
    app: dhq-enrichment-ms
    project: dhq
  name: dhq-enrichment-ms-configmap-env
EOF
```

Deployment dhq-enrichment-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-enrichment-ms
    project: dhq
  name: dhq-enrichment-ms
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dhq-enrichment-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-enrichment-ms
        project: dhq
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Xdebug -
agentlib:jdwp=transport=dt_socket,address=*:5005,server=y,suspend=n
        envFrom:
        - configMapRef:
            name: dhq-enrichment-ms-configmap-env
        image: dhq-enrichment-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-enrichment-ms
        ports:
        - containerPort: 8080
          name: enrichment
          protocol: TCP
        - containerPort: 5005
          name: debug
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-enrichment-ms
      serviceAccountName: dhq-enrichment-ms
      terminationGracePeriodSeconds: 30
EOF
```

Service dhq-enrichment-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-enrichment-ms
    project: dhq
  name: dhq-enrichment-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-enrichment-ms
  type: ClusterIP
EOF
```

2.9.8 Сервис dhq-event-processing-ms

Serviceaccount

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-event-processing-ms
    project: dhq
  name: dhq-event-processing-ms
EOF
```

Configmap dhq-event-processing-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  BPM_URL: http://dhq-bpm-ms:8080
  DATA_GATEWAY_URL: http://dhq-data-gateway-ms:8080
  INCIDENT_WORKFLOW_URL: http://dhq-incident-workflow-ms:8080
  KAFKA_BOOTSTRAP_SERVERS: kafka:9092
  KAFKA_SCHEMA_REGISTRY_URL: http://<schema-registry-host>:8081
  KAFKA_SERVER_HOST: kafka-headless
  KAFKA_SERVER_PORT: "9092"
  TZ: Europe/Moscow
kind: ConfigMap
metadata:
  labels:
    app: dhq-event-processing-ms
    project: dhq
  name: dhq-event-processing-ms-configmap-env
EOF
```

Deployment dhq-event-processing-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-event-processing-ms
    project: dhq
  name: dhq-event-processing-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dhq-event-processing-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-event-processing-ms
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-event-processing-ms-configmap-env
        image: dhq-event-processing-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-event-processing-ms
        ports:
        - containerPort: 8080
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-event-processing-ms
      serviceAccountName: dhq-event-processing-ms
EOF
```

Service dhq-event-processing-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-event-processing-ms
    project: dhq
  name: dhq-event-processing-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-event-processing-ms
  type: ClusterIP
EOF
```

2.9.9 Сервис dhq-frontend

Serviceaccount dhq-frontend

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-frontend
    project: dhq
  name: dhq-frontend
EOF
```

Configmap dhq-frontend-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  TZ: Europe/Moscow
kind: ConfigMap
metadata:
  labels:
    app: dhq-frontend
    project: dhq
  name: dhq-frontend-configmap-env
EOF
```

Deployment dhq-frontend

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-frontend
    project: dhq
  name: dhq-frontend
  namespace: dhq-dev
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dhq-frontend
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-frontend
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-frontend-configmap-env
        image: dhq-frontend:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-frontend
        ports:
        - containerPort: 80
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-frontend
      serviceAccountName: dhq-frontend
EOF
```

Service dhq-frontend

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-frontend
    project: dhq
  name: dhq-frontend
spec:
  ports:
  - name: http
    targetPort: 80
    port: 80
  selector:
    app: dhq-frontend
  type: ClusterIP
EOF
```

Ingress dhq-frontend

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  labels:
    app: dhq-frontend
    project: dhq
  name: dhq-frontend
  namespace: dhq-dev
spec:
  ingressClassName: nginx
  rules:
  - host: somedomain.ru
    http:
      paths:
      - backend:
          service:
            name: dhq-frontend
            port:
              number: 80
        path: /
        pathType: Prefix
  tls:
  - hosts:
    - somedomain.ru
    secretName: somedomain
EOF
```

2.9.10 Сервис dhq-integration-ms

Serviceaccount

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-integration-ms
    project: dhq
  name: dhq-integration-ms
EOF
```

Deployment dhq-integration-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-integration-ms
    project: dhq
  name: dhq-integration-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dhq-integration-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-integration-ms
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-integration-ms-configmap-env
        - secretRef:
            name: dhq-integration-ms-secrets
        image: dhq-integration-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-integration-ms
        ports:
        - containerPort: 8080
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-integration-ms
      serviceAccountName: dhq-integration-ms
EOF
```

