ИНСТРУКЦИЯ

по установке программного комплекса
«UDR»

Листов xx

2025

Содержание

1 Введение ... 3
1.1 Аппаратные требования к хостам ... 5
1.2 Подготовка хостов к установке ... 5
1.3 Подготовка операционной системы RedOS хостов ... 5
1.4 Подготовка сервера Nexus ... 7
2 Развертывание кластера K8S ... 9
2.1 Общая подготовительная часть для Master и Worker хостов ... 9
2.2 Развертывание Master хостов ... 15
2.3 Развертывание Worker хостов ... 26
3 Развертывание UDR ... 27
3.1 Установка базы данных Redis ... 27
3.2 Установка компонентов мониторинга (Graphana) ... 30
3.3 Установка сервисов UDR ... 41
3.4 Проверка и тестирование UDR после установки ... 42
4 История изменений ... 43
Список сокращений ... 43

Список рисунков

Рисунок 1-1 Диаграмма развертывания UDR ... 4
Рисунок 2-1 Конфигурация интерфейсов ens32, ens33, ens35 на “worker” хосте ... 10
Рисунок 2-2 Проверка ip_vs параметров нв хосте ... 15
Рисунок 2-3 Проверка доступа к Nexus ... 16
Рисунок 2-4 Проверка прав доступа для файла check_apiserver.sh ... 19
Рисунок 2-5 Графический интерфейс Nexus ... 20
Рисунок 2-6 Инсталляция K8S на “master-0” хосте ... 21
Рисунок 2-7 Процедура подключения к K8S кластеру управляющих “master” хостов ... 22
Рисунок 2-8 Успешный запуск сервиса Calico на K8S кластере ... 23
Рисунок 2-9 Сервис core-dns находится в состоянии RUNNING ... 23
Рисунок 2-10 Хосты K8S кластера находятся в состоянии Ready ... 24
Рисунок 2-11 Проверка параметра ipvs на всех kube-proxy pod K8S кластера ... 24
Рисунок 2-12 Настройка параметров толерантности ... 25

Рисунок 3-1 Высокоуровневая архитектура ПК UDR 27

Список таблиц

Таблица 1-1 Требования к хостам 5
Таблица 4-1 История изменений 43

1 Введение

Документ содержит детальные инструкции по установке программного комплекса UDR (User Data Repository). Персонал, осуществляющий установку ПК UDR, должен обладать знаниями и навыками работы с операционными системами семейства Linux.

Общая диаграмма развертывания UDR показана на Рисунке 1 1.

Для функциорирования ПК UDR необходимо развернуть следующие ресурсы:

1. Kubernetes (K8S) кластер в составе трех хостов типа “master” и трех хостов типа “worker”;
2. Хост с базой данных Redis;
3. Хост с компонентами для мониторинга;
4. Хост с Nexus для функций программного репозитория.

В состав ПК UDR входит СУБД Redis с открытой лицензией (open-source edition), являющаяся разрешенным к использованию составным компонентом программного обеспечения при внесении в реестр Минцифры России.

В состав ПК UDR входит операционная система RedOS. RedOS добавлена в Единый реестр российских программ для электронных вычислительных машин и баз данных под номером 3751.

