АКЦИОНЕРНОЕ ОБЩЕСТВО
«СИСТЕМНЫЙ ОПЕРАТОР ЕДИНОЙ ЭНЕРГЕТИЧЕСКОЙ СИСТЕМЫ»

ИНСТРУКЦИЯ ПО УСТАНОВКЕ И НАСТРОЙКЕ

программного обеспечения мониторинга функционирования устройств и комплексов системы мониторинга переходных режимов

Москва, 2022

СОДЕРЖАНИЕ

Перечень сокращений

Таблица 1. Перечень сокращений

Сокращение	Описание или расшифровка
API	Интерфейс программирования приложений (англ. Application programming interface) - набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах
PGP	preety good privacy, инструмент для шифрования
АС «АИП»	Автоматизированная система «Автоматизированная интеграционная платформа»
БД	База данных
ПО МФУК	Программное обеспечение мониторинга функционирования устройств и комплексов системы мониторинга переходных режимов
ПО ФЭП	Программное обеспечение формирования экспресс-протокола о достоверности и качестве данных системы мониторинга переходных режимов
НСИ	Нормативно-справочная информация
ОС	Операционная система
ПО	Программное обеспечение

1 Состав ПО МФУК для установки

Программное обеспечение МФУК состоит из следующих элементов:

- системное ПО;
- Docker-образы компонентов;
- конфигурации.

1.1 Системное ПО

Системное ПО обеспечивает среду запуска приложения и состоит из следующих элементов:

- ОС Astra Linux;
- среда выполнения приложения Docker с возможностью запуска контейнеров приложений в виртуальной среде;
- ПО оркестровки (взаимодействия и порядка запуска) и обеспечения работы контейнеров как единого приложения – Kubernetes;
- K9s актуальной версии;
- СУБД Postgres Pro Ent;
- ПО Keepalived.

1.2 Docker-образы компонентов

Docker-образы компонентов представляют собой отдельные функции приложения и по сути представляют собой основу для запуска виртуальной среды конкретного компонента в виде контейнера. Состав образов в рамках ПО МФУК следующий:

- postgresql - образ для запуска контейнера базы данных PostgreSQL;
- hazelcast - компонент кластеризации и взаимодействия контейнеров приложения между собой;
- map-server - компонент предоставляющий карту для отображения в интерфейсе на форме мониторинга;
- nsi-service - компонент обработки и хранения НСИ;
- aip-model-ws - компонент взаимодействия с АИП;
- configuration-manager - центральный компонент по обработке логики в рамках ПО МФУК и одновременно предоставляет REST API для пользовательского интерфейса;

– configuration-manager-ui - пользовательский интерфейс приложения.

1.3 Конфигурации

Конфигурации описывают связи между компонентами, необходимые им ресурсы и связь с внешними ресурсами. Поставляются в виде отдельного архива и подлежат редактированию при установке. Подробнее состав и назначение каждого редактируемого параметра в конфигурационных файлах описан в разделе установки ПО МФУК в среде Kubernetes.

Состав файлов следующий:

– mfuk-namespace.yaml - файл определения пространства имен приложения;

– app-config.yaml - набор конфигурационных файлов с параметрами системы;

– mfuk-stateful-deployment.yaml - набор контейнеров для запуска. На основе образов конфигурирует запуск контейнеров приложения;

– mfuk-service.yaml - сервисы приложения, которые связывают между собой контейнеры приложения;

– mfuk-pv.yaml - привязка директорий сервера к директориям контейнеров для сохранения данных (БД и др.);

– mfuk-storage-class.yaml - файл описания класса хранилища.

2 Системные требования

Для функционирования системы требуется три сервера (виртуальные или физические) в следующем составе:

1. Сервер БД/приложений:
   - 4х ядерный CPU с архитектурой x86-64;
   - 16 Гб RAM;
   - 800 Гб дискового пространства;
2. Сервер БД/приложений:
   - 4х ядерный CPU с архитектурой x86-64;
   - 16 Гб RAM;
   - 800 Гб дискового пространства.
3. Сервер БД:
   - 4х ядерный CPU с архитектурой x86-64;
   - 8 Гб RAM;
   - 400 Гб дискового пространства.

