

Общество с ограниченной ответственностью «Зетра»

ОГРН 1237700278155 ИНН 7707488152 КПП 770701001

Юр. адрес: 127473, г. Москва, вн.тер.г. муниципальный округ Тверской, пер 1-й Волконский, д. 15, помещение 1/3

тел.: +7 (909) 909-17-73,

e-mail: info@зетра.рф

Инструкция по установке программного обеспечения «Автоматизированная информационная система ZETRAMAN» («ZETRAMAN»)

Москва

2023 г.

Аннотация

Данный документ содержит инструкцию по установке и развертыванию АИС ZETRAMAN (далее Система).

Оглавление

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ.....	4
1. УСТАНОВКА И НАСТРОЙКА.....	5
1.1. Установка необходимых компонентов и дополнительного ПО	5
1.1.1. Кластер хранения данных на базе PostgreSQL.....	5
1.1.2. Установка kubernetes.....	15
1.1.3. Установка и настройка Kafka кластера.....	21
1.2. Установка АИС ZETRAMAN	33
1.2.1. Подготовка СУБД	33
1.2.2. Деплоймент приложения в кластер	34
1.2.3. Список обеспечивающих сервисов АИС ZETRAMAN.....	39
1.3. Схема размещения сервисов	40

Перечень принятых сокращений

АИС - Автоматизированная информационная система

АС - Автоматизированная система

БД - База данных

1. УСТАНОВКА И НАСТРОЙКА

1.1. Установка необходимых компонентов и дополнительного ПО

1.1.1. Кластер хранения данных на базе PostgreSQL

Для хранения данных используется совместимая с PostgreSQL (>v13) система управления базами данных. Процесс установки описан на примере PostgreSQL Patroni Cluster.

Patroni — это автоматическая система аварийного переключения для PostgreSQL. Patroni обеспечивает автоматическое и ручное переключение при отказе и хранит все важные данные в распределенном хранилище конфигурации (DCS) на базе систем ETCD, Consul и т.д.. Соединения приложения и базы данных не осуществляются напрямую, а маршрутизируются через прокси-сервер соединения, такой как HAProxy. Прокси определяет активный/главный узел, который в данный момент времени готов обрабатывать соединения. Использование прокси-сервера сводит к 0 шансы встретить split-brain в кластере баз данных.

Для развертывания сервиса потребуется создать инфраструктуру из 3-х виртуальных машин. 2 VM для кластера Postgresql и 1 для организации точки входа в кластер на базе HAProxy. На все VM устанавливаем ОС из списка в нашем случае это debian.

Потребуется предварительная подготовка:

VM для ReverseProxy RAM: 1 GB CPU: 2 Core HDD: 20 GB

VM для PostgreSQL RAM: 8 GB CPU: 4 Core HDD: 200 GB

1.1.1.1. Установка ETCD

ETCD — это отказоустойчивое распределенное хранилище ключей/значений, используемое для хранения состояния кластера Postgres. Используя Patroni все узлы Postgres обмениваются данными с etcd для поддержания работоспособности кластера Postgres. На момент составления инструкции актуальная версия ETCD 3.5.5.

```
```bash
```

```
URL для загрузки дистрибутива
```

```
GOOGLE_URL=https://storage.googleapis.com/etcd
```

```
GITHUB_URL=https://github.com/etcd-io/etcd/releases/download
```

```
DOWNLOAD_URL=${GITHUB_URL}
```

```
```
```

Удаляем прошлые установки если не первая попытка, и создаем новую папку для установки.

```
```bash
```

```
rm -f /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz
```

```
rm -rf /tmp/etcd-download-test && mkdir -p /tmp/etcd-download-test
```

```
```
```

Скачиваем и распаковываем файлы приложения, удаляем временные файлы

```
``bash
curl -L ${DOWNLOAD_URL}/${ETCD_VER}/etcd-${ETCD_VER}-linux-
amd64.tar.gz -o /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz
tar xzvf /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz -C /tmp/etcd-download-test --
strip-components=1
rm -f /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz
``
```

Проверяем корректность установки

```
``bash
/tmp/etcd-download-test/etcd --version
/tmp/etcd-download-test/etcdctl version
mv /tmp/etcd-download-test/etcd* /usr/local/bin/
``
```

Настраиваем службу ETCD

Создаем пользователя для работы со службой.

```
``bash
sudo groupadd --system etcd
sudo useradd -s /sbin/nologin --system -g etcd etcd
``
```

Создаем и предоставляем права на каталога /var/lib/etcd пользователю etcd.

```
``bash
sudo mkdir -p /var/lib/etcd/
sudo mkdir /etc/etcd
sudo chown -R etcd:etcd /var/lib/etcd/
sudo chmod -R 700 /var/lib/etcd/
``
```

Следующие операции необходимо выполнить на каждом сервере где будет установлен ETCD. Присваиваем переменным значения для формирования файлы автозапуска службы ETCD.

```
``bash
INT_NAME="eth1"
ETCD_HOST_IP=$(ip-адрес show $INT_NAME | grep "inet\b" | awk '{print $2}' | cut
-d/ -f1)
ETCD_NAME=$(hostname -s)
```

```

Где параметры:

1. INT\_NAME — имя сетевого интерфейса, который будет использоваться для трафика кластера. У каждого сервера он может быть свой.
2. ETCD\_HOST\_IP — внутренний IP-адрес указанного сетевого интерфейса. Он используется для обслуживания клиентских запросов и связи с узлами кластера etcd.
3. ETCD\_NAME — каждый элемент etcd должен иметь уникальное имя в пределах кластера etcd. Используемая команда установит имя etcd в соответствии с именем хоста текущего сервера.

Проверяем значения переменных. Если у какой-то из переменной не будет значения, служба автозапуска работать не будет.

