Руководство по развёртыванию приложения "Цифровой двойник рисков производства" (ЦДР) на стенде в Minikube для регистрации в реестре российского ПО

> Краткая справка по стенду `rosreestr`:
>
> **Приложение ЦДР `rosreestr`:**
>
> - Frontend: https://rosreestr-frontend.cdr.usetech.ru/ (доступ из Интернет)
>   ▶ Подробнее
> - Бэкенд configurator: http://rosreestr-configurator.cdr.usetech.ru/api/actuator/health (доступ только из внутренней сети Юзтех)
> - Бэкенд risksSolver: http://rosreestr-riskssolver.cdr.usetech.ru/api/actuator/health (доступ только из внутренней сети Юзтех)
>
> **Инфраструктурные подсистемы:**
>
> - Подсистема хранения - PostgreSQL: postgres.cdr.usetech.ru, порт: 5432 (доступ только из внутренней сети Юзтех)
>   ▶ Данные для подключения к БД стенда rosreestr
> - Веб-интерфейс подсистемы хранения - MinIO: https://minio-console.cdr.usetech.ru (доступ только из внутренней сети Юзтех)
> - Веб-интерфейс подсистемы мониторинга и логирования - Grafana: http://grafana.cdr.usetech.ru/dashboards (доступ только из внутренней сети Юзтех)

Содержание

Введение

Настоящее руководство содержит описание подготовки минимальной установки Kubernetes в варианте Minikube, приведённое в разделе 1, описания развёртывания инфраструктурных подсистем, не входящих в состав приложения ЦДР, таких как подсистема хранения данных и мониторинга, приведённые в разделе 2, а также описание развёртывания приложения ЦДР, приведённое в разделе 3.

В примерах команд, приведённых в настоящем руководстве, используется название стенда `dev`. При развёртывании на стенде с другим названием, например, `rosreestr`, следует заменить в командах все вхождения слова `dev` на `rosreestr`.

Следует учитывать, что в соответствии с политикой безопасности компании Юзтех, доступ из сети Интернет открыт только к веб-интерфейсу пользователя (фронтенду) приложения ЦДР. Доступ к прочим компонентам приложения ЦДР и к инфраструктурным подсистемам возможен только из внутренней сети Юзтех.

1 Развёртывание Minikube и настройка интеграции с GitLab

1.1 Установка Minikube и необходимых утилит

1.1.1 Добавить текущего пользователя в группу `docker` и убедиться, что служба `docker` запущена и доступна пользователю:

```
sudo usermod -a -G docker $USER && \
sudo -u $USER $SHELL && \
docker ps
```

1.1.2 Установить утилиту kubectl:

```
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg && \
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | \
  sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg && \
sudo chmod 644 /etc/apt/keyrings/kubernetes-apt-keyring.gpg && \
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | \
  sudo tee /etc/apt/sources.list.d/kubernetes.list && \
sudo chmod 644 /etc/apt/sources.list.d/kubernetes.list && \
sudo apt-get update && \
sudo apt-get install -y kubectl && \
echo 'source <(kubectl completion bash)' >> ~/.bashrc
```

1.1.3 Установить утилиту helm:

```
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee
/usr/share/keyrings/helm.gpg > /dev/null && \
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/
all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list && \
sudo apt-get update && \
sudo apt-get install helm
```

1.1.4 Настроить Bash-completion для утилит `kubectl` и `helm`:

```
sudo sh -c 'kubectl completion bash > /etc/bash_completion.d/kubectl'
sudo sh -c 'helm completion bash > /etc/bash_completion.d/helm'
```

1.1.5 Установить вспомогательные утилиты:

```
DEBIAN_FRONTEND=noninteractive \
sudo apt install -y \
  jq \
  iptables-persistent
```

1.1.6 Создать файл `/etc/iptables/rules.v4`, содержащий правила NAT IPTables, необходимые для обеспечения возможности сетевого доступа к сервисам, запущенным в Minikube:

```
*nat
-A PREROUTING -i eth0 -p tcp -m tcp --dport 80 -j DNAT --to-destination
192.168.123.123:80
-A PREROUTING -i eth0 -p tcp -m tcp --dport 443 -j DNAT --to-destination
192.168.123.123:443
-A PREROUTING -i eth0 -p tcp -m tcp --dport 5432 -j DNAT --to-destination
192.168.123.123:30432
COMMIT
```

1.1.7 Запустить службу `netfilter-persistent` и перезапустить службу `docker`:

```
sudo systemctl enable --now netfilter-persistent
sudo systemctl restart docker
```

1.1.8 Создать файл `/etc/sysctl.d/99-minikube.conf` с параметрами `sysctl`, необходимыми для работы Minikube:

```
fs.inotify.max_user_watches=2099999999
fs.inotify.max_user_instances=2099999999
fs.inotify.max_queued_events=2099999999
```

1.1.9 Применить параметры `sysctl` из файла `/etc/sysctl.d/99-minikube.conf`:

```
sysctl -p /etc/sysctl.d/99-minikube.conf
```