Общая схема

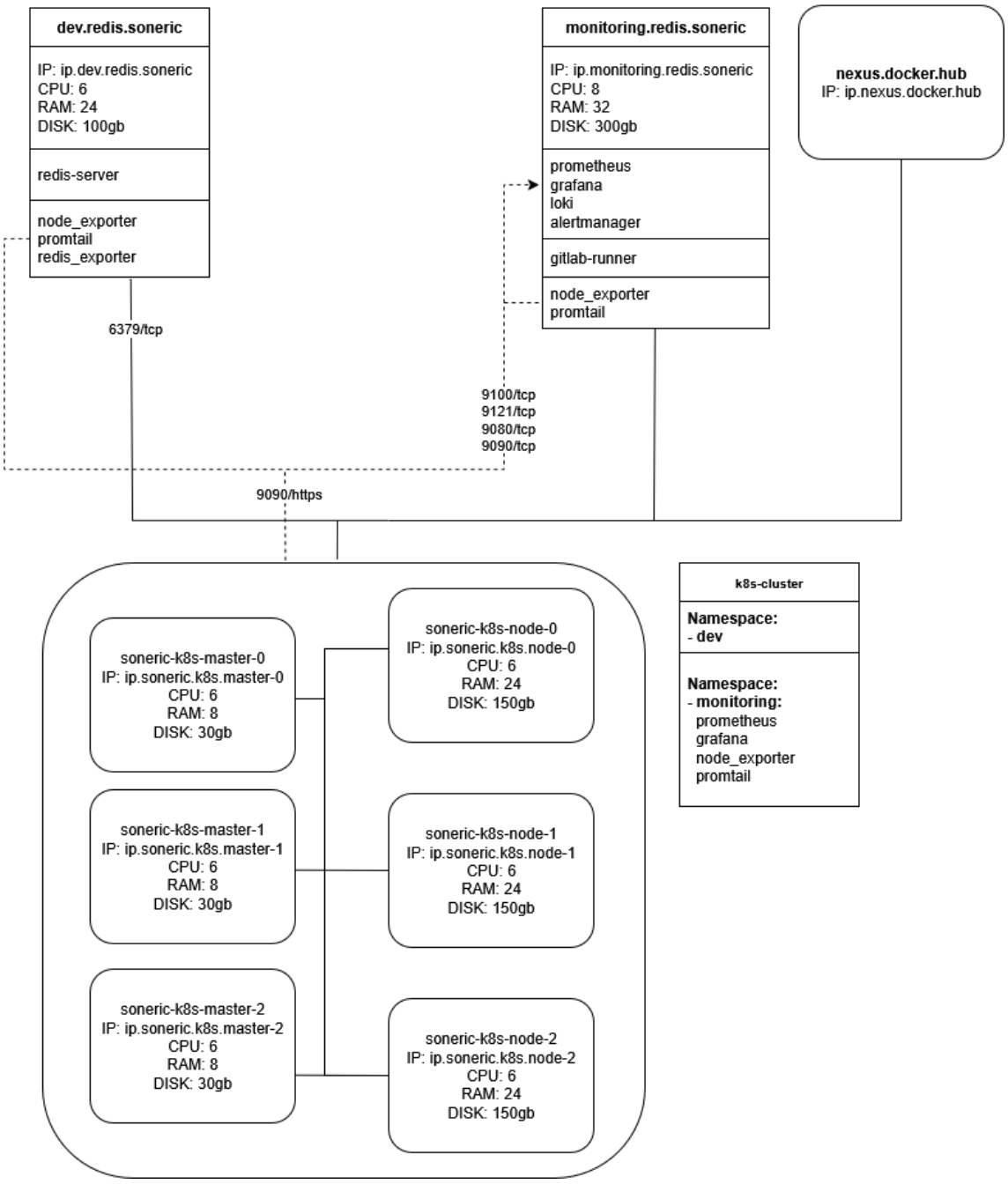

Рисунок 1-1 Диаграмма развертывания UDR

1.1 Аппаратные требования к хостам

Данный документ предполагает, что перед началом установки программного комплекса UDR, произведен монтаж и установка аппаратной части хостов.

Требования к производительности хостов приведены в Таблице 1-1.

Таблица 1-1 Требования к хостам

Тип хоста	CPU	RAM	Disk
k8s-node-master	6	8 Gb	30 Gb
k8s-node-worker	6	24 Gb	150 Gb
redis	6	24 Gb	100 Gb
monitoring	8	32 Gb	300 Gb
nexus	4	8 Gb	60 Gb

1.2 Подготовка хостов к установке

Необходимо привести все хосты в соответствие с требованиями. Хосты должны быть доступны друг для друга по сети.

1.3 Подготовка операционной системы RedOS хостов

В состав ПК UDR входит операционная система RedOS. RedOS добавлена в Единый реестр российских программ для электронных вычислительных машин и баз данных под номером *3751*.

На хостах должна быть установлена операционная система RedOS 8. Необходимо подготовить окружение операционной системы, для этого выполняются следующие шаги.

На всех хостах отключается SELinux и переводится в режим permissiv.

```
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing/SELINUX=permissive/' /etc/selinux/config
```

На всех хостах отключается firewalld.

```
sudo systemctl stop firewalld
sudo systemctl disable firewalld
```

На хостах добавляются DNS и соответствующие имена в файле hosts.