3 Установка системного ПО

Необходимо установить следующее системное ПО, необходимое для функционирования системы:

– Astra Linux "Орел" версии 2.12 и выше;

– Docker версии не ниже 19.03;

– Kubernetes актуальной версии (1.19 на данный момент);

– K9s актуальной версии.

3.1 Установка Astra Linux

Установка описана в официальной документации, расположенной на сайте производителя - https://astralinux.ru/assets/docs/AstraLinuxCE_install_2-12.pdf.

3.2 Установка Docker

Установка docker для операционной системы Astra Linux ничем не отличается от установки на ОС Debian, описанной по адресу https://docs.docker.com/engine/install/debian/ .

Для этого надо установить (если не установлено) ПО для интеграции с https-репозиториями для пакетного менеджера apt (все последующие команды - команды командной строки bash):

```
$ sudo apt-get update
```

```
$ sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
gnupg-agent \
software-properties-common
```

Далее нужно установить официальный PGP ключ репозитория производителя docker:

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add
-
```

Далее добавьте репозиторий docker’a:

```
$ sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/debian \
$(lsb_release -cs) \
stable"
```

После чего произведите обновление доступных пакетов и осуществите их установку:

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Проверка корректной установки docker осуществляется запуском тестового контейнера:

```
$ sudo docker run hello-world
```

После его запуска будет отображено сообщение «hello world» в командной строке и контейнер завершит свою работу. Это значит, что docker установлен успешно.

3.3 Установка Kubernetes

Установка Kubernetes осуществляется согласно официальной документации https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/

Установка осуществляется в несколько шагов:

1. Установка kubeadm (утилита администрирования Kubernetes), kubelet (рабочий сервис kubernetes, осуществляющий его функционирование), kubectl (клиент командной строки Kubernetes).
2. Инициализация кластера и узлов кластера.
3. Настройка клиента для доступа к кластеру.

В случае если по каким-то причинам на сервере отсутствует выход в интернет и, как следствие, нужные образы для kubernetes не смогут автоматически загрузиться из сети, их стоит загрузить в docker явно из архивов.

3.3.1Предварительная установка образов в режиме offline

Производится в случае отсутствия прямого выхода в интернет на сервере.

Для этих целей образы выгружены в отдельный архив, в котором находятся архивы образов, - kubernetes-images.tar.

После разархивирования самого архива, необходимо загрузить образы для функционирования kubernetes в репозиторий Docker'а для дальнейшей возможности запуска контейнеров:

```
$ sudo docker load -i k8s.gcr.io-coredns.tar

$ sudo docker load -i k8s.gcr.io-etcd.tar

$ sudo docker load -i k8s.gcr.io-pause.tar

$ sudo docker load -i k8s.gcr.kube-apiserver.tar

$ sudo docker load -i k8s.gcr.kube-controller-manager.tar

$ sudo docker load -i k8s.gcr.kube-proxy.tar

$ sudo docker load -i k8s.gcr.kube-scheduler.tar

$ sudo docker load -i flannel.tar
```

3.3.2Установка kubeadm, kubelet, kubectl

Сначала нужно добавить PGP-ключ репозитория с Kubernetes и сам репозиторий:

```
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key
add -

$ cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list

deb https://apt.kubernetes.io/ kubernetes-xenial main

EOF
```

Далее установить сами дистрибутивы:

```
$ sudo apt-get update

$ sudo apt-get install -y kubelet kubeadm kubectl

$ sudo apt-mark hold kubelet kubeadm kubectl
```

3.4 Установка первого узла (master) kubernetes

На первом узле производится создание кластера с мастером kubernetes. При наличии второго узла кластера он должен быть инициализирован добавлением к кластеру (см. добавление второго узла).

3.4.1Инициализация кластера и узлов кластера

Инициализация осуществляется посредством команды (с указанием внутренней подсети, которая далее будет использоваться при создании сетевой модели на основе flannel):

```
$ kubeadm init --pod-network-cidr=XX.XXX.X.X/XX
```

3.4.2Настройка клиента для доступа к кластеру

Далее необходимо сконфигурировать доступ к кластеру посредством копирования сгенерированной конфигурации доступа администратором к кластеру:

```
$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

После этого необходимо завершить настройку кластера посредством установки драйвера сети для Kubernetes:

```
$ kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
```