```
```bash
echo $INT_NAME
echo $ETCD_HOST_IP
echo $ETCD_NAME
```
```

Создаем файл службы для сервиса ETCD.

```
```bash
cat << EOF > /etc/systemd/system/etcd.service
[Unit]
Description=etcd service
Documentation=https://github.com/coreos/etcd

[Service]
User=etcd
Type=notify
ExecStart=/usr/local/bin/etcd \
--name ${ETCD_NAME} \
--enable-v2=true \
--data-dir /var/lib/etcd \
--initial-advertise-peer-urls http://${ETCD_HOST_IP}:2380 \
--listen-peer-urls http://${ETCD_HOST_IP}:2380 \
--listen-client-urls http://${ETCD_HOST_IP}:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://${ETCD_HOST_IP}:2379 \
--initial-cluster-token etcd-cluster-1 \
```
```

```
--initial-cluster
revproxy=http://10.15.73.170:2380,postgresql1=http://10.15.73.176:2380,postgresql2=http://1
0.15.73.177:2380 \\
--initial-cluster-state new \\
--heartbeat-interval 1000 \\
--election-timeout 5000
Restart=on-failure
RestartSec=5

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

После создания службы на всех серверах, включаем её, первый сервер будет выполнять роль сервера начальной загрузки, после успешного запуска службы на всех серверах, будет выбран лидер.

```
````bash  
systemctl daemon-reload
systemctl enable etcd
systemctl start etcd.service
````
```

Проверить состояние кластера после запуска.

```
````bash  
systemctl status -l etcd.service
etcdctl member list
etcdctl endpoint status --write-out=table
etcdctl endpoint health
````
```

1.1.1.2. Установка PostgreSQL

Подключаем дополнительный репозиторий с дистрибутивами PostgreSQL и добавляем ключ шифрования. Обновляем кеш пакетов:

```
````bash  
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add
```

-



---

```
echo "deb http://apt.postgresql.org/pub/repos/apt/ $(lsb_release -cs)-pgdg main" | sudo tee
/etc/apt/sources.list.d/postgresql-pgdg.list > /dev/null

sudo apt update
...

```

Устанавливаем PostgreSQL, проверяем наличие установленных пакетов.

```
```bash
sudo apt install postgresql-14
dpkg -l | grep postgresql
...

```

Останавливаем службу PostgreSQL если она сама запустилась. Управлением сервисом PostgreSQL будет заниматься Patroni.

```
```bash
sudo systemctl stop postgresql
sudo systemctl disable postgresql
...

```

### **1.1.1.3. Установка Patroni**

Устанавливаем необходимые пакеты.

```
```bash
sudo apt -y install python3 python3-pip
sudo -H pip install --upgrade testresources
sudo -H pip install --upgrade setuptools
sudo -H pip install psycopg2
sudo -H pip install patroni
sudo -H pip install python-etcd
...

```

Создаем файл конфигурации для кластера Patroni /etc/patroni.yml. Конфигурационный файл необходимо создать на всех ВМ с PostgreSQL.

```yaml

scope: postgres

name: postgresql2

restapi:

listen: 0.0.0.0:8008

connect\_address: 10.15.73.177:8008

etcd:

host: revproxy:2379

bootstrap:

dc:

ttl: 30

loop\_wait: 10

retry\_timeout: 10

maximum\_lag\_on\_failover: 1048576

postgresql:

use\_pg\_rewind: true

use\_slots: true

parameters:

wal\_level: replica

hot\_standby: "on"

logging\_collector: "on"

max\_wal\_senders: 5

max\_replication\_slots: 5

initdb:

- encoding: UTF8

- data-checksums

pg\_hba:

- host replication replicator 127.0.0.1/32 trust

- host replication replicator 10.15.73.177/24 md5

- host all all 10.15.73.1/24 md5

- host all all 0.0.0.0/0 md5

users:

admin:

password: \*\*\*\*\*

options:

- createrole

- createdb

postgresql:

listen: 0.0.0.0:5432

connect\_address: 10.15.73.177:5432

data\_dir: "/data/patroni"

pgpass: /tmp/pgpass

authentication:

replication:

username: replicator

password: \*\*\*\*\*

superuser:

username: postgres

password: \*\*\*\*\*

parameters:

unix\_socket\_directories: '/var/run/postgresql'

tags:

nofailover: false

noloadbalance: false

clonefrom: false

nosync: false

```

Проверяем конфигурацию. На всех VM с PostgreSQL.

```bash

```
patroni --validate-config /etc/patroni.yml
...
```

Подготавливаем директорию для работы PostgreSQL.