Важно! IP адреса приведены как пример, необходимо заменить IP адреса на знаяения из сетевого плана проекта.

```
10.82.0.13 soneric-k8s-master.novalocal master
10.82.0.98 soneric-k8s-node-0.novalocal node-0
10.82.0.115 soneric-k8s-node-1.novalocal node-1
10.82.0.44 soneric-k8s-node-2.novalocal node-2
```

Загружаются все последние обновления операционной системы

```
sudo dnf update -y
```

Для дальнейшей настройки необходимо перейти в сеанс пользователя root.

```
sudo su
swapoff -a
```

Чтобы swap не появился после перезагрузки сервера, выполняется команда.

```
swapoff -a && sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Создается файл для автозагрузки модулей ядра, необходимых для работы сервиса crio.

```
vi /etc/modules-load.d/crio.conf
sctp
overlay
br_netfilter
```

В ядро загружаются следующие модули.

```
modprobe sctp
modprobe overlay
modprobe br_netfilter
```

Проверяем, что загруженные модули корректно работают.

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

Создаем конфигурационный файл для работы сети внутри Kubernetes.

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

Применяем параметры командой.

```
sysctl --system
```

Перезагружаем операционную систему.

```
reboot
```

На этом подготовка операционный системы хостов закончена.

1.4 Подготовка сервера Nexus

Для того, чтобы продолжить установку ПК UDR, необходимо в первую очредь установить сервер Nexus, который является репозиторий сервером.

Для установки сервера Nexus необходимо подготовленный архив "nexus.tgz" перенести на целевую ВМ и разархивировать командой.

```
tar -xzvf nexus.tgz
```

Перейти в сеанс работы root user и выключить репозитории "enabled=0".

```
sudo su
sed -i 's/led=1/led=0/' /etc/yum.repos.d/RedOS-*.repo
```

Убедиться в текущем значении enabled=0.

```
grep "enabled" /etc/yum.repos.d/RedOS-*.repo
```

Производится локальная установка docker, docker-compose.

```
cd nexus
dnf install docker_install_rpm/*.rpm

systemctl enable docker
systemctl restart docker
systemctl status docker
```

Добавляем образ Nexus в локальное хранилище docker.

```
docker load -i nexus3.tar
chown -R 200:200 nexus-data
```

Запускаем Nexus и проверяем факт успешного запуска.

```
docker compose up -d
docker ps
docker logs <ID>
```

Для того, чтобы удостовериться, что Nexus запущен успешно, можно также проверить фразу "Started Sonatype Nexus COMMUNITY 3.78.2-04".

```
docker_ID=$(docker ps |grep -i 'sonat' | awk -F "sonat" '{print $1}')
docker logs $docker_ID |grep -C2 "Started Sonatype Nexus COMMUNITY 3.78.2-04"
```

Добавляем Nexus в /etc/hosts (например IP nexus = IP.Nexus)

```
echo "IP.Nexus nexus nexus" >> /etc/hosts
```

На этом установка сервера Nexus завершена. После успешного запуска сервера Nexus мы получили репозиторий со всеми необходимыми пакетами для дальнейшей установки инфраструктуры в изолированной среде.

2 Развертывание кластера K8S

ПК UDR реализован на микросервисной архитектуре и развернут в Kubernetes (K8S) кластере. Kubernetes (K8S) кластер включает в себя три “master” хоста и три “worker” хоста, как показано на Рисунке 1-1. Наличие трех хостов каждого функционала обеспечивает необходимое резервирование. Перед началом развертывания Kubernetes (K8S) кластера необходимо установить операционную систему в соответствии с инструкцией “Подготовка операционной системы RedOS хостов”. Развертывание K8S кластера включает в себя общую подготовительную часть как для “master”, так и для “worker” хостов. После выполнения общей подготовительной части на всех хостах, выполняется установка “master” и “worker” хостов.

2.1 Общая подготовительная часть для Master и Worker хостов

На всех Виртуальных Машинах K8S добавляем репозиторий Nexus для дальнейшей установки соответствующих пакетов.