Secret dhq-integration-ms-secrets

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  KEYCLOAK_PASSWORD: <KEYCLOAK_PASSWORD_base64>
kind: Secret
metadata:
  labels:
    app: dhq-integration-ms
    project: dhq
  name: dhq-integration-ms-secrets
type: Opaque
EOF
```

Configmap dhq-integration-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  APPLICATION_PORT: "8080"
  BASTION_ADAPTER_URL: dhq-bastion-adapter-ms:8080
  BPM_URL: http://dhq-bpm-ms:8080
  COMPANY_FLAG: "false"
  DATA_GATEWAY_URL: http://dhq-data-gateway-ms:8080
  EDMS_PASSWORD: ""
  EDMS_USERNAME: ""
  IS_COMPANY: "false"
  KEYCLOAK_CLIENT_ID: <KEYCLOAK_CLIENT_ID>
  KEYCLOAK_REALM: DHQ
  KEYCLOAK_URL: https://somedomain.ru/auth
  KEYCLOAK_USER: <KEYCLOAK_USER>
  ORGANIZATION_ID: <ORGANIZATION_ID>
  SYSTEM_USER_ID: <SYSTEM_USER_ID>
  TOKEN_GRANT_TYPE: password
  TZ: Europe/Moscow
kind: ConfigMap
metadata:
  labels:
    app: dhq-integration-ms
    project: dhq
  name: dhq-integration-ms-configmap-env
EOF
```

Service dhq-integration-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-integration-ms
    project: dhq
  name: dhq-integration-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-integration-ms
  type: ClusterIP
EOF
```

2.9.11 Сервис dhq-messages-ms

Serviceaccount dhq-messages-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-messages-ms
    project: dhq
  name: dhq-messages-ms
EOF
```

Configmap dhq-messages-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  APPLICATION_PORT: "8080"
  DATA_GATEWAY_URL: http://dhq-data-gateway-ms:8080
  DB_HOST: <mongodb_host>
  DB_PASSWORD: <mongodb_user_pass>
  DB_PORT: "27017"
  DB_USER: <mongodb_user>
  RDM_URL: dhq-rdm-ms:8080
  REDIS_SERVER_HOST: redis
  REDIS_SERVER_PORT: "6379"
  TZ: Europe/Moscow
  WEBSOCKET_ENABLED: "true"
  WEBSOCKET_URL: ws://dhq-backend-for-frontend-ms:8080/api/ws
kind: ConfigMap
metadata:
  labels:
    app: dhq-messages-ms
    project: dhq
  name: dhq-messages-ms-configmap-env
EOF
```

Deployment dhq-messages-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-messages-ms
    project: dhq
  name: dhq-messages-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dhq-messages-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-messages-ms
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-messages-ms-configmap-env
        image: dhq-messages-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-messages-ms
        ports:
        - containerPort: 8080
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-messages-ms
      serviceAccountName: dhq-messages-ms
EOF
```

Service dhq-messages-ms

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-messages-ms
    project: dhq
  name: dhq-messages-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-messages-ms
  type: ClusterIP
EOF
```

2.9.12 Сервис dhq-metadata

Serviceaccount dhq-metadata

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-metadata
    project: dhq
  name: dhq-metadata
EOF
```

Configmap dhq-metadata-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
apiVersion: v1
data:
  TZ: Europe/Moscow
kind: ConfigMap
metadata:
  labels:
    app: dhq-metadata
    project: dhq
  name: dhq-metadata-configmap-env
EOF Deployment dhq-metadata
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-metadata
    project: dhq
  name: dhq-metadata
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dhq-metadata
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-metadata
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-metadata-configmap-env
        image: dhq-metadata:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-metadata
        ports:
        - containerPort: 80
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-metadata
      serviceAccountName: dhq-metadata
EOF
```

Service dhq-metadata

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-metadata
    project: dhq
  name: dhq-metadata
spec:
  ports:
  - name: http
    targetPort: 80
    port: 80
  selector:
    app: dhq-metadata
  type: ClusterIP
EOF
```