1.1.10 Установить и запустить Minikube:

```
curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd
64.deb && \
sudo dpkg -i minikube_latest_amd64.deb && \
export CPUS=$(expr $(nproc) - 1) && \
export MEM=$(expr $(cat /proc/meminfo |grep MemTotal |sed -r 's/^.* ([0-
9]+) kB/\1/') \/ 1024 \/ 1024 - 2) && \
export LAN_IP=$(ip -4 addr show eth0 |grep -oP '(?<=inet\s)\d+(\.\d+){3}')
&& \
echo "Minikube will use: CPUs: $CPUS, MEM: $MEM GB"
minikube start \
  --cpus=${CPUS} \
  --memory=${MEM}g \
  --apiserver-ips=${LAN_IP} \
  --static-ip=192.168.123.123 \
  --ports=80:80,443:443,5432:30432,8443:8443 \
  --addons=ingress \
  --addons=storage-provisioner \
  --addons=default-storageclass \
  --addons=metrics-server && \
minikube status && \
kubectl get nodes && \
kubectl apply -f https://github.com/kubernetes-sigs/metrics-
server/releases/latest/download/components.yaml && \
kubectl top nodes
```

1.1.11 Создать файл `/usr/lib/systemd/system/minikube.service` конфигурации systemd для запуска Minikube при загрузке ОС:

```
[Unit]
Description=minikube
After=network-online.target docker.service
Wants=network-online.target docker.service
Requires=docker.socket docker.service

[Service]
```

```
Type=oneshot
RemainAfterExit=yes
WorkingDirectory=/home/developer
ExecStart=/usr/bin/minikube start
ExecStop=/usr/bin/minikube stop
User=developer
Group=developer

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

1.1.12 Включить автоматический запуск Minikube при загрузке ОС:

```
systemctl daemon-reload; \
systemctl enable minikube
```

1.2 Настройка доступа Minikube к GitLab Docker Registry

> ПРИМЕЧАНИЕ: Далее по тексту используется название стенда `dev`. При развёртывании на стенде с другим названием, например, `rosreestr`, следует заменить в командах все вхождения слова `dev` на `rosreestr`.

1.2.1 В веб-интерфейсе GitLab в меню `Settings -> Repository -> Deploy Tokens` группы проектов `cdr` создать групповой токен развёртывания с именем `deploy-token-cdr`, именем пользователя `deploy-token-cdr` и полномочиями `read_registry`. Сохранить токен - он потребуется на следующем шаге.

1.2.2 В пространстве имён Kubernetes настраиваемого стенда, например, `dev`, создать секрет Kubernetes с учётными данными для подключения к Docker Registry в GitLab UseTech для загрузки docker-образов приложения:

```
kubectl -n dev create secret docker-registry usetech-registry \
  --docker-server=registry.usetech.ru \
  --docker-username=deploy-token-cdr \
  --docker-password=<токен для R/O доступа к GitLab registry>
```

1.3 Настройка доступа GitLab CI/CD к кластеру Kubernetes

1.3.1 Вывести на консоль IP-адрес сетевого интерфейса `eth0` узла Minikube:

```
ip -4 addr show eth0 | grep -oP '(?<=inet\s)\d+(\.\d+){3}'
```

1.3.2 В GitLab-проектах каждой из подсистем (бэкенд, фронтенд) создать переменную GitLab CI/CD `KUBE_URL` со значением `https://<IP-адрес узла Minikube>:8443` и именем окружения `Minikube`.

1.3.3 Создать YAML-файл `sa-admin-user.yaml` с описанием объекта `ServiceAccount` и секрета для него:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kube-system
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: admin-user-token
  namespace: kube-system
  annotations:
    kubernetes.io/service-account.name: "admin-user"
```

1.3.4 Создать объект `ServiceAccount` и секрет для него:

```
kubectl create -f sa-admin-user.yaml
```

1.3.5 Получить токен администратора кластера Kubernetes для развёртывания приложения из GitLab CI/CD:

```
kubectl -n kube-system get secret $(kubectl -n kube-system get secret | \
  grep admin-user-token | \
  awk '{print $1}') -o jsonpath={.data.token} | \
  base64 -d
```

1.3.6 В GitLab-проектах каждой из подсистем создать переменную GitLab CI/CD `KUBE_TOKEN` со значением, полученным на предыдущем шаге и именем окружения `Minikube`.

2 Развёртывание инфраструктурных сервисов в Minikube

2.1 Развёртывание подсистемы хранения данных - MinIO

2.1.1 Создать YAML-файл `minio-operator-values.yaml`, содержащий параметры настройки Helm-чарта оператора MinIO:

```
operator:
  env:
    - name: TZ
      value: "UTC+3"
    - name: PROMETHEUS_NAMESPACE
      value: "monitoring"
    - name: MINIO_PROMETHEUS_URL
      value: "http://monitoring-kube-prometheus-
prometheus.monitoring.svc.cluster.local:9090"
  ingress:
    enabled: true
    ingressClassName: "nginx"
    labels: { }
    annotations: { }
    tls: [ ]
    host: minio-operator-console.cdr.usetech.ru
    path: /
    pathType: Prefix
```

2.1.2 Установить Helm-чарт оператора MinIO `minio-operator` [4]:

```
helm repo add minio-operator https://operator.min.io && \
helm install --create-namespace -n minio-operator \
  -f minio-operator-values.yaml \
  operator minio-operator/operator
```

2.1.3 Убедиться, что все компоненты оператора MinIO находятся в состоянии `READY`:

```
kubectl -n minio-operator get all
```

2.1.4 Получить JWT-токен для входа в веб-интерфейс оператора MinIO:

```
kubectl get secret/console-sa-secret -n minio-operator \
  -o jsonpath={.data.token} | base64 -d; echo
```

2.1.5 Проверить возможность входа в веб-интерфейс оператора MinIO.