```
#Базовая настройка всех K8s VM
root user
#####
sudo su
sed -i 's/led=1/led=0/' /etc/yum.repos.d/RedOS-*.repo
grep "enabled" /etc/yum.repos.d/RedOS-*.repo
####
Добавляем nexus в /etc/hosts (например IP nexus 10.200.127.60)
####
echo "10.200.127.60 nexus nexus" >> /etc/hosts
####
Репозиторий для установки пакетов
####
cat > /etc/yum.repos.d/nexus.repo <<EOF
[nexus-rpm]
name=Nexus RPM
baseurl=http://nexus:8081/repository/rpm-soneric/rpms/
enabled=1
gpgcheck=0
EOF
####
dnf makecache
####
#ipset и libseccomp уже установлены по умолчанию
####
dnf install --repo nexus-rpm -y ipvsadm  sysstat conntrack
dnf install --repo nexus-rpm -y wget net-tools telnet traceroute tcpdump
```

Отключаем поддержку протокола IPv6 на интерфейсе ens32 “master” хостов.

```
#Disable IPv6 on ens32 interface
####
nmcli connection modify ens32 ipv6.method ignore
nmcli connection up ens32
```

Отключаем поддержку протокола IPv6 на интерфейсах ens32, ens33, ens35 “worker” хостов.

```
nmcli connection modify ens32 ipv6.method ignore
nmcli connection modify ens33 ipv6.method ignore
nmcli connection modify ens35 ipv6.method ignore
nmcli connection up ens32
nmcli connection up ens33
nmcli connection up ens35
```

Пример целевой конфигурации интерфейсов ens32, ens33, ens35 на “worker” хосте показан на Рисунке 2-1.

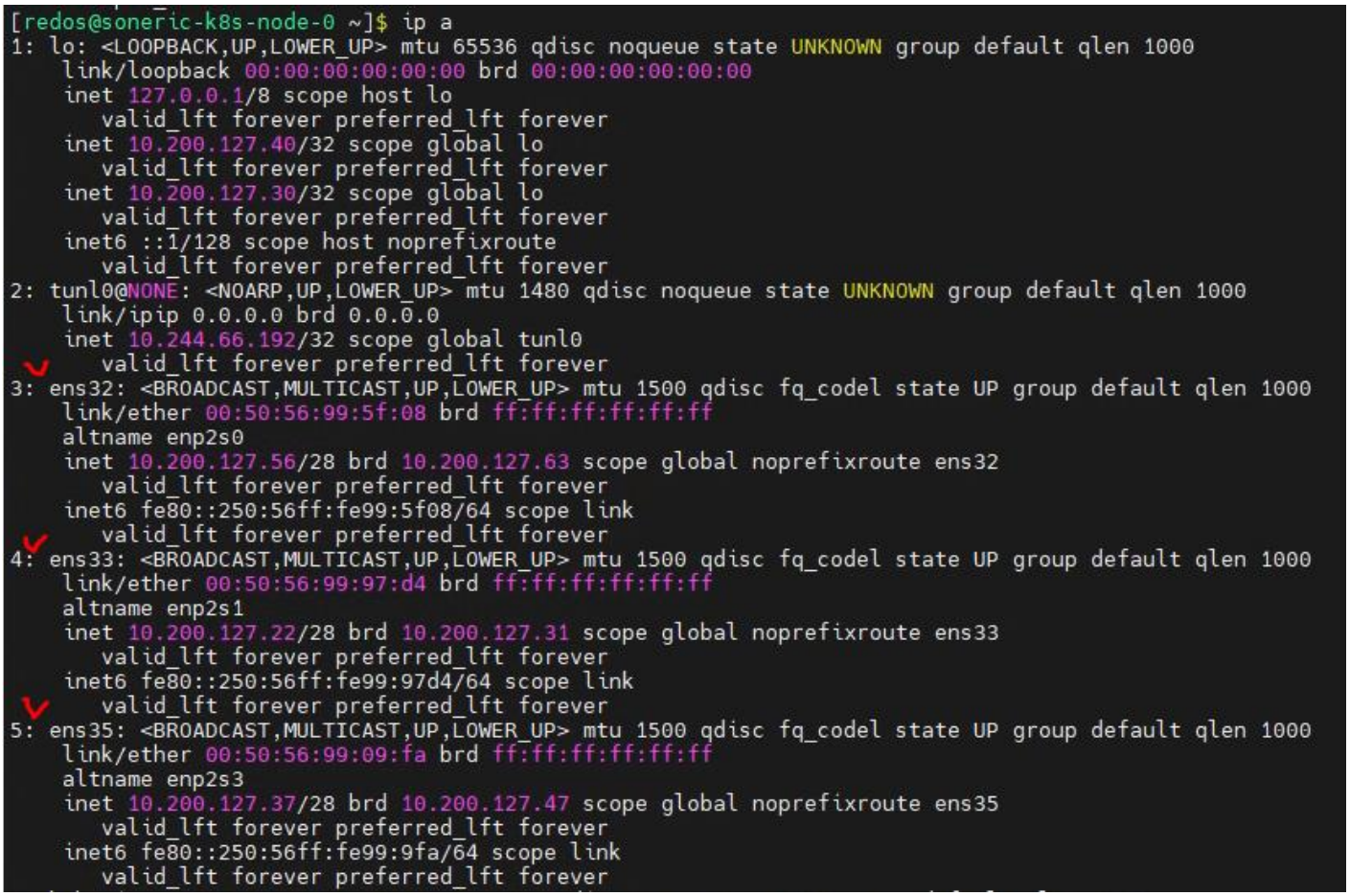

Рисунок 2-1 Конфигурация интерфейсов ens32, ens33, ens35 на “worker” хосте

Отключаем управление интерфейсами K8S от NetworkManager.