Ingress dhq-metadata

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/configuration-snippet: |
      more_set_headers "Access-Control-Allow-Origin: \$http_origin";
    nginx.ingress.kubernetes.io/enable-cors: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /\$1
  labels:
    app: dhq-metadata
    project: dhq
  name: dhq-metadata
spec:
  ingressClassName: nginx
  rules:
  - host: somedomain.ru
    http:
      paths:
      - backend:
          service:
            name: dhq-metadata
            port:
              number: 80
        path: /templates/(.*)
        pathType: Prefix
  tls:
  - hosts:
    - somedomain.ru
    secretName: somedomain
EOF
```

2.9.13 Сервис dhq-notification-ms

Serviceaccount

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-notification-ms
    project: dhq
  name: dhq-notification-ms
EOF
```

Configmap dhq-notification-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  APP.EMAIL.BATCH_SIZE: "1"
  APP.EMAIL.CRON: 0 * * * * *
  APP.EMAIL.FROM: <user@somedomain.ru>
  APP.USERS.URL: http://dhq-rdm-ms:8080
  DATA_GATEWAY_URL: http://dhq-data-gateway-ms:8080
  DATASOURCE.PASSWORD: <pgsql_user_pass>
  DATASOURCE.URL: jdbc:postgresql://<pgsql_db_host>:5432/<pgsql_db_name>
  DATASOURCE.USERNAME: <pgsql_db_username>
  DB_HOST: mongodb
  DB_PASSWORD: <mongodb_db_pass>
  DB_PORT: "27017"
  DB_USER: <mongodb_user_pass>
  JPA.DATABASE.PLATFORM: org.hibernate.dialect.PostgreSQLDialect
  JPA.SHOW.SQL: "false"
  MAIL.AUTH: "true"
  MAIL.CON.TIMEOUT: "5000"
  MAIL.DEBUG: "true"
  MAIL.HOST: email.somedomain.ru
  MAIL.PASSWORD: <mail_pass>
  MAIL.PORT: "587"
  MAIL.PROTOCOL: smtp
  MAIL.READ.TIMEOUT: "5000"
  MAIL.TEST_CONNECTION: "false"
  MAIL.TLS.ENABLED: "true"
  MAIL.TLS.REQUIRED: "true"
  MAIL.USERNAME: <user@somedomain.ru>
  MAIL.WRITE.TIMEOUT: "5000"
  REDIS_SERVER_HOST: redis
  REDIS_SERVER_PORT: "6379"
  SERVER.PORT: "8080"
kind: ConfigMap
metadata:
  labels:
    app: dhq-notification-ms
    project: dhq
  name: dhq-notification-ms-configmap-env
EOF
```

Deployment dhq-notification-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-notification-ms
    project: dhq
  name: dhq-notification-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dhq-notification-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-notification-ms
        project: dhq
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: dhq-notification-ms-configmap-env
        image: dhq-notification-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-notification-ms
        ports:
        - containerPort: 8080
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-notification-ms
      serviceAccountName: dhq-notification-ms
      terminationGracePeriodSeconds: 30
      volumes:
      - configMap:
          defaultMode: 420
          name: dhq-notification-ms-configmap
        name: dhq-notification-ms-configmap
EOF
```

Service

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-notification-ms
    project: dhq
  name: dhq-notification-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-notification-ms
  type: ClusterIP
EOF
```

2.9.14 Сервис dhq-rdm-ms

Serviceaccount

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-rdm-ms
    project: dhq
  name: dhq-rdm-ms
EOF
```

Configmap dhq-rdm-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  APPLICATION_PORT: "8080"
  DB_CONNECTION: pgsql
  DB_HOST: <DB_HOST>
  DB_NAME: <DB_NAME>
  DB_PASSWORD: <DB_PASSWORD>
  DB_PORT: "5432"
  DB_USER: <DB_USER>
  EMAIL_NOTIFICATION_ENABLED: "true"
  EMAIL_NOTIFICATION_URL: http://dhq-notification-ms:8080/notification
  IS_COMPANY: "false"
  LOGGING_LEVEL: INFO
  NOTIFICATION_URL: http://dhq-notification-ms:8080/notification
  ORGANIZATION_ID: <ORGANIZATION_ID>
  REDIS_SERVER_HOST: redis
  REDIS_SERVER_PORT: "6379"
  SPRING_PROFILES_ACTIVE: organization
  TZ: Europe/Moscow
  UI_NOTIFICATION_ENABLED: "true"
  WEBSOCKET_ENABLED: "true"
  WEBSOCKET_URL: ws://dhq-backend-for-frontend-ms:8080/api/ws
kind: ConfigMap
metadata:
  labels:
    app: dhq-rdm-ms
    project: dhq
  name: dhq-rdm-ms-configmap-env
EOFDeployment
kubectl apply -f - <<EOF
```

Service

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-rdm-ms
    project: dhq
  name: dhq-rdm-ms
spec:
  ports:
  - name: http
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-rdm-ms
  type: ClusterIP
EOF
```