2.1.7 Создать пространство имён Kubernetes `minio-cdr`:

```
kubectl create ns minio-cdr
```

2.1.6 Создать секрет Kubernetes, содержащий имя пользователя и пароль администратора сервера MinIO:

```
echo -e "export MINIO_ROOT_USER=<имя пользователя>\nexport
MINIO_ROOT_PASSWORD=<пароль>" | \
  kubectl -n minio-cdr create secret generic \
  minio-cdr-secret --from-file=config.env=/dev/stdin
```

2.1.7 Подготовить файлы закрытого ключа и fullchain-сертификата TLS для домена `*.cdr.usetech.ru`, затем создать секрет Kubernetes, содержащий сертификат TLS, который будет использоваться ingress-ом для защиты доступа к MinIO по сети:

```
kubectl -n minio-cdr create secret tls cdr-ingress-tls \
  --key="../path/to/privkey.pem" \
  --cert="../path/to/fullchain.pem"
```

2.1.8 Создать YAML-файл `minio-cdr-values.yaml`, содержащий параметры настройки Helm-чарта MinIO для развёртывания одиночного сервера MinIO с одним диском объёмом 10 ГБ:

```
existingSecret:
  name: minio-cdr-secret
tenant:
  name: cdr
  configuration:
    name: minio-cdr-secret
  pools:
    - servers: 1
      name: pool-0
      volumesPerServer: 1
      size: 10Gi
      storageClassName: standard
  metrics:
    enabled: true
    port: 9000
    protocol: http
  prometheusOperator: true
  logging: {
    json
  }
  env:
    - name: TZ
      value: "UTC+3"
    - name: PROMETHEUS_NAMESPACE
```

```
      value: "monitoring"
    - name: MINIO_PROMETHEUS_URL
      value: "http://monitoring-kube-prometheus-
prometheus.monitoring.svc.cluster.local:9090"
  certificate:
    externalCertSecret: [ ]
    requestAutoCert: false
ingress:
  api:
    enabled: true
    ingressClassName: nginx
    labels: { }
    host: minio.cdr.usetech.ru
    path: /
    pathType: Prefix
    tls:
      - secretName: cdr-ingress-tls
  console:
    enabled: true
    ingressClassName: nginx
    labels: { }
    host: minio-console.cdr.usetech.ru
    path: /
    pathType: Prefix
    tls:
      - secretName: cdr-ingress-tls
```

2.1.9 Установить Helm-чарт тенанта MinIO `minio-tenant` [5]:

```
helm upgrade --install --namespace minio-cdr \
  --values minio-cdr-values.yaml \
  cdr minio-operator/tenant
```

2.1.10 Убедиться, что все компоненты тенанта MinIO находятся в состоянии `READY`:

```
kubectl -n minio-cdr get all
```

2.1.11 Вывести на консоль имя пользователя и пароль администратора MinIO:

```
kubectl -n minio-cdr get secrets minio-cdr-secret \
  -o jsonpath={.data."config\.env"} | base64 -d
```

2.1.12 Проверить возможность входа в веб-интерфейс тенанта MinIO.

> ВНИМАНИЕ: Для доступа к MinIO приложений, развёрнутых в Kubernetes, следует использовать адрес: https://minio.minio-cdr.svc.cluster.local/.

2.1.13 Создать в веб-интерфейсе Grafana дашборд MinIO: https://grafana.com/grafana/dashboards/13502-minio-dashboard/.

2.2 Развёртывание подсистемы мониторинга - Kube-Prometheus-Stack

2.2.1 Подготовить файлы закрытого ключа и fullchain-сертификата TLS для домена `*.cdr.usetech.ru`, затем создать секрет Kubernetes, содержащий сертификат TLS, который будет использоваться ingress-ом для защиты доступа к веб-интерфейсу подсистемы мониторинга по сети:

```
kubectl -n monitoring create secret tls cdr-ingress-tls \
  --key="../path/to/privkey.pem" \
  --cert="../path/to/fullchain.pem"
```

2.2.2 Создать YAML-файл `kube-prometheus-stack-values.yaml`, содержащий параметры настройки Helm-чарта Kube-Prometheus-Stack:

```
prometheus:
  prometheusSpec:
    retention: 10d
  ingress:
    enabled: true
    ingressClassName: nginx
    hosts:
      - prometheus.cdr.usetech.ru
    paths:
      - /
    tls:
      - secretName: cdr-ingress-tls
grafana:
  defaultDashboardsTimezone: utc+3
  persistence:
    enabled: true
  ingress:
    enabled: true
    ingressClassName: nginx
    hosts:
      - grafana.cdr.usetech.ru
    paths:
      - /
    tls:
      - secretName: cdr-ingress-tls
  additionalDataSources:
   - name: loki
     editable: true
     orgId: 1
     type: loki
     url: http://loki-gateway.loki.svc.cluster.local/
     version: 1
```