```
# Create the following configuration file to prevent NetworkManager from interfering
with the interfaces
####
cat <<'EOF' | sudo tee /etc/NetworkManager/conf.d/calico.conf > /dev/null
[keyfile]
```

```
unmanaged-devices=interface-name:cali*;interface-name:tunl*
EOF

systemctl restart NetworkManager
####
#Проверить
###
nmcli device status
###
```

Прописываем в конфигурации все хосты, которые будут входить в состав K8S кластера.

Важно! IP адреса хостов показаны для примера, необходимо указать выделенные адреса хостов из сетевого плана.

```
####
cat <<EOF >> /etc/hosts
10.200.127.53 soneric-k8s-master-0 master-0
10.200.127.54 soneric-k8s-master-1 master-1
10.200.127.55 soneric-k8s-master-2 master-2
10.200.127.56 soneric-k8s-node-0 node-0
10.200.127.57 soneric-k8s-node-1 node-1
10.200.127.58 soneric-k8s-node-2 node-2
10.200.127.62 soneric-k8s-master-lb master-lb
EOF
####
```

На всех “worker” хостах отключаем функции FIREWALL, SELINUX, DNSMASQ, SWAP.

```
# All nodes turn off firewall, SELINUX, DNSMASQ, SWAP. The server is configured as
follows:
systemctl stop firewalld
systemctl disable --now firewalld
systemctl disable --now dnsmasq

setenforce 0
sed -i 's#SELINUX=enforcing#SELINUX=disabled#g' /etc/sysconfig/selinux
sed -i 's#SELINUX=enforcing#SELINUX=disabled#g' /etc/selinux/config
```

Отключаем SWAP.

```
#Close SWAP partition
####
swapoff -a && sysctl -w vm.swappiness=0
sed -ri '/^[^#]*swap/s@^@#@' /etc/fstab
####
```

Выполняем тонкую настройку параметров на всех “worker” хостах.

```
# All node configurationlimit :
####
```

```
ulimit -SHn 65535
####
# Add the following
####
cat /etc/security/limits.conf
cat <<EOF >> /etc/security/limits.conf
* soft nofile 655360
* hard nofile 131072
* soft nproc 655350
* hard nproc 655350
* soft  memlock unlimited
* hard memlock unlimited
EOF
####
# проверка
####
cat /etc/security/limits.conf
####
```

Проверяем версию kernel, конфигурируем ip_vs параметры.

```
#kernel version check
####
uname -r
####
# All nodes configure IPvs modules
####
modprobe -- ip_vs
modprobe -- ip_vs_rr
modprobe -- ip_vs_wrr
modprobe -- ip_vs_sh
modprobe -- nf_conntrack
####
# Add the following
####
cat <<EOF > /etc/modules-load.d/ipvs.conf
ip_vs
ip_vs_lc
ip_vs_wlc
ip_vs_rr
ip_vs_wrr
ip_vs_lblc
ip_vs_lblcr
ip_vs_dh
ip_vs_sh
ip_vs_fo
ip_vs_nq
ip_vs_sed
ip_vs_ftp
ip_vs_sh
nf_conntrack
ip_tables
ip_set
xt_set
ipt_set
ipt_rpfilter
```

```
ipt_REJECT
ipip
EOF
####
# Проверить
####
cat /etc/modules-load.d/ipvs.conf
####
```

Выполняем тонкую настройку параметров K8S на всех хостах.