Либо при отсутствии доступа в интернет использовать файл flannel.yml отдельного архива, в котором находятся архивы образов, - kubernetes-images.tar:

```
$ kubectl apply -f kube-flannel.yml
```

Далее задайте возможность запуска pod’ов на master’е:

```
$ kubectl taint nodes --all node-role.kubernetes.io/master-
```

3.5 Добавление второго узла в кластер

На master (первом узле) создаем токен для авторизации на подключение к кластеру (токен действует 24 часа):

```
sudo kubeadm token create
```

Выводим его на экран и запоминаем:

```
kubeadm token list
```

Его вид примерно следующий:

```
jy7a6u.xv7sryvgd6bnu4w7
```

Далее извлекаем хэш из ssl/tls-сертификата мастера для валидации на стороне подключаемого узла по токену:

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -
outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.* //'
```

Его вид примерно следующий:

```
675de2819e94b0e166a63013058be6b19b582e0785040adb8a6e171bc9b677eb
```

Смотрим адрес доступа к мастеру (API endpoint):

```
kubectl cluster-info
```

Результат будет примерно следующий:

```
Kubernetes master is running at https://XXX.XXX.XXX.XXX:XXXX

KubeDNS is running at https:// XXX.XXX.XXX.XXX:XXXX /api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy

Metrics-server is running at https://
XXX.XXX.XXX.XXX:XXXX/api/v1/namespaces/kube-system/services/https:metrics-
server:/proxy
```

Далее на стороне подключаемого узла делаем инициализацию рабочей ноды, указывая адрес API, токен и хэш ssl-сертификата мастера:

```
    kubeadm join <API endpoint> --token <token> --discovery-token-ca-cert-hash
sha256:<hash>
```

Пример на основе параметров выше:

```
    kubeadm join XXX.XXX.XXX.XXX:XXXX --token jy7a6u.xv7sryvgd6bnu4w7 --
discovery-token-ca-cert-hash
sha256:675de2819e94b0e166a63013058be6b19b582e0785040adb8a6e171bc9b677eb
```

Теперь на стороне master можно проверить, что узел успешно подключился к кластеру:

```
    kubectl get nodes
```

Должно отобразиться два узла - мастер и вновь добавленный рабочий узел.

Если необходимо работать с кластером через консоль и на добавленном узле - скопируйте файл $HOME/.kube/config из мастера (ранее созданный) в соответствующую директорию на добавленном узле.

3.5.1 Возможные проблемы

При старте pod’ов на новом узле могут возникнуть проблемы (отраженные в логах) вида:

```
    network for pod "xxxxx": NetworkPlugin cni failed to set up pod "xxxxx"
network: failed to set bridge addr: "cni0" already has an IP address different
from 10.x.x.x/xx
```

Это означает, что было подключение на узле, которое ранее уже было в кластере (другом) инициализировано. И была создана подсеть этого узла (cni0), которая теперь конфликтует с ожидаемой подсетью для текущего узла.

Ожидаемую подсеть можно посмотреть следующим образом:

```
    $ cat /run/flannel/subnet.env
```

Результат будет примерно следующий (FLANNEL_SUBNET - ожидаемая подсеть этого узла):

```
    FLANNEL_NETWORK=XX.XXX.X.X/XX

FLANNEL_SUBNET=XX.XXX.X.X/XX

FLANNEL_MTU=XXXX
```

```
FLANNEL_IPMASQ=true
```

Посмотреть реально существующую подсеть можно следующим образом:

```
sudo ifconfig cni0
```

При их различии достаточно удалить cni0 и через несколько минут она будет заново создана автоматически с уже нужной подсетью:

```
ifconfig cni0 down
ip link delete cni0
```

3.6 Установка k9s

K9s является утилитой командной строки для возможности просмотра и навигации по namespace'ам в Kubernetes, просмотра ресурсов и анализа работы систем, установленных в Kubernetes. Работает поверх утилиты kubectl и облегчает работу с ним.

Установка k9s описана по адресу https://github.com/derailed/k9s и осуществляется следующим образом:

```
$ apt-get update
$ apt-get install k9s
```

3.6.1 Установка посредством ручного скачивания дистрибутива

Также можно установить k9s посредством явного скачивания и установки дистрибутива, если по каким-либо причинам через пакетный миенеджер установка недоступна.

Для этого необходимо зайти на страницу https://github.com/derailed/k9s/releases и скачать k9s_Linux_x86_64.tar.gz. После этого распаковать и файл k9s скопировать и сделать доступным для выполнения:

```
$ sudo cp k9s /usr/bin
$ sudo chmod +x /usr/bin/k9s
```

4 Установка образов в docker

Далее необходимо загрузить образы приложения в репозиторий Docker’а для дальнейшей возможности запуска контейнеров в kubernetes:

```
$ sudo docker load -i hazelcast.tar

$ sudo docker load -i postgres.tar

$ sudo docker load -i nsi-service.tar

$ sudo docker load -i aip-model-ws.tar

$ sudo docker load -i configuration-manager.tar

$ sudo docker load -i configuration-manager-ui.tar

$ sudo docker load -i map-server.tar
```

5 Настройка параметров приложения

Большая часть настроек приложения относится к внутренним связям между контейнерами и не требует дополнительной настройки, однако имеется ряд параметров в yaml-файлах приложения, которые требуют привязки к внешним ресурсам и, как следствие, их предварительную настройку перед установкой.

Указанные настройки можно разделить на следующие три вида:

- привязка к ресурсам диска хостового сервера (месту на жестком диске);
- привязка к сетевым портам хостового сервера (для обращения извне);
- привязка к адресам внешних систем.

Привязка к ресурсам сервера требуется по причине того, что контейнеры по умолчанию выполняются в полностью изолированной среде со своей файловой системой и сетевой инфраструктурой. Чтобы связать их с реальными внешними ресурсами - необходимо привязать виртуальные ресурсы к внешним по отношению к контейнерам.

5.1 Привязка к директории диска хостового сервера

В ОС должна быть создана директория /mnt/datastorage для хранения данных системы. Рекомендуется для этого выделить отдельный диск и сделать mount в эту директорию. Например, через команду mount:

```
sudo mount /dev/<sddN> /mnt/datastorage
```

Привязка описана в файле mfuk-pv.yaml для ресурса kubernetes вида PersistentVolume с наименованием postgres-mfuk-pv-volume - задается привязка к хостовому дисковому ресурсу вида:

```
  hostPath:
path: "/mnt/datastorage/mfuk-postgres-storage"
```

Сохранение всех данных БД приложения будет производиться в указанную директорию.

5.2 Привязка к сетевым портам хостового сервера

Все связи внутри приложения, также как и вообще доступ к любому из контейнеров, осуществляется через ресурсы Kubernetes типа Service. Все сервисы приложения описаны в файле mfuk-service.yaml.

Наружу выставлены два сервиса - configuration-manager-ui-web и configuration-manager-web:

– configuration-manager-web - доступ к серверной части приложения, представляющей собой REST API для пользовательского интерфейса. По умолчанию привязан к порту 31001 хостового сервера;

– configuration-manager-ui-web - доступ к пользовательскому интерфейсу приложения - представляет собой веб-приложение. По умолчанию привязан к порту 31004 хостового сервера.

5.3 Привязка к адресам внешних систем

Доступ к внешним системам описан через конфигурационные файлы приложения, которые находятся в файле app-config.yaml.

Все конфигурации представлены в виде Kubernetes ресурсов типа ConfigMap.

В конфигурации mfuk-config находятся две data-секции, требующие задания внешних связей.

5.3.1 aip-model-ws.config

В data-секции aip-model-ws.config задаются следующие настройки доступа к АИП:

```
aip.model.ws.service-url=http://<host>:<port>
aip.model.ws.credentials.username=<user>
aip.model.ws.credentials.password=<password>
aip.model.ws.credentials.domain=CDU
```

Где:

– aip.model.ws.service-url - url веб-сервиса АИП;

– aip.model.ws.credentials.username - пользователь, под которым будет производиться запрос (без доменного суффикса или префикса);

– aip.model.ws.credentials.password - пароль пользователя;

– aip.model.ws.credentials.domain - домен, которому принадлежит пользователь (в авторизационной нотации).

5.3.2 configuration-manager.config

В data-секции configuration-manager.config задаются следующие настройки доступа к ПО ФЭП:

```
configuration-manager.fep.service-url=http://<host>:<port>/
configuration-manager.fep.auth.access-token-uri=http://<host>:<port>/oauth/token
configuration-manager.fep.auth.username=<user>
configuration-manager.fep.auth.password=<password>
configuration-manager.default-admin-user=<user>
configuration-manager.security.auth-type=ldap
configuration-manager.ldap.url=ldap://<domen>:<port>
configuration-manager.ldap.domain=<domen>
configuration-manager.ldap.base=<base path>
configuration-manager.ldap.userDn=<full dn>
configuration-manager.ldap.password=<password>
```

Где:

- configuration-manager.fep.service-url - url REST. API ПО ФЭП;
- configuration-manager.fep.auth.access-token-uri - url для получения токена авторизации. Стандартно - "configuration-manager.fep.service-url + /oauth/token";
- configuration-manager.fep.auth.username - пользователь, под которым будет производиться запрос;
- configuration-manager.fep.auth.password - пароль пользователя;
- configuration-manager.default-admin-user - дефолтный администратор в системе, который будет иметь возможность настройки системы при первом ее запуске. Соответствует sAMAccountName из AD;
- configuration-manager.security.auth-type - тип аутентификации - для LDAP использовать настройку равную значению "ldap";
- configuration-manager.ldap.url - url LDAP-сервера (AD);
- configuration-manager.ldap.domain - домен AD;
- configuration-manager.ldap.base - базовый путь в AD для поиска пользователей;
- configuration-manager.ldap.userDn - полный DN-путь пользователя для аутентификации самого приложения в AD;
- configuration-manager.ldap.password - пароль пользователя для аутентификации самого приложения в AD.

5.3.3 configuration-manager-ui-config

В ресурсе configuration-manager-ui-config для data-секции env-specific.json требуется указать следующие настройки:

```
"backendhost": "http:// <host>:<port>"
"mapurl": "http:// <host>:<port>/map-server/images/osm_tiles/{z}/{x}/{y}.png"
"oauth2": {
```

```
        "endpoint": "http:// <host>:<port>/oauth"
}
```

Где:

– "backendhost" - внешний адрес к серверному REST API. Необходимо явное указание внешнего доступа, т.к. он в последующем отправляется пользователю как ресурс клиентского приложения;

– "mapurl" - адрес сервера карт. Используется для отображения географической карты на форме мониторинга;

– "oauth2:endpoint" - адрес к API выдачи токенов для аутентификации.

5.4 Настройка Kerberos аутентификации

В системе есть возможность настроить аутентификацию посредством схемы Negotiate для HTTP с поддержкой Kerberos. Для этого необходимо выпустить ketyab файл.

Сначала создается spn для сервисного аккаунта, под которым будет работать веб-сервер, с именем HTTP/<host>@<имя домена> - имя хоста содержит полное имя с доменом. После генерируется keytab-файл с информацией о spn для веб-сервера.

Создание spn для пользователя, от которого работает веб-сервер из командной строки:

```
    setspn -a HTTP/<host>@<domen> <user name>
```

Пролистываем список spn для пользователя в целях убедиться, что spn был задан:

```
    setspn -l <user name>
```

Далее генерируем mfuk.keytab файл для заданного spn:

```
    ktpass -princ HTTP/<host>@<domen>O -pass <password>  -mapuser <user name> -
crypto ALL -ptype KRB5_NT_PRINCIPAL -out mfuk.keytab
```

Полученный keytab-файл нужно перенести на сервер приложений для подключения к configuration-manager по следующему пути:

```
    /mnt/datastorage/mfuk-share-storage/kerberos.keytab
```

Также следует внести изменения в файлы настроек:

5.4.1 configuration-manager.config

Добавить/изменить следующие настойки

```
configuration-manager.kerberos.service-principal-name=<SPN>
configuration-manager.kerberos.keytab-location=/etc/opt/kerberos.keytab
configuration-manager.jwt.redirect-uri=<uri>
```

Где:

- configuration-manager.kerberos.service-principal-name - назначенный SPN. Обычно имеет вид HTTP/<хост http сервиса в корпоративной сети>@<имя домена>;
- configuration-manager.kerberos.keytab-location - полный путь до .keytab файла внутри контейнера (/etc/opt/kerberos.keytab);
- configuration-manager.jwt.redirect-uri - адрес на который вернёт backend после успешной аутентификации.

5.4.2 configuration-manager-ui-config

Добавить следующие настройки (не стирая предыдущие для oauth2):