2.2.3 Установить Helm-чарт Kube-Prometheus-Stack [1]:

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts && \
helm repo update && \
helm upgrade --install --create-namespace monitoring -n monitoring \
  -f kube-prometheus-stack-values.yaml \
  prometheus-community/kube-prometheus-stack
```

2.2.4 Убедиться, что все компоненты kube-prometheus-stack находятся в состоянии READY:

```
kubectl -n monitoring get all
```

2.2.5 Вывести на консоль имя пользователя и пароль Grafana:

```
kubectl -n monitoring get secrets monitoring-grafana \
  -o jsonpath={.data.admin-user} |base64 -d; \
echo ; \
kubectl -n monitoring get secrets monitoring-grafana \
  -o jsonpath={.data.admin-password} |base64 -d; \
echo
```

2.2.6 Проверить доступность веб-интерфейса Grafana.

2.2.7 Проверить доступность веб-интерфейса Prometheus.

2.3 Развёртывание подсистемы логирования - Loki и Promtail

2.3.1 Создать YAML-файл `loki-values.yaml`, содержащий параметры настройки Helm-чарта Loki для развёртывания одиночного сервера Loki с хранением данных в файловой системе и сроком хранения данных 10 дней:

```
deploymentMode: SingleBinary
loki:
  auth_enabled: false
  commonConfig:
    replication_factor: 1
  storage:
    type: 'filesystem'
  schemaConfig:
    configs:
    - from: "2024-01-01"
      store: tsdb
      index:
        prefix: loki_index_
        period: 24h
```

```
      object_store: filesystem
      schema: v13
  compactor:
    working_directory: /var/loki/compactor
    compaction_interval: 10m
    retention_enabled: true
    delete_request_store: filesystem
    retention_delete_delay: 2h
    retention_delete_worker_count: 150
  limits_config:
    # Keep logs for 10 days
    retention_period: 240h
singleBinary:
  replicas: 1
read:
  replicas: 0
backend:
  replicas: 0
write:
  replicas: 0
monitoring:
  serviceMonitor:
    enabled: true
    labels:
      release: monitoring
```

2.3.2 Установить Helm-чарт Loki:

```
helm repo add grafana https://grafana.github.io/helm-charts && \
helm repo update && \
helm upgrade --install --create-namespace --namespace=loki \
  --values loki-values.yaml \
  loki grafana/loki
```

2.3.3 Создать YAML-файл `promtail-values.yaml`, содержащий параметры настройки Helm-чарта Promtail:

```
config:
  clients:
    - url: http://loki-gateway.loki.svc.cluster.local/loki/api/v1/push
      tenant_id: 1
serviceMonitor:
  enabled: true
  labels:
    release: monitoring
```

2.3.4 Установить Helm-чарт Promtail:

```
helm upgrade --install --namespace=loki \
  --values promtail-values.yaml \
  promtail grafana/promtail
```

2.3.5 Проверить доступность данных журналов (логов) в Grafana по ссылке.

2.3.6 Добавить в Grafana дашборды:

- Loki: https://grafana.com/grafana/dashboards/13407-loki2-0-global-metrics/
- Promtail: https://grafana.com/grafana/dashboards/15443-promtail/.

2.4 Развёртывание подсистемы хранения данных - PostgreSQL

2.4.1 Создать секрет Kubernetes `postgres-secret`, содержащий пароль администратора СУБД PostgreSQL - `postgres`:

```
kubectl create secret generic postgres-secret \
  --from-literal=password=<пароль пользователя postgres>
```

2.4.2 Создать YAML-файл `postgres-sts.yaml` с описанием объектов `Service`, `StatefulSet`, `PersistentVolumeClaim`, `ServiceMonitor` и настройками параметров производительности СУБД PostgreSQL, оптимизированными для использования 2 ядер ЦПУ и 4 ГБ ОЗУ:

```
apiVersion: v1
kind: Service
metadata:
  name: postgres
  labels:
    app: postgres
spec:
  selector:
    app: postgres
  type: NodePort
  ports:
    - port: 5432
      nodePort: 30432
      name: postgres
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app: postgres
  name: postgres
spec:
  serviceName: postgres
  replicas: 1
  selector:
```

```
      matchLabels:
        app: postgres
    template:
      metadata:
        labels:
          app: postgres
      spec:
        containers:
          - name: postgres
            image: postgres:15
            args:
              - -c
              - max_connections=120
              - -c
              - shared_buffers=1280MB
              - -c
              - effective_cache_size=3840MB
              - -c
              - maintenance_work_mem=320MB
              - -c
              - checkpoint_completion_target=0.9
              - -c
              - wal_buffers=16MB
              - -c
              - default_statistics_target=100
              - -c
              - random_page_cost=4
              - -c
              - effective_io_concurrency=2
              - -c
              - work_mem=3276kB
              - -c
              - huge_pages=off
              - -c
              - min_wal_size=1GB
              - -c
              - max_wal_size=4GB
            env:
              - name: POSTGRES_USER
                value: "postgres"
              - name: POSTGRES_PASSWORD
                valueFrom:
                  secretKeyRef:
                    name: postgres-secret
                    key: password
              - name: PGDATA
                value: "/var/lib/postgresql/data/pgdata"
            ports:
              - containerPort: 5432
                name: postgres
            volumeMounts:
              - name: postgres-data
                mountPath: /var/lib/postgresql/data
        volumes:
```