```
```bash  
sudo mkdir -p /data/patroni  
sudo chown postgres:postgres /data/patroni  
sudo chmod 700 /data/patroni  
...
```

Создаем файл службы для автозапуска Patroni. На всех ВМ с PostgreSQL.

```
```bash  
sudo nano /etc/systemd/system/patroni.service
...
```

Содержимое:

```
```ini  
[Unit]  
Description=High availability PostgreSQL Cluster  
After=syslog.target network.target  
  
[Service]  
Type=simple  
User=postgres  
Group=postgres  
ExecStart=/usr/local/bin/patroni /etc/patroni.yml  
KillMode=process  
TimeoutSec=30  
Restart=no  
  
[Install]  
WantedBy=multi-user.target
```

Запускаем Patroni. Проверяем состояние службы. На всех ВМ с PostgreSQL.

```
```bash
sudo systemctl start patroni
sudo systemctl status patroni

```

#### **1.1.1.4. Установка HAProxy**

HAProxy предоставляет единую точку, к которой подключатся приложения работающие с базой данных. Контроль за узлами входа в кластер баз данных осуществляет HAProxy на основе информации от Patroni. Установка HAProxy.

```
```bash
sudo apt -y install haproxy
***
```

1.1.1.5. Настройка конфигурации для HAProxy

Создаем копии конфигурационного файла для изменения. Редактируем базовый конфигурационный файл под наши требования.

```
```bash
sudo cp -p /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg-orig
sudo nano /etc/haproxy/haproxy.cfg

```

Содержимое файла конфигурации. IP-адреса используем свои.

```
```cfg
global
    maxconn 100

defaults
    log global
    mode tcp
    retries 2
    timeout client 30m
```

```
timeout connect 4s
timeout server 30m
timeout check 5s
```

```
listen stats
  mode http
  bind *:7000
  stats enable
  stats uri /
```

```
listen postgres
  bind *:5000
  option httpchk
  http-check expect status 200
  default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
  server patroni1 10.15.73.176:5432 maxconn 100 check port 8008
  server patroni2 10.15.73.177:5432 maxconn 100 check port 8008
...

```

Проверяем конфигурационный файл на наличие ошибок.

```
```bash
sudo /usr/sbin/haproxy -c -V -f /etc/haproxy/haproxy.cfg
```
```

Запускаем службу. Проверяем состояние.

```
```bash
sudo systemctl restart haproxy
sudo systemctl status haproxy
```
```

Переходим на WEB-интерфейс и проверяем корректность работы. WEB-интерфейс доступен по IP-адресу или имени ВМ на порту 7000.

Настройка кластера Postgresql Patroni завершена, для доступа в кластер используется прокси-сервер HAProxy, для хранения состояние кластера Patroni используется ETCD-хранилище, состоящее из 3-х нод. Patroni полностью управляет переключением и работой кластера Postgresql.

1.1.2. Установка kubernetes

Рекомендуемые настройки продуктового кластера, могут отличаться от данного руководства, выше в данном руководстве описаны системные требования по ресурсам ВМ, рекомендуется для надежности использовать 3 ВМ для мастеров и не менее 4 нод под рабочую нагрузку для работы основных приложений информационной системы ZETRAMAN.

Существует несколько вариантов развертывания кластера Kubernetes. Самый простой из них — Minikube или кластер на одну ноду. Подробный процесс его установки описан на официальном сайте Kubernetes. В данном руководстве рассмотрим установку кластера на нескольких нодах, используя kubespray. Вы можете использовать любой удобный для вас способ развертывания кластера в том числе используя официальную [документацию](#).

****Kubespray**** — это набор Ansible-ролей для установки и конфигурации Kubernetes. Он обеспечивает:

1. кластер высокой доступности
2. поддержку большинства популярных дистрибутивов Linux
3. тесты CI

Для понимания процесса установки рассмотрим структуру кластера. Кластер Kubernetes состоит из двух типов ресурсов:

1. Master отвечает за управление кластером. Мастер координирует все действия в вашем кластере, такие как планирование приложений, поддержание желаемого состояния приложений, масштабирование приложений и развертывание новых обновлений.

2. Node (узел) — это виртуальная машина или физический компьютер, который служит рабочим компьютером в кластере Kubernetes. У каждого узла есть Kubelet, который является агентом для управления узлом и взаимодействия с мастером Kubernetes. Узел также должен иметь инструменты для обработки контейнерных операций, такие как containerd или Docker.

При разворачивании приложения в Kubernetes, мы сообщаем Мастеру, что нужно запустить контейнеры приложений. Мастер планирует запуск контейнеров на узлах кластера. Узлы связываются с мастером с помощью Kubernetes API, который предоставляет Мастер.

1.1.2.1. Подготовка ВМ

Перейдем непосредственно к процессу установки. Подготавливаем несколько виртуальных машин. У нас будет 3 виртуальных машины с ОС Centos 7 minimal. В дальнейших статьях, посвященных Kubernetes, мы будем добавлять виртуальные машины для демонстрации всех возможностей кластера.

Адресация виртуальных машин:

1. k8s-1 ip 10.15.73.41
2. k8s-3 ip 10.15.73.42
3. k8s-3 ip 10.15.73.43

Для каждой виртуальной машины меняем имя командой внутри гостевой ОС. Выполняем команду:

```
```bash
[root@k8s-1 ~]# hostnamectl set-hostname k8s-1
```
```

Здесь k8s-1 — имя первой виртуальной машины, k8s-2 и k8s-3 — второй и третьей соответственно. После применения перегружаем виртуальные машины.