```
      "oauth2": {
    ...
        "redirectUri": "http://<host>:<port>/monitoring",
        "authType": "kerberos"
}
```

Где:

- "oauth2:authType" - тип аутентификации;
- "oauth2:redirectUri" - адрес, на который вернёт backend после успешной аутентификации.

5.4.3Настройка доступа к keytab

Для того, чтобы контейнер configuration-manager получил доступ к keytab файлу, нужно сделать его подключение через persistent-volume с доступом к файловой структуре сервера.

Нужно в конфигурации mfuk-stateful-deployment.yaml добавить подключение файла следующим образом, переопределив у configuration-manager volumeMounts на следующий:

```
      volumeMounts:
         - name: config-app
           mountPath: /etc/opt/ch
           readOnly: true
```

```
        - name: share
          mountPath: /etc/opt/kerberos.keytab
          subPath: kerberos.keytab
```

и volumes на:

```
      - name: share
          persistentVolumeClaim:
            claimName: share-mfuk-pv-claim
        - name: config-app
          configMap:
            name: mfuk-config
            items:
            - key: "configuration-manager.config"
              path: "application.properties"
```

Тем самым мы определили подключение файла keytab внутрь контейнера configuration-manager по пути /etc/opt/kerberos.keytab из поставщика файловой части системы извне share-mfuk-pv-claim.

В конфигурации mfuk-pv.yaml необходимо добавить share-mfuk-pv-claim с привязкой к директории с keytab-файлом:

```
     ---
kind: PersistentVolume
apiVersion: v1
metadata:
  name: share-mfuk-pv-volume
  namespace: mfuk
  labels:
    type: local
spec:
  storageClassName: kube-volumes
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/mnt/datastorage/mfuk-share-storage"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: share-mfuk-pv-claim
  namespace: mfuk
spec:
  storageClassName: kube-volumes
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 5Gi
```

Данной настройкой мы обеспечили привязку файла kerberos.keytab из директории /mnt/datastorage/mfuk-share-storage хостового сервера в файл

/etc/opt/kerberos.keytab внутри самого контейнера configuration-manager, доступ к которому указали в configuration-manager.config (внутри app-config.yaml).

Далее необходимо применить конфигурации (следующий раздел).

6 Установка Kubernetes-приложения

Приложение представляет собой набор контейнеров и связей между ними в виде сервисов, а также наборов конфигураций.

Установка заключается в установке описательных файлов приложения, которые описывают параметры запуска всего приложения на основе образов и конфигурационных параметров. Состав файлов, следующий:

– mfuk-namespace.yaml - файл определения пространства имен приложения;

– app-config.yaml - набор конфигурационных файлов с параметрами системы;

– mfuk-stateful-deployment.yaml - набор контейнеров для запуска. На основе образов конфигурирует запуск контейнеров приложения;

– mfuk-service.yaml - сервисы приложения, которые связывают между собой контейнеры приложения;

– mfuk-pv.yaml - привязка директорий сервера к директориям контейнеров для сохранения данных (БД и др.);

– mfuk-storage-class.yaml - файл описания класса хранилища.

Установка приложения осуществляется посредством применения следующих команд командной строки (предварительно необходимо установить параметры связи с внешними системами согласно разделу Настройка параметров приложения):