Deployment dhq-rdm-ms

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-rdm-ms
    project: dhq
  name: dhq-rdm-ms
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dhq-rdm-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-rdm-ms
        project: dhq
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Xdebug -
agentlib:jdwp=transport=dt_socket,address=*:5005,server=y,suspend=n
        envFrom:
        - configMapRef:
            name: dhq-rdm-ms-configmap-env
        image: dhq-rdm-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-rdm-ms
        ports:
        - containerPort: 8080
          name: http-rdm
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      initContainers:
      - args:
        - -c
        - until pg_isready -h $DB_HOST -p $DB_PORT; do echo waiting for database;
          sleep 2; done;
        command:
        - /bin/sh
        envFrom:
        - configMapRef:
            name: dhq-rdm-ms-configmap-env
        image: postgres:11.15
        imagePullPolicy: IfNotPresent
        name: check-db-ready
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      restartPolicy: Always
      schedulerName: default-scheduler
      serviceAccount: dhq-rdm-ms
      serviceAccountName: dhq-rdm-ms
      terminationGracePeriodSeconds: 30
      volumes:
      - configMap:
```

```
            defaultMode: 420
            name: dhq-rdm-ms-configmap
          name: dhq-rdm-ms-configmap
EOF
```

2.9.15 Сервис dhq-video-processing-ms

Serviceaccount

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: dhq-video-processing-ms
    project: dhq
  name: dhq-video-processing-ms
EOF
```

Configmap configmap-nginx

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  nginx.conf: |-
    worker_processes auto;
    rtmp_auto_push on;
    events {}
    rtmp {
        server {
            listen 1935;
            application live {
                live on;
                #record off;
                #interleave on;

                hls on;
                hls_path /tmp/hls/;
                hls_nested on;
                hls_fragment 4s; # default is 5s
                # hls_playlist_length 60; # default is 30s
            }
        }
    }
    http {
        default_type application/octet-stream;

        server {
            listen 80;
            location /video {
                root /tmp/hls;
            }
        }

        types {
            application/vnd.apple.mpegurl m3u8;
            video/mp2t ts;
            text/html html;
        }
    }
kind: ConfigMap
metadata:
  labels:
    app: dhq-video-processing-ms
    project: dhq
  name: configmap-nginx
EOF
```

Configmap dhq-video-processing-ms-configmap-env