Устанавливаем дополнительное ПО:

```
```bash
[root@k8s-1 ~]# apt install wget curl git screen python-pip sshpass -y
```
```

Генерируем ключ SSH:

```
```bash
[root@k8s-1 ~]# ssh-keygen
```
```

Копируем ключ на удаленные серверы. Выполняем команды:

```
```bash
[root@k8s-1 ~]# ssh-copy-id -i ~/.ssh/id_rsa.pub root@k8s-1
[root@k8s-1 ~]# ssh-copy-id -i ~/.ssh/id_rsa.pub root@k8s-2
[root@k8s-1 ~]# ssh-copy-id -i ~/.ssh/id_rsa.pub root@k8s-3
```
```

Устанавливаем ansible:

```
```
[root@k8s-1 ~]# apt install ansible pip -y
```
```


Редактируем файл хостов для ansible:

```
```bash
[root@k8s-1 ~]# vim /etc/ansible/hosts
```
```

Добавляем строки:

```
```bash
[k8sservers]
k8s-1
k8s-2
k8s-3
```
```

Создаем новый playbook для ansible. Данный playbook необходим для подготовки серверов:

```
```
[root@k8s-1 ~]# vim system-prepare.yml
```
```

Содержимое файла system-prepare.yml:

```
```yaml

- hosts: all
 tasks:

 - name: Disable SELinux
 selinux:
 state: disabled

 - name: Disable SWAP since kubernetes can't work with swap enabled (1/2)
 shell: |
 swapoff -a

 - name: Disable SWAP in fstab since kubernetes can't work with swap enabled (2/2)
 replace:
 path: /etc/fstab
```
```

```
regexp: '^([\^#].*?\sswap\s+sw\s+.*)$'
replace: '# \1'

- name: set timezone to Europe/Moscow
  timezone:
    name: Europe/Moscow

- name: Ensure firewalld service is disabled and stopped
  systemd:
    name: firewalld
    state: stopped
    enabled: no
    masked: yes
    register: firewalld_result
    failed_when: "firewalld_result is failed and 'Could not find the requested service' not in
firewalld_result.msg"
    when: ansible_os_family == 'RedHat' and ansible_distribution_major_version >= '7'

- name: restart server
  shell: 'sleep 1 && shutdown -r now "Reboot triggered by Ansible" && sleep 1'
  async: 1
  poll: 0
  become: true
...

```

Данный файл выполняет следующие действия на всех нодах:

1. Отключает Selinux
2. Отключает Swap
3. Отключает firewalld
4. Меняет timezone

Эти действия необходимы для дальнейшей корректной установки и работы Kubernetes.

Выполняем команду:

```
```bash
[root@k8s-1 ~]# ansible-playbook system-prepare.yml

```

\*\*\*

После выполнения, все серверы автоматически перезагрузятся. Подготовка серверов закончена, можно переходить к установке Kubernetes.

### 1.1.2.2. Установка кластера

Клонируем репозиторий kubespray. Выполняем команду:

```
```bash
[root@k8s-1 ]# git clone https://github.com/kubernetes-sigs/kubespray
***
```

```
```bash
[root@k8s-1]#cd kubespray

```

Выполняем установку необходимых зависимостей:

```
```bash
[root@k8s-1 kubespray ]# pip install -r requirements.txt
***
```

Редактируем файл где описываем хосты и кто какую роль будет выполнять. Так же прописываем IP-адреса, которые будут использоваться:

```
```bash
[root@k8s-1]# vim /root/kubespray/inventory/sample/inventory.ini

```

```
```ini
k8s-1 ansible_ssh_host=10.15.73.41 ip=10.15.73.41
k8s-2 ansible_ssh_host=10.15.73.42 ip=10.15.73.42
k8s-3 ansible_ssh_host=10.15.73.43 ip=10.15.73.43
```

```
[kube-master]
```

```
k8s-1
```

```
[etcd]
```

```
k8s-1
```

k8s-2

k8s-3

[kube-node]

k8s-2

k8s-3

[k8s-cluster:children]

kube-node

kube-master

...

Меняем устанавливаемую версию не ниже 1.24. Редактируем файл:

```
```bash
```

```
[root@k8s-1]# vim /root/kubespray/inventory/sample/group_vars/k8s-cluster/k8s-
cluster.yml
```

```
...
```

Меняем следующее значение:

```
```bash
```

```
## Change this to use another Kubernetes version, e.g. a current beta release
```

```
kube_version: v1.27.4
```

```
...
```

Запускаем установку кластера:

```
```bash
```

```
[root@k8s-1]# cd /root/kubespray
```

```
[root@k8s-1 kubespray]# ansible-playbook -u root -b -i inventory/sample/inventory.ini
cluster.yml
```

```
...
```

Во время не должно быть ошибок. Если же они возникли, то необходимо исправить найденные проблемы и запустить установку кластера снова.

После установки проверяем информацию по кластеру (должно выдавать kubernetes master is running at ...):

```
```bash
[root@k8s-1 kubespray]# kubectl cluster-info
```
```

Проверяем статус узлов (покажется список узлов, роль и статус Ready с таймингом работы):