```
#Turn on the kernel parameters necessary in the K8S cluster, all nodes Configure the
K8S kernel:
####
cat <<EOF > /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
fs.may_detach_mounts = 1
vm.overcommit_memory=1
vm.panic_on_oom=0
fs.inotify.max_user_watches=89100
fs.file-max=52706963
fs.nr_open=52706963
net.netfilter.nf_conntrack_max=2310720
net.ipv4.tcp_keepalive_time = 600
net.ipv4.tcp_keepalive_probes = 3
net.ipv4.tcp_keepalive_intvl =15
net.ipv4.tcp_max_tw_buckets = 36000
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_max_orphans = 327680
net.ipv4.tcp_orphan_retries = 3
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.ip_conntrack_max = 65536
net.ipv4.tcp_timestamps = 0
net.core.somaxconn = 16384
EOF
####
sysctl --system
####
cat /etc/sysctl.d/k8s.conf
####
cat <<EOF >> /etc/modules-load.d/crio.conf
sctp
xt_sctp
overlay
br_netfilter
EOF
####
modprobe sctp
modprobe xt_sctp
modprobe overlay
modprobe br_netfilter
```

Конфигурируем NTP синхронизацию всех хостах в составе K8S кластера от Nexus. Необходимо, чтобы все хосты синхронизировались от Nexus.

```
-------------------ntp client на всех VM
####
timedatectl status
####
Добавить ntp сервер
####
sudo vi /etc/chrony.conf
#server ntp2.vniiftri.ru iburst
#server ntp3.vniiftri.ru iburst
#server ntp4.vniiftri.ru iburst
server nexus iburst

Перезапустить сервис
###
sudo systemctl restart chronyd
sudo systemctl status chronyd
####
Прверить статус ntp (ждать 2-3 минуты)
####
chronyc sources
chronyc tracking
chronyc activity
chronyc sourcestats
####
```

После того, как на всех хостах выполнены вышеуказанные процедуры, производится их перезагрузка и проводится проверка примененных параметров.

```
####
# After all nodes are configured, restart the server and ensure that the kernel is
still loaded after restart.
####
reboot
lsmod | grep --color=auto -e ip_vs -e nf_conntrack
sudo sysctl --system
####
```

Результат проверки ip_vs параметров показан на Рисунке 2-2.

```
[redos@soneric-k8s-node-0 ~]$ lsmod | grep --color=auto -e ip_vs -e nf_conntrack
nf_conntrack_netlink    61440  0
nfnetlink               20480  4 nft_compat,nf_conntrack_netlink,nf_tables,ip_set
ip_vs_ftp               12288  0
nf_nat                  57344  3 nft_chain_nat,xt_MASQUERADE,ip_vs_ftp
ip_vs_sed               12288  0
ip_vs_nq                12288  0
ip_vs_fo                12288  0
ip_vs_sh                12288  0
ip_vs_dh                12288  0
ip_vs_lblcr             12288  0
ip_vs_lblc              12288  0
ip_vs_wrr               12288  0
ip_vs_rr                12288  57
ip_vs_wlc               12288  0
ip_vs_lc                12288  0
ip_vs                  221184  90 ip_vs_wlc,ip_vs_rr,ip_vs_dh,ip_vs_lblcr,ip_vs_sh,ip_vs_fo,ip_vs_nq,ip_vs_lblc,ip_vs_wrr,i
p_vs_lc,ip_vs_sed,ip_vs_ftp
nf_conntrack           188416  5 xt_conntrack,nf_nat,nf_conntrack_netlink,xt_MASQUERADE,ip_vs
nf_defrag_ipv6          24576  2 nf_conntrack,ip_vs
nf_defrag_ipv4          12288  1 nf_conntrack
[redos@soneric-k8s-node-0 ~]$
```

Рисунок 2-2 Проверка ip_vs параметров нв хосте

На этом бызовые настройки, применимые ко всех K8S хостам, завершены. Далее приступаем к установке “master” и “worker” хостов.

2.2 Развертывание Master хостов

На всех “master” хостах устанавливаем пакеты “haproxy” и “keepalived”.