```
        - name: postgres-data
          persistentVolumeClaim:
            claimName: postgres-pvc
      initContainers:
        - name: exporter
          image: prometheuscommunity/postgres-exporter:v0.15.0
          env:
            - name: DATA_SOURCE_URI
              value: "postgres?sslmode=disable"
            - name: DATA_SOURCE_USER
              value: "postgres"
            - name: DATA_SOURCE_PASS
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: password
          ports:
            - containerPort: 9187
              name: exporter
          restartPolicy: Always
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: postgres-pvc
  labels:
    app: postgres
spec:
  storageClassName: standard
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-exporter
  labels:
    app: postgres
spec:
  selector:
    app: postgres
  ports:
    - protocol: TCP
      port: 9187
      targetPort: 9187
      name: exporter
---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
```

```
    app: postgres
    release: monitoring
  name: postgres-exporter
spec:
  endpoints:
  - interval: 15s
    path: /metrics
    port: exporter
    scheme: http
  selector:
    matchLabels:
      app: postgres
```

2.4.3 Создать объекты Kubernetes, описанные в файле `postgres-sts.yaml`:

```
kubectl apply -f postgres-sts.yaml
```

2.4.4 Вывести на консоль Kubernetes-ресурсы PostgreSQL и убедиться, что запустился pod `postgres-0`:

```
kubectl get all -l app=postgres
```

2.4.5 Вывести на консоль пароль пользователя PostgreSQL, хранящийся в секрете Kubernetes `postgres-secret`:

```
kubectl get secrets postgres-secret -o jsonpath={.data.password} |base64 -d
```

2.4.6 Проверить подключение к БД PostgreSQL с узла Minikube:

```
kubectl exec -it postgres-0 -- psql -h localhost -U postgres --password -p
5432
```

На запрос вышеприведённой команды ввести пароль пользователя PostgreSQL, полученный на предыдущем шаге. Пример успешного выполнения команды:

```
Password:
psql (15.7 (Debian 15.7-1.pgdg120+1))
Type "help" for help.

postgres=#
```

Для того, чтобы отключиться от PostgreSQL, ввести `\q` и нажать на клавишу `<Enter>`.

2.4.7 Проверить подключение к БД PostgreSQL с удалённого узла через VPN:

```
psql -h 192.168.50.92 -p 5432 -U postgres
```

2.4.8 Добавить в Grafana дашборд: https://grafana.com/grafana/dashboards/9628-postgresql-database/.

2.5 Настройка резервного копирования БД PostgreSQL

2.5.1 Войти в веб-интерфейс тенанта MinIO и создать бакет `postgres-backup`, пользователя `postgres-backup` с полномочиями на чтение и запись, а также ключи доступа (service account) для означенного пользователя. Ключи необходимо сохранить - они потребуются на следующем шаге.

2.5.2 Создать секрет Kubernetes, содержащий параметры, необходимые для сохранения резервных копий БД в бакет MinIO:

```
kubectl create secret generic postgres-backup-secret \
  --from-literal=AWS_ACCESS_KEY_ID=<ключ доступа MinIO> \
  --from-literal=AWS_SECRET_ACCESS_KEY=<секретный ключ MinIO>
```

2.5.3 Создать YAML-файл `dev-postgres-backup-cronjob.yaml` с описанием объекта Kubernetes CronJob для выполнения резервного копирования БД `dev-configurator` ежедневно в 02:00 по Московскому времени в бакет MinIO `postgres-backup`:

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: dev-postgres-backup
spec:
  schedule: "00 02 * * *"
  timeZone: "Europe/Moscow"
  concurrencyPolicy: Forbid
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      backoffLimit: 0
      template:
        spec:
          restartPolicy: Never
          initContainers:
            - name: pg-dump
              image: postgres:15
              volumeMounts:
                - name: data
                  mountPath: /backup
```

```
            args:
              - pg_dump
              - "-Fc"
              - "-f"
              - "/backup/$(PGNAME).pgdump"
              - "-Z"
              - "9"
              - "-v"
              - "-h"
              - "$(PGHOST)"
              - "-p"
              - "$(PGPORT)"
              - "-U"
              - "$(PGUSER)"
              - "-d"
              - "$(PGNAME)"
            env:
              - name: PGHOST
                value: "postgres-0.postgres.default.svc.cluster.local"
              - name: PGPORT
                value: "5432"
              - name: PGNAME
                value: "dev-configurator"
              - name: PGUSER
                value: "postgres"
              - name: PGPASSWORD
                valueFrom:
                  secretKeyRef:
                    name: postgres-secret
                    key: password
        containers:
          - name: s3-upload
            image: amazon/aws-cli
            volumeMounts:
              - name: data
                mountPath: /backup
            command: ["/bin/sh", "-c"]
            args:
              - |
                aws s3 cp \
                --region "$(AWS_DEFAULT_REGION)" \
                --endpoint-url "$(AWS_ENDPOINT_URL)" \
                "/backup/$(PGNAME).pgdump" \
                "s3://$(AWS_BUCKET_NAME)/$(PGNAME)-$(date +%Y-%m-%d-%H-
%M).pgdump"
            env:
              - name: AWS_BUCKET_NAME
                value: "postgres-backup"
              - name: AWS_DEFAULT_REGION
                value: "ru-central1"
              - name: AWS_ENDPOINT_URL
                value: "http://minio.minio-cdr.svc.cluster.local"
              - name: PGNAME
                value: "dev-configurator"
```

```
                - name: TZ
                  value: "Europe/Moscow"
              envFrom:
                - secretRef:
                    # Must contain AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY
                    name: postgres-backup-secret
          volumes:
            - name: data
              emptyDir: {}
```