```
```bash
[root@k8s-1 kubespray]# kubectl get nodes
```
```

Проверяем статус подов:

```
```bash
[root@k8s-1 ]# kubectl get pod -A
```
```

На этом базовая установка k8s закончена. Дополнительную информацию по kubespray смотрите в официальной [документации](<https://kubespray.io/#/>)

Если вы устанавливаете кластер иным способом необходимо дополнительно установить nginx-ingress-controller и local-storage-provisioner

### 1.1.3. Установка и настройка Kafka кластера

В качестве шины данных используются продукты, совместимые с Apache Kafka. В данном руководстве рассматривается установка Kafka кластер из 3-х серверов

Действия описанные ниже выполняем на всех 3-х серверах.

#### 1.1.3.1. Добавляем правила Firewall для Kafka и Zookeeper

```
```
sudo vim /etc/firewalld/services/zooKeeper.xml
```
```

```
```
sudo vim /etc/firewalld/services/zooKeeper.xml
```
```

```
```xml
<?xml version="1.0" encoding="utf-8"?>
```

```
<service>
<short>ZooKeeper</short>
<description>Firewall rule for ZooKeeper ports</description>
<port protocol="tcp" port="2888"/>
<port protocol="tcp" port="3888"/>
<port protocol="tcp" port="2181"/>
</service>
```

```
...
```

```
...
```

```
sudo nano /etc/firewalld/services/kafka.xml
```

```
...
```

```
```bash
```

```
sudo nano /etc/firewalld/services/kafka.xml
```

```
...
```

```
```xml
```

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<service>
```

```
<short>Kafka</short>
```

```
<description>Firewall rule for Kafka port</description>
```

```
<port protocol="tcp" port="9092"/>
```

```
</service>
```

```
...
```

1.1.3.2. *Активируем правила*

```
```bash
```

```
sudo service firewalld restart
```

```
sudo firewall-cmd --permanent --add-service=zooKeeper
```

```
sudo firewall-cmd --permanent --add-service=kafka
```

```
sudo service firewalld restart
```

```
...
```

### **1.1.3.3. Создание пользователей**

```
```bash
sudo adduser kafka
sudo passwd kafka
```
```

### **1.1.3.4. Устанавливаем Java**

```
```bash
apt install java-1.8.0-openjdk
```
```

### **1.1.3.5. Установка kafka**

Скачиваем последнюю версию Kafka

```
```
wget https://downloads.apache.org/kafka/3.6.0/kafka_2.12-3.6.0.tgz
```
```

Распаковываем

```
```bash
tar -xzf kafka_2.12-3.6.0.tgz
```
```

Перемещаем в /opt

```
```
mv kafka_2.12-3.6.0 /opt/kafka
```
```

Создаем каталоги для логов Kafka и для zooKeeper

```
```
mkdir -p /opt/kafka/zookeeper/data
mkdir -p /opt/kafka/kafka-logs
```
```

\*\*\*

### 1.1.3.6. Конфигурация zookeeper

Переходим к конфигурации zooKeeper, открываем файл с конфигурацией

```
```bash
vim /opt/kafka/config/zookeeper.properties
***
```

и указываем директорию с данными:

```
```conf
dataDir=/opt/kafka/zookeeper/data

```

Сервера и лимиты синхронизации:

```
```conf
server.1=kafka1.dev.local:2888:3888
server.2=kafka2.dev.local:2888:3888
server.3=kafka3.dev.local:2888:3888
initLimit=5
syncLimit=2
***
```

Далее, на каждом сервере создаем свой id для zooKeeper

```
```bash
echo "1" > /opt/kafka/zookeeper/data/myid (для сервера kafka1.dev.local)
echo "2" > /opt/kafka/zookeeper/data/myid (для сервера kafka2.dev.local)
echo "3" > /opt/kafka/zookeeper/data/myid (для сервера kafka3.dev.local)

```

### 1.1.3.7. Настройка kafka

Редактируем конфиг сервера

```

vim /opt/kafka/config/server.properties

```



Добавляем:

```
broker.id=1 (для каждого сервера свой 1,2,3)
```

директорию с логами

```
```conf
log.dirs=/opt/kafka/kafka-logs
```
```

указываем прослушиватели:

```
```conf
listeners=PLAINTEXT://:9092
advertised.listeners=PLAINTEXT://kafka1.dev.local:9092
```
```

`listeners` — используется для внутреннего траффика между нодами кластера

`advertised.listeners` — используется для клиентского траффика

указываем ноды zooKeeper:

```
```conf
zookeeper.connect=kafka1.dev.local:2181,kafka2.dev.local:2181,kafka3.dev.local:2181
```
```

Для поддержки удаления топиков, включите опцию ниже:

```
```conf
delete.topic.enable=true
```
```

Далее меняем владельца на пользователя `kafka`

```
```bash
chown -R kafka /opt/kafka
```
```

### 1.1.3.8. Сервисы `systemd` для `zooKeeper` и `kafka`

#### 1.1.3.8.1. Сервис zooKeeper

```
```bash
vim /etc/systemd/system/zookeeper.service
```

```ini
[Unit]
Description=ZooKeeper for Kafka
After=network.target
[Service]
Type=forking
User=kafka
Restart=on-failure
LimitNOFILE=16384:16384
ExecStart=/opt/kafka/bin/zookeeper-server-start.sh
/opt/kafka/config/zookeeper.properties -daemon
[Install]
WantedBy=multi-user.target
```
```