```
####
dnf install --repo nexus-rpm -y haproxy keepalived
dnf install --repo nexus-rpm -y kubernetes1.30-client.x86_64 kubernetes1.30-
kubeadm.x86_64 kubernetes1.30-master.x86_64 cri-o1.30.x86_64 cri-tools1.30.x86_64
runc
####
```

Для установления сессий между хостами K8S кластера по сохраненным ключам, без указания имени пользователя и пароля, генерируем ключ и копируем его на все хосты K8S кластера.

```
####
ssh-keygen -t rsa -b 4096
####
####
for i in soneric-k8s-master-0 soneric-k8s-master-1 soneric-k8s-master-2 soneric-k8s-
node-0 soneric-k8s-node-1 soneric-k8s-node-2;do ssh-copy-id -i .ssh/id_rsa.pub
$i;done
```

Меняем свойства iptables и конфигурационный файл crio.conf.

```
####
iptables -P FORWARD ACCEPT
sed -i '/^\[crio\.runtime\]/a seccomp_profile = "\/etc\/containers\/policy.json"'
/etc/crio/crio.conf
systemctl enable --now crio
####
```

Выполняем изменения в конфигурационных файлах.

```
# Подключение репозитория docker-soneric найти #pause_image и добавить строку
####
vi /etc/crio/crio.conf
pause_image = "nexus:5000/repository/docker-soneric/pause:3.9"
####
```

Конфигурируем работу протокола HTTP, для этого добавляем параметры в файл registries.conf.

```
#Разрешить http. Добавить в конце файла /etc/containers/registries.conf
####
vi /etc/containers/registries.conf
#Добавить в конце
[[registry]]
location = "nexus:5000"
insecure = true
####
```

Запускаем сервисы.

```
#Запусть сервисы
####
systemctl daemon-reload
systemctl restart crio
systemctl enable kubelet.service
ping nexus
```

Проверяем дуступ к Nexus.

```
####
kubeadm config images list --image-repository nexus:5000/repository/docker-soneric
kubeadm config images pull --image-repository nexus:5000/repository/docker-soneric --
cri-socket unix:///var/run/crio/crio.sock
####
```

Пример успешной проверки доступа к Nexus показан на Рисунке 2-3.

```
[redos@soneric-k8s-master-0 ~]$ kubeadm config images list --image-repository nexus:5000/repository/docker-soneric
W0806 12:33:35.980558 1511087 version.go:104] could not fetch a Kubernetes version from the internet: unable to get
txt": dial tcp: lookup dl.k8s.io on 127.0.0.53:53: server misbehaving
W0806 12:33:35.980686 1511087 version.go:105] falling back to the local client version: v1.30.10
nexus:5000/repository/docker-soneric/kube-apiserver:v1.30.10
nexus:5000/repository/docker-soneric/kube-controller-manager:v1.30.10
nexus:5000/repository/docker-soneric/kube-scheduler:v1.30.10
nexus:5000/repository/docker-soneric/kube-proxy:v1.30.10
nexus:5000/repository/docker-soneric/coredns:v1.11.3
nexus:5000/repository/docker-soneric/pause:3.9
nexus:5000/repository/docker-soneric/etcd:3.5.15-0
[redos@soneric-k8s-master-0 ~]$
```

Рисунок 2-3 Проверка доступа к Nexus

Перед установкой haproxy сохраняем старую конфигурацию для резерва.

```
mv /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy_default.cfg
```

Конфигурируем haproxy, для этой цели создаем новый файс с указанием в нем актуальных IP адресов “master” хостов.