2.5.4 Создать объект CronJob, описанный в файле `dev-postgres-backup-cronjob.yaml`:

```
kubectl apply -f dev-postgres-backup-cronjob.yaml
```

2.6 Развёртывание подсистемы аутентификации - Keycloak

2.6.1 Вывести на консоль пароль пользователя PostgreSQL, хранящийся в секрете Kubernetes `postgres-secret`:

```
kubectl get secrets postgres-secret -o jsonpath={.data.password} |base64 -d
```

2.6.2 Подключиться к БД PostgreSQL:

```
kubectl exec -it postgres-0 -- psql -h localhost -U postgres --password -p
5432
```

2.6.3 Создать базу данных и пользователя `keycloak` в СУБД PostgreSQL:

```
CREATE USER "dev-cdr-keycloak" WITH ENCRYPTED PASSWORD '<пароль
пользователя keycloak>';
CREATE DATABASE "dev-cdr-keycloak" OWNER "dev-cdr-keycloak";
```

2.6.4 Создать в пространстве имён Kubernetes `dev` секрет Kubernetes с паролем администратора Keycloak и данными для подключения Keycloak к СУБД PostgreSQL:

```
kubectl -n dev create secret generic cdr-keycloak-secret \
  --from-literal=admin-password=<пароль администратора Keycloak> \
  --from-literal=postgresql-user=dev-cdr-keycloak \
  --from-literal=postgresql-password=<пароль пользователя keycloak>
```

2.6.5 Создать YAML-файл `keycloak-values.yaml`, содержащий параметры настройки Helm-чарта Keycloak [2-3] для развёртывания одиночного сервера Keycloak с хранением данных в ранее развёрнутой СУБД PostgreSQL:

```
nameOverride: cdr-keycloak
auth:
  adminUser: admin
  existingSecret: "cdr-keycloak-secret"
  passwordSecretKey: "admin-password"
replicaCount: 1
production: true
postgresql:
  enabled: false
externalDatabase:
  host: "postgres-0.postgres.default.svc.cluster.local"
  port: 5432
  database: "dev-cdr-keycloak"
  existingSecret: "cdr-keycloak-secret"
  existingSecretUserKey: "postgresql-user"
  existingSecretPasswordKey: "postgresql-password"
proxy: edge
httpRelativePath: /auth/
ingress:
  enabled: true
  ingressClassName: "nginx"
  hostname: dev-frontend.cdr.usetech.ru
  path: /auth/
  pathType: Prefix
resources:
  requests:
    memory: "512Mi"
    cpu: "500m"
  limits:
    memory: "4Gi"
    cpu: "2"
metrics:
  enabled: true
  serviceMonitor:
    enabled: true
    labels:
      release: "monitoring"
```

2.6.6 Установить Helm-чарт Keycloak:

```
helm repo add bitnami https://charts.bitnami.com/bitnami && \
helm repo update && \
helm upgrade --install --namespace=dev \
  --version 23.0.0 \
  --values keycloak-values.yaml \
  cdr-keycloak bitnami/keycloak
```

2.6.7 Убедиться, что развёртывание Keycloak успешно завершилось:

```
kubectl -n dev get all
```

2.6.8 Вывести на консоль пароль администратора Keycloak:

```
kubectl -n dev get secrets keycloak-secret \
  -o jsonpath={.data.admin-password} |base64 -d
```

2.6.9 Проверить доступность веб-интерфейса администратора Keycloak и возможность входа с ранее заданным паролем администратора.

> ВНИМАНИЕ: Для доступа к Keycloak приложений, развёрнутых в Kubernetes, следует использовать адрес: http://cdr-keycloak.dev.svc.cluster.local/.

2.6.10 Добавить в Grafana дашборд: https://grafana.com/grafana/dashboards/19659-keycloak-metrics-dashboard/.