#### 1.1.3.8.2. Сервис Kafka

```
```bash
vim /etc/systemd/system/kafka.service
```

```ini
[Unit]
Description=Kafka Broker
After=network.target
After=zookeeper.service
[Service]
Type=forking
User=kafka
SyslogIdentifier=kafka (%i)
Restart=on-failure
LimitNOFILE=16384:16384
```

```
ExecStart=/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties  
[Install]  
WantedBy=multi-user.target  
...  

```

После создания файлов загружаем информацию о новых сервисах и включаем их

```
```bash  
systemctl daemon-reload
systemctl enable zookeeper
systemctl enable kafka
...

```

Перезагружаем сервера и убеждаемся что сервисы запущены

Логинимся под пользователем kafka

```
```bash  
su kafka  
...  

```

переходим в каталог с kafka

```
```bash  
cd /opt/kafka
...

```

Проверим что zooKeeper работает корректно и все ноды видят друг друга

```
```bash  
bin/zookeeper-shell.sh localhost:2181 ls /brokers/ids  
...  

```

вывод должен быть таким:

```
```bash  
Connecting to localhost:2181
WATCHER::
WatchedEvent state:SyncConnected type:None path:null

```

[1, 2, 3]

...

### 1.1.3.9. Проверка корректности работы

Проверим корректность работы Kafka, на текущей ноде запустим Producer, обратите внимание, что топик будет создан автоматически

```
```bash
bin/kafka-console-producer.sh --broker-list kafka1:9092,kafka2:9092,kafka3:9092 --topic
example-topic
```
```

На другой ноде, например на kafka2, запустим Consumer

```
```bash
su kafka
cd /opt/kafka
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic example-topic
```
```

Теперь на запущенном Producer вводим сообщения, они должны появиться на ноде с запущенным Consumer

Посмотреть список созданных топиков можно следующей командой:

```
```bash
bin/kafka-topics.sh --list --zookeeper localhost:2181
```
```

Получить информацию о созданном топике можно следующей командой:

```
```bash
bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic example-topic
```
```

Удалить топик можно следующей командой:

```
```bash
bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic example-topic
```
```

\*\*\*

Для удаления топика необходимо добавить `delete.topic.enable=True` в конфиг сервера Kafka

Создать топик с определенными параметрами можно следующей командой:

```
```bash
bin/kafka-topics.sh --create \
--zookeeper localhost:2181 \
--topic <topic-name> \
--partitions <number-of-partitions> \
--replication-factor <number-of-replicating-servers>
```
```

Также можно выполнить более тонкую настройку топика при его создании, например:

```
```bash
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 3 -
-config min.insync.replicas=1 --config retention.ms=-1 --config
unclean.leader.election.enable=false --topic test-topic
```
```

Изменить параметры топика можно следующей командой (изменим количество партиций и параметр `**min.insync.replicas**`):

```
```bash
bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic test-topic --partitions 8 --
config min.insync.replicas=2
```
```

Посмотреть кол-во открытых соединений к Kafka можно командой:

```
```bash
netstat -anp | grep :9092 | grep ESTABLISHED | wc -l
```
```

Узнать размер топика можно командой:

```
```bash
bin/kafka-log-dirs.sh --bootstrap-server localhost:9092 --topic-list 'test-topic' --describe |
grep '^{' | jq '[ ..|.size? | numbers ] | add'
```
```

\*\*\*

Посмотреть список активных consumer можно командой:

```
```bash
bin/kafka-consumer-groups.sh --list --bootstrap-server localhost:9092
```

Получив список клиентов можно посмотреть более подробную информацию о consumer, в примере ниже клиент console-consumer-70311

```
```bash
bin/kafka-consumer-groups.sh --describe --group console-consumer-70311 --bootstrap-server localhost:9092
```

\*\*\*

### 1.1.3.10. Тестирование kafka на отказоустойчивость

Для этого сгенерируем текстовый файл в 10 000 000 строк, эти строки мы запишем в топик kafka, далее примем сообщения consumer и сохраним их в текстовый файл.

Во время заливки мы отключим одну из 3-х нод kafka и в конце сравним, файл на consumer и оригинальный файл.

Поскольку kafka гарантирует доставку сообщений, то файлы должны совпадать.

Создадим топик million

```
```bash
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 3 -
-config min.insync.replicas=2 --config retention.ms=-1 --config
unclean.leader.election.enable=false --topic million
```

Создадим тестовый файл

```
```bash
touch generate
vim generate
```

\*\*\*

```
```bash
#/bin/bash
for i in {1..10000000};
```

```
do
echo $i
echo $i >> tokafka.txt
...

```

```
```bash
chmod +x generate
./generate
...

```

Тестовый файл с сообщениями готов.

Теперь запускаем команду для заливки данных

```
```bash
cat tokafka.txt | /opt/kafka/bin/kafka-console-producer.sh --producer.config
client.propertiesssl --broker-list
kafka1.dev.local:9093,kafka2.dev.local:9093,kafka3.dev.local:9093 --topic million
...

```

теперь на сервере с consumer запускаем сам consumer с выводом в файл

```
```bash
/opt/kafka/bin/kafka-console-consumer.sh --consumer.config client.propertiesssl --
bootstrap-server kafka1.dev.local:9093 --topic million > from-kafka.txt
...