```
$ kubectl apply -f mfuk-namespace.yaml

$ kubectl apply -f app-config.yaml

$ kubectl apply -f mfuk-service.yaml

$ kubectl apply -f mfuk-storage-class.yaml

$ kubectl apply -f mfuk-pv.yaml

$ kubectl apply -f mfuk-stateful-deployment.yaml
```

6.1 Проверка корректности работы и диагностика

Проверка состоит из двух этапов – статуса каждого из контейнеров приложения, а также доступности веб-интерфейса.

Для проверки статусов каждого из контейнеров приложения необходимо запустить утилиту k9s, выбрать namespace mfuk и проверить, что у всех контейнеров в пространстве mfuk стоит статус Running.

Для проверки веб-интерфейса необходимо зайти по адресу https://<общий адрес МФУК> и проверить доступность веб-интерфейса.

Также при выборе конкретного контейнера можно открыть логи и увидеть факт успешности или неуспешности запуска приложения (в случае отсутствия записей с началом в строке ERROR запуск можно считать удачным).

7 Создание ingress контроллера

Ingress - сервис связи внутренних сервисов в Kubernetes с внешней сетью для целей централизованного управления внешними url-ми сервисов и ssl-сертификатами для них. В рамках системы устанавливается привязка к статическому ip, при наличии нескольких ip - должно быть несколько привязок (файлов с конфигурацией из static-ip-svc.yml).

Для этого необходимо предварительно загрузить образ ingress-nginx:

```
$ docker load -i ingress-nginx.tar
```

Также предварительно надо сгенерировать ssl-сертификат (или взять уже готовый):

```
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout tls.key -
out tls.crt -subj "/CN=<host>/O=<host>"
```

После чего установить его в kubernetes с именем tls-secret в default namespace:

```
$ kubectl create secret --namespace=mfuk tls tls-secret --key tls.key --cert
tls.crt
```

Далее нужно установить и настроить ingress в Kubernetes. Для этого предусмотрены следующие три файла:

– static-ip-svc.yml - привязка к статическому адресу сервера для связи с ingress;

– nginx-ingress-controller.yml - установка самого ingress-контроллера в kubernetes, что позволит связывать внутренние сервисы с внешней сетью;

– mfuk-nginx-ingress.yml - настройка привязок внутренних сервисов к внешним адресам. Для backend и frontend.

```
$ kubectl create -f static-ip-svc.yml
$ kubectl apply -f nginx-ingress-controller.yml
$ kubectl create -f mfuk-nginx-ingress.yml
```

Содержимое файла nginx-ingress-controller.yml описывает контроллер и не подлежит изменению.

Содержимое файла static-ip-svc.yml описывает привязку внешнего адреса и портов. Его вид следующий:

```
      kind: Service
      apiVersion: v1
      metadata:
        name: nginx-ingress-ext
        namespace: ingress-nginx
        labels:
          app.kubernetes.io/component: controller
          app.kubernetes.io/instance: ingress-nginx
          app.kubernetes.io/name: ingress-nginx
      spec:
        ports:
          - name: http
            protocol: TCP
            port: 80
            targetPort: http
          - name: https
            protocol: TCP
            port: 443
            targetPort: https
        selector:
          app.kubernetes.io/component: controller
          app.kubernetes.io/instance: ingress-nginx
          app.kubernetes.io/name: ingress-nginx
        type: ClusterIP
        externalIPs:
          - <host1>
          - <host2>
```

Где loadBalancerIP задает адрес, который будет обрабатываться ingress контроллером. Также указаны два порта для обработки - 80 и 443.

Содержимое файла nginx-ingress.yml описывает доступ к внутренним сервисам и имеет следующий вид:

```
      apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-nginx-ui
  namespace: mfuk
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
spec:
  tls:
  # This assumes tls-secret exists.
  - secretName: tls-secret
  rules:
```

```
  - http:
      paths:
      - path: /
        backend:
          # This assumes http-svc exists and routes to healthy endpoints.
          serviceName: configuration-manager-ui-web
          servicePort: 80