3 Развёртывание приложения "ЦДР" на стенде dev в Minikube

3.1 Подготовка Kubernetes для развёртывания стенда dev

3.1.1 Создать пространство имён Kubernetes dev:

```
kubectl create namespace dev
```

3.1.2 Подготовить файлы закрытого ключа и fullchain-сертификата TLS для домена *.cdr.usetech.ru, затем создать секрет Kubernetes, содержащий сертификат TLS, который будет использоваться ingress-ом для защиты доступа к подсистемам приложения ЦДР по сети:

```
kubectl -n dev create secret tls cdr-ingress-tls \
  --key="../path/to/privkey.pem" \
  --cert="../path/to/fullchain.pem"
```

3.2 Подготовка к развёртыванию подсистемы "configurator"

3.2.1 Вывести на консоль пароль пользователя PostgreSQL, хранящийся в секрете Kubernetes postgres-secret:

```
kubectl get secrets postgres-secret -o jsonpath={.data.password} |base64 -d
```

3.2.2 Подключиться к БД PostgreSQL:

```
kubectl exec -it postgres-0 -- psql -h localhost -U postgres --password -p
5432
```

3.2.3 Создать базу данных и пользователя `dev-configurator` в СУБД PostgreSQL:

```
CREATE USER "dev-configurator" WITH ENCRYPTED PASSWORD '<пароль
пользователя configurator>';
CREATE DATABASE "dev-configurator" OWNER "dev-configurator";
```

3.2.4 Создать секрет Kubernetes с данными для подключения подсистемы "configurator" к СУБД PostgreSQL:

```
kubectl -n dev create secret generic configurator-secret \
  --from-literal=DATASOURCE_USERNAME=dev-configurator \
  --from-literal=DATASOURCE_PASSWORD=<пароль пользователя dev-configurator>
```

3.3 Подготовка к развёртыванию подсистемы "riskssolver"

3.3.1 Вывести на консоль имя пользователя и пароль администратора MinIO:

```
kubectl -n minio-cdr get secrets minio-cdr-secret \
  -o jsonpath={.data."config\.env"} |base64 -d
```

3.3.2 Войти в веб-интерфейс тенанта MinIO и создать бакет `dev-riskssolver`, пользователя `dev-riskssolver` с полномочиями на чтение и запись, а также ключи доступа (service account) для означенного пользователя. Ключи необходимо сохранить - они потребуются на следующем шаге.

3.3.3 Создать секрет Kubernetes с данными для подключения подсистемы "riskssolver" к объектному хранилищу MinIO:

```
kubectl -n dev create secret generic riskssolver-secret \
  --from-literal=S3_ACCESS_KEY=<ключ доступа minio dev-riskssolver> \
  --from-literal=S3_SECRET_KEY=<секретный ключ minio dev-riskssolver>
```

3.4 Подготовка к развёртыванию подсистемы "frontend"

3.4.1 Посдсистема "frontend" на стенде `rosreestr` доступна из сети Интернет, поэтому при развёртывании на означенном стенде необходимо ограничить доступ к подсистеме при помощи аутентификации HTTP basic.

3.4.2 Создать файл с именем `auth`, содержащий данные для HTTP basic auth фронтенда приложения ЦДР на стенде `rosreestr`:

> ВНИМАНИЕ: имя файла изменять нельзя!

```
htpasswd -b -c auth rosreestr Luvx0Iv0w8gdDVc2v
```

3.4.3 В пространстве имён вышеозначенного стенда создать секрет Kubernetes `frontend-basic-auth` с данными для HTTP basic auth:

```
kubectl -n <namespace> create secret generic \
  frontend-basic-auth --from-file=auth
```

3.5 Развёртывание подсистем приложения ЦДР из Helm-чарта

3.5.1 Подсистемы приложения ЦДР разворачиваются автоматически при сборке новых версий подсистем посредством пайплайна GitLab CI/CD и Helm-чарта. Helm-чарт развёртывания приложения ЦДР находится в репозитории Helm-чарта.

Первое развёртывание вышеозначенных подсистем необходимо выполнить вручную для того, чтобы задать начальные значения параметров развёртывания, таких как: веб-адрес, по которому будет осуществляться доступ к подсистеме, настройки доступа к БД и хранилищу S3, а также настройки взаимодействия подсистем между собой. Последовательность действий для выполнения первоначального развёртывания подсистем приложения ЦДР описана ниже в данном разделе.