```
kubectl apply -f - <<EOF
apiVersion: v1
data:
  APPLICATION_PORT: "8080"
  COMPANY_STREAM_LIMIT: "4"
  DATA_GATEWAY_URL: dhq-data-gateway-ms:8080
  DB_HOST: pgsql
  DB_NAME: <db_name>
  DB_PASSWORD: <db_pass>
  DB_PORT: "5432"
  DB_USER: <db_user>
  INTELLECT_ADAPTER_URL: http://<intellect_ip>:8080
  IS_ORGANIZATION: "true"
  MEDIA_SERVER_IP: <mediamtx_ip>:8554
  MEDIA_SERVER_LOGIN: <MEDIA_SERVER_LOGIN>
  MEDIA_SERVER_PASSWORD: <MEDIA_SERVER_PASSWORD>
  MEDIA_SERVER_URL: ${MEDIA_SERVER_LOGIN}:${MEDIA_SERVER_PASSWORD}@<mediamtx_ip>:8554
  MINIO_ACCESS_KEY: <MINIO_ACCESS_KEY>
  MINIO_SECRET_KEY: <MINIO_SECRET_KEY>
  MINIO_URL: https://minio.somedomain.ru
  ORGANIZATION_STREAM_LIMIT: "4"
  TZ: Europe/Moscow
  VIDEO_PROVIDER_PROCESSING_RATE: "30"
  VIDEO_PROVIDER_RECOVERY_RATE: "120"
kind: ConfigMap
metadata:
  labels:
    app: dhq-video-processing-ms
    project: dhq
  name: dhq-video-processing-ms-configmap-env
EOF
```

Deployment

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: dhq-video-processing-ms
    project: dhq
  name: dhq-video-processing-ms
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dhq-video-processing-ms
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhq-video-processing-ms
        project: dhq
    spec:
      containers:
      - image: tiangolo/nginx-rtmp:latest-2023-04-03
        imagePullPolicy: IfNotPresent
        name: nginx
        ports:
        - containerPort: 80
          name: http-nginx
          protocol: TCP
        securityContext:
          runAsUser: 0
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /etc/nginx/nginx.conf
          name: configmap-nginx
          subPath: nginx.conf
        - mountPath: /tmp/hls
          name: shared-data
        - mountPath: /etc/localtime
          name: tz
      - envFrom:
        - configMapRef:
            name: dhq-video-processing-ms-configmap-env
        image: dhq-video-processing-ms:v1.0
        imagePullPolicy: IfNotPresent
        name: dhq-video-processing-ms
        ports:
        - containerPort: 8080
          name: http
          protocol: TCP
        securityContext:
          runAsUser: 0
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /dhq
          name: shared-data
      dnsPolicy: ClusterFirst
      initContainers:
      - args:
```

```
          - cp -a /dhq/. /dhq2/; chown -R dhq:dhq /dhq2/;
          command:
          - /bin/sh
          - -c
          envFrom:
          - configMapRef:
              name: dhq-video-processing-ms-configmap-env
          image: dhq-video-processing-ms:v1.0
          imagePullPolicy: IfNotPresent
          name: migrate
          securityContext:
            runAsUser: 0
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          volumeMounts:
          - mountPath: /dhq2
            name: shared-data
        restartPolicy: Always
        schedulerName: default-scheduler
        serviceAccount: dhq-dev-dhq-video-processing-ms
        serviceAccountName: dhq-dev-dhq-video-processing-ms
        terminationGracePeriodSeconds: 30
        volumes:
        - name: shared-data
        - configMap:
            defaultMode: 420
            name: dhq-video-processing-ms-configmap
          name: dhq-video-processing-ms-configmap
        - configMap:
            defaultMode: 420
            name: configmap-nginx
          name: configmap-nginx
        - hostPath:
            path: /usr/share/zoneinfo/Europe/Moscow
            type: ""
          name: tz
EOF
```

Service

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: dhq-video-processing-ms
    project: dhq
  name: dhq-video-processing-ms
spec:
  ports:
  - name: http
    targetPort: 80
    port: 80
  - name: http-jar
    targetPort: 8080
    port: 8080
  selector:
    app: dhq-video-processing-ms
  type: ClusterIP
EOF
```

Ingress dhq-video-processing-ms

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/configuration-snippet : |
        if ($request_uri ~* \.(m3u8|ts)) {
           add_header Cache-Control "no-cache,no-store";
        }
    nginx.ingress.kubernetes.io/cors-allow-origin: https://somedomain.ru
    nginx.ingress.kubernetes.io/enable-cors: "true"
  labels:
    app: dhq-video-processing-ms
    project: dhq
  name: dhq-video-processing-ms
spec:
  ingressClassName: nginx
  rules:
  - host: somedomain.ru
    http:
      paths:
      - backend:
          service:
            name: dhq-video-processing-ms
            port:
              number: 80
        path: /video/.*
        pathType: Prefix
  tls:
  - hosts:
    - somedomain.ru
    secretName: somedomain
EOF
```

Ingress dhq-video-processing-ms-video

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/cors-allow-credentials: "true"
    nginx.ingress.kubernetes.io/cors-allow-methods: PATCH, PUT, GET, POST, OPTIONS
    nginx.ingress.kubernetes.io/cors-allow-origin: https://somedomain.ru
    nginx.ingress.kubernetes.io/enable-cors: "true"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "1800"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "1800"
  labels:
    app: dhq-video-processing-ms
    project: dhq
  name: dhq-video-processing-ms-video
spec:
  ingressClassName: nginx
  rules:
  - host: somedomain.ru
    http:
      paths:
      - backend:
          service:
            name: dhq-video-processing-ms
            port:
              number: 8080
        path: /videostream.*
        pathType: Prefix
      - backend:
          service:
            name: dhq-video-processing-ms
            port:
              number: 8080
        path: /videofiles/.*
        pathType: Prefix
  tls:
  - hosts:
    - somedomain.ru
    secretName: somedomain
EOF
```