---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-nginx-backend
  namespace: mfuk
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
spec:
  tls:
  # This assumes tls-secret exists.
  - secretName: tls-secret
  rules:
  - http:
      paths:
      - path: /backend(/|$)(.*)
        backend:
          # This assumes http-svc exists and routes to healthy endpoints.
          serviceName: configuration-manager-web
          servicePort: 8080
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-nginx-map
  namespace: mfuk
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
spec:
  tls:
  # This assumes tls-secret exists.
  - secretName: tls-secret
  rules:
  - http:
      paths:
      - path: /map(/|$)(.*)
        backend:
          # This assumes http-svc exists and routes to healthy endpoints.
          serviceName: map-server-web
          servicePort: 8080
```

Где secretName: tls-secret привязывает к ранее созданному tls-сертификату, а в paths привязываем пути в url к соответствующим сервисам и их портам из приложения - configuration-manager-ui-web для ui, configuration-manager-web для backend и map-server-web для map-server.

После этого необходимо заменить в env-specific.json backendhost на backendhost=https://<url>/backend, oauth2:endpoint на endpoint=https://<url>/backend/oauth и mapurl на https://<url>/map/map-

server/images/osm_tiles/{z}/{x}/{y}.png. и заново применить конфигурацию (kubectl apply -f) и перезапустить контейнеры приложения.

8 Кластеризация БД

Все компоненты в системе по умолчанию кластеризуемы за исключением БД. Кластеризация БД осуществляется через использование внешней БД Postgres.

Все измененные ниже yaml-файлы необходимо применить через команду kubectl apply -f <file.yaml>

8.1 Установка и настройка кластера Postgres Pro

Для настройки кластера Postgres Pro в режиме мультимастера необходимо воспользоваться инструкцией по настройке соответствующего дополнительно модуля - https://postgrespro.ru/docs/enterprise/12/multimaster.

Также необходимо указать в настройках Postgres Pro параметр listen_addresses равным 0.0.0.0 для того, чтобы БД могла принимать запросы по общему адресу кластера от сервисов системы.

В секциях nsi-service.config и configuration-manager.config файла app-config.yaml необходимо установить свойства spring.datasource.username, spring.datasource.password, spring.datasource.url в значения для БД - соответственно логин, пароль, адрес БД (общий адрес кластера) с указанием инстанса (обычно postgres):

```
spring.datasource.username=postgres

spring.datasource.password=postgres

spring.datasource.url=jdbc:postgresql://<общий адрес МФУК>/<инстанс>
```

После чего необходимо обновить конфигурацию приложения командой:

```
kubectl apply -f app-config.yaml
```

После чего перезапустить сервисы МФУК в kubernetes.

9 Настройка общего ip-адреса в кластере

Настройка осуществляется только на кластерной инсталляции с двумя узлами (в ИА).

Для возможности использования на двухузловом кластере общего ip-адреса (для возможности обращения по общему dns-имени) необходимо установить сервис keepalived для управления общим ip-адресом (его назначение одному из доступных узлов) в кластере посредством протокола VRRP. Установка следующим образом:

```
sudo apt-get update

sudo apt-get install keepalived
```

Далее необходимо указать возможность привязки к нескольким ip-адресам при прослушивании запросов (установкой свойства net.ipv4.ip_nonlocal_bind в файл /etc/sysctl.conf):

```
    echo "net.ipv4.ip_nonlocal_bind = 1" >> /etc/sysctl.conf
```

Далее необходимо применить изменения:

```
    sysctl -p
```

После чего необходимо сконфигурировать /etc/keepalived/keepalived.conf

```
    cd /etc/keepalived/

mv keepalived.conf keepalived.conf.org

vi keepalived.conf
```

На первом узле указываем такую:

```
    ! Configuration File for keepalived

vrrp_instance VI_1 {

    state MASTER

    interface eth0

    virtual_router_id 101

    priority 101

    advert_int 5

    authentication {

        auth_type PASS
```

```
        auth_pass 1111
    }
    virtual_ipaddress {
        <host>
    }
}
```

На втором узле указываем такую:

```
     ! Configuration File for keepalived
vrrp_instance VI_1 {
    state SLAVE
    interface eth0
    virtual_router_id 101
    priority 100
    advert_int 5
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        <host>
    }
}
```

Отличие между первым и вторым узлом в приоритете (priority) - если первый узел доступен, то общий адрес будет на нем, иначе - на втором. Сам адрес задается внутри virtual_ipaddress, interface должен содержать наименование сетевого интерфейса (зачастую - eth0), virtual_router_id должен быть одинаковым на обоих узлах, advert_int - количество секунд после которых определяется недоступность другого узла.

После этого запускаем сервис keepalived:

```
     systemctl start keepalived
```

И делаем его автозапуск при старте ОС:

```
systemctl enable keepalived
```

9.1 Проверка корректности настройки.

Проверка назначенных адресов на сетевой интерфейс:

```
ip addr show eth0
```

Проверка логов на предмет изменения ip-адресов:

```
tailf /var/log/syslog
```