3.5.2 Создать файл `values-dev.yaml` с параметрами развёртывания Helm-чарта приложения ЦДР на стенде `dev`:

```
imagePullSecrets:
 - name: "usetech-registry"

configurator:
  enabled: true
  enableJavaDebug: true
  env:
    - name: DATASOURCE_URL
      value: "jdbc:postgresql://postgres-
0.postgres.default.svc.cluster.local:5432/dev-configurator"
    - name: DATASOURCE_USERNAME
      valueFrom:
        secretKeyRef:
          name: configurator-secret
          key: DATASOURCE_USERNAME
          optional: false
    - name: DATASOURCE_PASSWORD
      valueFrom:
        secretKeyRef:
```

```
          name: configurator-secret
          key: DATASOURCE_PASSWORD
          optional: false
  strategy:
    type: RollingUpdate
  resources:
    requests:
      cpu: 100m
      memory: 1400Mi
    limits:
      cpu: 1
      memory: 6000Mi
  ingress:
    enabled: true
    ingressClassName: nginx
    hosts:
      - dev-configurator.cdr.usetech.ru
    path: /
    pathType: Prefix
    tls:
      - secretName: cdr-ingress-tls
        hosts:
          - '*.cdr.usetech.ru'

modeling:
  enabled: false
  enableJavaDebug: true
  strategy:
    type: RollingUpdate
  resources:
    requests:
      cpu: 100m
      memory: 1400Mi
    limits:
      cpu: 1
      memory: 3000Mi
  ingress:
    enabled: true
    ingressClassName: nginx
    hosts:
      - dev-modeling.cdr.usetech.ru
    path: /
    pathType: Prefix
    tls:
      - secretName: cdr-ingress-tls
        hosts:
          - '*.cdr.usetech.ru'

riskssolver:
  enabled: true
  enableJavaDebug: true
  env:
    - name: S3_ENDPOINT
      value: "http://minio.minio-cdr.svc.cluster.local/"
```

```
      - name: S3_INSECURE
        value: "true"
      - name: S3_BUCKET_NAME
        value: "dev-riskssolver"
      - name: S3_ACCESS_KEY
        valueFrom:
          secretKeyRef:
            name: riskssolver-secret
            key: S3_ACCESS_KEY
            optional: false
      - name: S3_SECRET_KEY
        valueFrom:
          secretKeyRef:
            name: riskssolver-secret
            key: S3_SECRET_KEY
            optional: false
    strategy:
      type: RollingUpdate
    resources:
      requests:
        cpu: 100m
        memory: 800Mi
      limits:
        cpu: 1
        memory: 1Gi
    ingress:
      enabled: true
      ingressClassName: nginx
      hosts:
        - dev-riskssolver.cdr.usetech.ru
      path: /
      pathType: Prefix
      tls:
        - secretName: cdr-ingress-tls
          hosts:
            - '*.cdr.usetech.ru'

  frontend:
    enabled: true
    strategy:
      type: RollingUpdate
    resources:
      requests:
        cpu: 100m
        memory: 100Mi
      limits:
        cpu: 500m
        memory: 300Mi
    ingress:
      enabled: true
      ingressClassName: nginx
      hosts:
        - dev-frontend.cdr.usetech.ru
      path: /
```

```
    pathType: Prefix
    tls:
      - secretName: cdr-ingress-tls
        hosts:
          - '*.cdr.usetech.ru'
```

3.5.3 При развёртывании на стенде `rosreestr` необходимо ограничить доступ к подсистеме `frontend` при помощи аутентификации HTTP basic. Для этого следует добавить в файл параметров развёртывания Helm-чарта подраздел `.frontend.ingress.annotations` следующего содержания:

```
      nginx.ingress.kubernetes.io/auth-type: basic
      nginx.ingress.kubernetes.io/auth-secret: frontend-basic-auth
      nginx.ingress.kubernetes.io/auth-realm: 'Authentication Required'
```

3.5.4 В веб-интерфейсе GitLab в меню `Settings -> Repository -> Deploy Tokens` проекта `cdr/helm` создать токен развёртывания с именем `deploy-token-helm`, именем пользователя `deploy-token-helm` и полномочиями `read_package_registry`. Сохранить токен - он потребуется в дальнейшем для доступа к репозиторию Helm-чарта.

3.5.5 Добавить репозитоий Helm и установить Helm-чарт приложения ЦДР:

```
helm repo add \
  --username deploy-token-helm --password <токен> \
  cdr-develop \
  https://gitlab.usetech.ru/api/v4/projects/644/packages/helm/develop && \
helm install --create-namespace --namespace dev \
  -f values-dev.yaml cdr cdr-develop/cdr
```

ПРИМЕЧАНИЕ: При развёртывании на других стендах, например, на стенде `rosreestr`, следует заменить в командах `helm` все вхождения слова `dev` на `rosreestr`.

3.4.6 Проверить готовность подсистем приложения ЦДР:

```
helm -n dev test cdr
```

Пример вывода вышеозначенной команды:

```
NAME: cdr
LAST DEPLOYED: Fri Aug 23 12:14:16 2024
NAMESPACE: dev
STATUS: deployed
REVISION: 1
TEST SUITE:     cdr-configurator-test-connection
Last Started:   Fri Aug 23 12:16:58 2024
```

```
Last Completed: Fri Aug 23 12:17:04 2024
Phase:          Succeeded
TEST SUITE:     cdr-frontend-test-connection
Last Started:   Fri Aug 23 12:17:04 2024
Last Completed: Fri Aug 23 12:17:09 2024
Phase:          Succeeded
TEST SUITE:     cdr-modeling-test-connection
Last Started:   Fri Aug 23 12:17:09 2024
Last Completed: Fri Aug 23 12:17:14 2024
Phase:          Succeeded
TEST SUITE:     cdr-riskssolver-test-connection
Last Started:   Fri Aug 23 12:17:14 2024
Last Completed: Fri Aug 23 12:17:18 2024
Phase:          Succeeded
...
```

> ПРИМЕЧАНИЕ: Проверка может быть неуспешной, если она производится сразу после развёртывания приложения ЦДР, так как запуск подсистем занимает продолжительное время. В таком случае следует повторить проверку через некоторое время.

3.5.7 При необходимости обновление подсистем приложения ЦДР может быть выполнено вручную:

```
helm repo update cdr-develop && \
helm upgrade -n dev --reset-then-reuse-values cdr cdr-develop/cdr
```

4 Перечень источников

1 **Kube Prometheus Stack**

2 **Bitnami Keycloak**

3 **Keycloak on Kubernetes: A Comprehensive Guide to Deploying and Scaling Your Identity and Access Management Solution**

4 **MinIO - Deploy Operator With Helm**

5 **MinIO - Deploy a MinIO Tenant with Helm**