```

Запускаем скрипт Producer-а , видим что пошла заливка данных и через некоторое время отключаем любую из нод сервера kafka

После завершения заливки данных проверим файл from-kafka.txt ,командой:

```
```bash
wc -l from-kafka.txt
...

```

число строк в выводе должно быть 10000000.

Недоступность одного из брокеров не повлияла на отправку/доставку сообщений.

Проверим состояние топика во время недоступности одной из нод:

```
```bash
bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic million
```
```

Вывод будет следующим:

```
```bash
Topic:million PartitionCount:3 ReplicationFactor:3 Configs:retention.ms=-
1,unclean.leader.election.enable=false,min.insync.replicas=2
Topic: million Partition: 0 Leader: 1 Replicas: 1,2,3 Isr: 1,2
Topic: million Partition: 1 Leader: 2 Replicas: 2,3,1 Isr: 2,1
Topic: million Partition: 2 Leader: 1 Replicas: 3,1,2 Isr: 1,2
```
```

Как видим число синхронных реплик — 1 (Isr), все партиции доступны и у каждой из них есть лидер.

В данном случае топик был настроен правильно, т.к. мы указали следующие параметры:

`min.insync.replicas` — кол-во реплик, на которые должны быть синхронизированы данные, прежде чем записаться.

`unclean.leader.election.enable=false` отключение возможности провести failover на не синхронную отстающую реплику с потенциальной потерей данных

`ReplicationFactor:3` — кол-во реплик, на которые реплицируются данные.

После восстановления отключенной реплики видим что данные успешно синхронизированы

```
```bash
Topic:million PartitionCount:3 ReplicationFactor:3 Configs:retention.ms=-
1,unclean.leader.election.enable=false,min.insync.replicas=2
Topic: million Partition: 0 Leader: 1 Replicas: 1,2,3 Isr: 1,2,3
Topic: million Partition: 1 Leader: 2 Replicas: 2,3,1 Isr: 2,1,3
Topic: million Partition: 2 Leader: 1 Replicas: 3,1,2 Isr: 1,2,3
```
```

На этом этапе подготовка кластера завершена и можно приступать к развертыванию приложения ZETRAMAN в кластер k8s

1.2. Установка АИС ZETRAMAN

Установка выполняется из предварительно собранного helm-пакета АИС нужной версии с необходимыми параметрами.

1.2.1. Подготовка СУБД

Перед развертыванием приложения необходимо создать базы данных для всех сервисов в нашем кластере patroni

Создание СУБД при помощи скрипта:

Создадим скрипт на одной из машин кластера patroni

```
```bash
```

```
vim /opt/scripts/create_db.sh
```

```
```
```

```
```bash
```

```
#!/usr/bin/env bash
```

```
pg_db_list+=(
```

```
 'addresses'
```

```
 'broadcaster'
```

```
 'emergency'
```

```
 'file-storage'
```

```
 'kc'
```

```
 'monitoring'
```

```
 'notification'
```

```
 'order-manager'
```

```
 'pusk'
```

```
 'risk-agent'
```

```
 'risk-simulator-manager'
```

```
 'users'
```

```
 'weather-manager'
```

```
 'xwiki'
```

```
)
```

```
create_pg_databases(){
```

```
for pg_db in "${pg_db_list[@]}"; do
 echo "Create postgres database \"${pg_db}\""
 "createdb -U ${POSTGRES_USER} -O ${POSTGRES_USER} ${pg_db} || true"
 psql -U ${POSTGRES_USER} ${pg_db} -c 'CREATE EXTENSION IF NOT
EXISTS postgis'
done
}
create_pg_databases
...

```

сделаем скрип исполняемым и выполним его:

```
```bash
chmod +x /opt/scripts/create_db.sh
/opt/scripts/create_db.sh
...

```

1.2.2. Деплоймент приложения в кластер

На машине на которой будет выполняться установка должен быть доступ к кластеру кubernetes, а так же установлены утилиты kubectl и helm

1.2.2.1. Установка сервиса подготовки конфигурации

Распакуйте архив чарта для подготовки конфигураций

```
```bash
tar -xzf prepare-configs.tar.gz /opt/charts/
...

```

Структура архива:

```
```bash
|-- helm
|   |-- base
|       |-- Chart.yaml
|       |-- README.md
|       |-- charts
|           |-- config-service
|               |-- Chart.yaml
|               |-- files
|               |-- config-service.tar.bzip2.base64
|               |-- environment.yaml
|               |-- kub-all-connection.yml
|               |-- templates
|               |-- _init_containers.yaml

```

```

|         |         |         | -- configmap-all.yaml
|         |         |         | -- configmap-config.yaml
|         |         |         | -- configmap-env.yaml
|         |         |         | -- configmap-services.yaml
|         |         |         | -- gitea-credentials-secret.yaml
|         |         |         | -- service.yaml
|         |         |         | -- statefulset.yaml
|         |         |         | -- values.yaml
|         |         |         | -- templates
|         |         |         |   | -- _configmap-config.yaml
|         |         |         |   | -- _configmap-env.yaml
|         |         |         |   | -- _service.yaml
|         |         |         |   | -- _statefullset.yaml
|         |         |         |   | -- values.yaml
|         |         |         | -- payload.sh
|         |         |         | --

```

Архив содержит данные сервиса конфигураций, который подготавливает конфигурационные файлы, и его требуется установить в первую очередь с помощью скрипта:

```

```bash
/opt/charts/prepare-configs/payload.sh
```

```

После установки проверим что сервис у нас установлен

```

```bash
kubectl get pods -n kub-conf-prep
NAME READY STATUS RESTARTS AGE
config-service-0 1/1 Running 0 19h
```

```

1.2.2.2. Установка прикладных сервисов АИС ZETRAMAN

Распакуйте архив чарта

```

```bash
tar -xzvf kub.tar.gz /opt/charts/
```

```

Архив содержит набор chart для прикладных сервисов.

Все настройки чарта хранятся в файле values.yaml:

```

```yaml
global:
 add_teleport_acls: true
 asterisk_host: ""
 busybox_image: "tools/busybox:1.32.0"
 config_service_url: "http://config-service:7999"
```

```

```
create_local_path_storage: false
create_pv: false
create_secret: false
db_host: "192.168.100.54"
db_password: "passwd_here"
db_port: "5432"
db_user: "admin"
default_timezone: Europe/Moscow
deployment_update_strategy: RollingUpdate
elasticsearch_host: "elasticsearch"
env: "dev"
geo_url: ""
host_aliases:
  - ip: 192.168.100.250
    hostnames:
      - git-dev.k3s.zetra.space
      - front-dev.k3s.zetra.space
      - auth-kub-dev.k3s.zetra.space
      - addresses-manager-kub-dev.k3s.zetra.space
      - config-service-kub-dev.k3s.zetra.space
      - emergency-manager-kub-dev.k3s.zetra.space
      - gateway-kub-dev.k3s.zetra.space
      - gitea-kub-dev.k3s.zetra.space
      - order-manager-kub-dev.k3s.zetra.space
      - pusik-kub-dev.k3s.zetra.space
      - users-kub-dev.k3s.zetra.space
      - xwiki-kub-dev.k3s.zetra.space
imagePullSecrets: regcred
images:
  local_path_provisioner: rancher/local-path-provisioner:v0.0.22
k8s_front_fqdn: "front-dev.k3s.zetra.space"
k8s_front_port: "80"
k8s_front_scheme: "http"
kafka_host: "192.168.100.54"
kafka_port: "9092"
liveness_delay: 120
```

```
location: "dev"
main_domain: "k3s.zetra.space"
minio_access_key: "root"
minio_host: "192.168.100.54"
minio_port: "9001"
minio_secret_key: "your_secret"
project: "kub"
readiness_delay: 120
redis_host: "redis"
redis_port: "6379"
registry: "registry.zetra.space"
replicas: 1
statefullset_update_strategy: RollingUpdate
storageClassName: local-path
use_ingress_location:
wait_for_image: "tools/groundnutty/k8s-wait-for:v1.3"
zookeeper_host: "192.168.100.54"
zookeeper_port: "2181"
nodes:
- k3s-w01.zetra.space
- k3s-w02.zetra.space
- k3s-w03.zetra.space
- k3s-w04.zetra.space
```

```
infra:
  storage: "15Gi"
```

```
image: {}
...

```

В данном файле необходимо изменить настройки доступа к кластеру patroni на ваши

```
```yaml
db_host: "192.168.100.54"
db_password: "passwd_here"
db_port: "5432"
db_user: "admin"
```

\*\*\*

Так же заменить адрес сопоставления алиасов внутри кластера на ваш ingress LB service адрес и fqdn сервисов на ваше доменное имя

```
```bash
host_aliases:
  - ip: 192.168.100.250
***

```yaml
k8s_front_fqdn: "kub.mydomain.com"

```

Так же заменить параметры доступа к zookeeper и kafka на ваши

```
```yaml
kafka_host: "192.168.100.54"
kafka_port: "9092"
zookeeper_host: "192.168.100.54"
zookeeper_port: "2181"
***
```

Если необходимо можете исправить дополнительные настройки по вашему усмотрению

После всех необходимых настроек можно приступать к установке приложения

Для этого выполним скрипт payload.sh

```
```bash
/opt/charts/kub/payload.sh

```

После установки проверим что сервисы у нас установились

```
```bash
kubectl get pods -n kub
***
```

Получаем список запущенных pod прикладных сервисов.

После того как вы убедились, что все поды в состоянии `Running` можно зайти в систему по адресу, который вы указали при установке в параметре `k8s_front_fqdn`

1.2.3. Список обеспечивающих сервисов АИС ZETRAMAN

| Название сервиса | Описание |
|------------------|--|
| gitea | Система контроля версий, служит для хранения конфигураций всех сервисов, а также как единая база для discovery |
| redis | Используется для кеширования данных для прикладных сервисов |
| web | Прокси сервер на базе nginx, обеспечивает маршрутизацию к конечным сервисам, используется как точка входа |
| xwiki | Система хранения прикладной документации |

1.3. Схема размещения сервисов