```
cat <<'EOF' | sudo tee /etc/haproxy/haproxy.cfg > /dev/null
global
  maxconn  2000
  ulimit-n  16384
  log  127.0.0.1 local0 err
  stats timeout 30s

defaults
  log global
  mode  http
  option  httplog
  timeout connect 5000
  timeout client  50000
  timeout server  50000
  timeout http-request 15s
  timeout http-keep-alive 15s

frontend monitor-in
  bind *:33305
  mode http
  option httplog
  monitor-uri /monitor

frontend soneric-k8s-master
  bind 0.0.0.0:16443
  bind 127.0.0.1:16443
  mode tcp
  option tcplog
  tcp-request inspect-delay 5s
  default_backend soneric-k8s-master

backend soneric-k8s-master
  mode tcp
  option tcplog
  option tcp-check
  balance roundrobin
  default-server inter 10s downinter 5s rise 2 fall 2 slowstart 60s maxconn 250
maxqueue 256 weight 100
  server soneric-k8s-master-0 10.200.127.53:6443 check
  server soneric-k8s-master-1 10.200.127.54:6443 check
  server soneric-k8s-master-2 10.200.127.55:6443 check
EOF
```

Прописываем конфигурацию KeePalived для каждого “master” хоста. Сохраняем существующую конфигурацию KeePalived для резерва.

```
####
mv /etc/keepalived/keepalived.conf /etc/keepalived/keepalived_default.conf
```

```
####
```

Создаем новый файл конфигурации KeePalived. В данном примере в качестве VIP адреса сконфигурирован 10.200.127.62.

Выполняем эту конфигурацию на всех “master” хостах, изменив в параметр mcast_src_ip актуальный IP адрес целевого “master” хоста.

```
cat <<'EOF' | sudo tee /etc/keepalived/keepalived.conf > /dev/null
! Configuration File for keepalived
global_defs {
    router_id LVS_DEVEL
script_user root
    enable_script_security
}
vrrp_script chk_apiserver {
    script "/etc/keepalived/check_apiserver.sh"
    interval 5
    weight -5
    fall 2
rise 1
}
vrrp_instance VI_1 {
    state MASTER
    interface ens32
    mcast_src_ip 10.200.127.53
    virtual_router_id 51
    priority 101
    advert_int 2
    authentication {
        auth_type PASS
        auth_pass Soneric1
    }
    virtual_ipaddress {
        10.200.127.62
    }
    track_script {
       chk_apiserver
    }
}
EOF
```

На всех “master” хостах выполнить конфигурацию Keepalived.

```
#All Master Nodes Configure KeePaliveD Health Checkup:
####

cat <<'EOF' | sudo tee /etc/keepalived/check_apiserver.sh > /dev/null
#!/bin/bash

err=0
for k in $(seq 1 3)
do
```

```
    check_code=$(pgrep haproxy)
    if [[ $check_code == "" ]]; then
        err=$(expr $err + 1)
        sleep 1
        continue
    else
        err=0
        break
    fi
done

if [[ $err != "0" ]]; then
    echo "systemctl stop keepalived"
    /usr/bin/systemctl stop keepalived
    exit 1
else
    exit 0
fi
EOF

or via vi
####
vi /etc/keepalived/check_apiserver.sh
####
```

Сконфигурировать права для Keepalived.

```
#Add X permissions
####
cat /etc/keepalived/check_apiserver.sh
chmod +x /etc/keepalived/check_apiserver.sh
```

Пример проверки прав доступа для check_apiserver.sh показан на Рисунке 2-4.

```
[redos@soneric-k8s-master-0 ~]$ ls -l /etc/keepalived/
total 12
-rwxr-xr-x 1 root root  356 Aug  4 11:37 check_apiserver.sh
-rw-r--r-- 1 root root  572 Aug  4 11:26 keepalived.conf
-rw-r--r-- 1 root root 3625 Nov 25  2024 keepalived_default.conf
[redos@soneric-k8s-master-0 ~]$
```

Рисунок 2-4 Проверка прав доступа для файла check_apiserver.sh

Стартовать сервисы Haproxy и KeePalived.

```
####
#Start Haproxy and KeePalived
####
systemctl daemon-reload
systemctl enable --now haproxy
systemctl enable --now keepalived
####
```

После завершения конфигурирования Keepalived на всех “master” хостах проверить доступность VIP адреса со всех “master” хостов.

```
#Important: If Keepalived and Haproxy are installed, you need to test whether
Keepalived is normal.
#Test VIP
####
ping 10.200.127.62 -c 4
telnet 10.200.127.62 16443
####
```

На этом базовые настройки для “master” хостов завершены и следующим этапом является установка K8S. Локальным репозиторием в ПК UDR является Nexus, пример графического интерфейса Nexus показан на Рисунке 2-5.

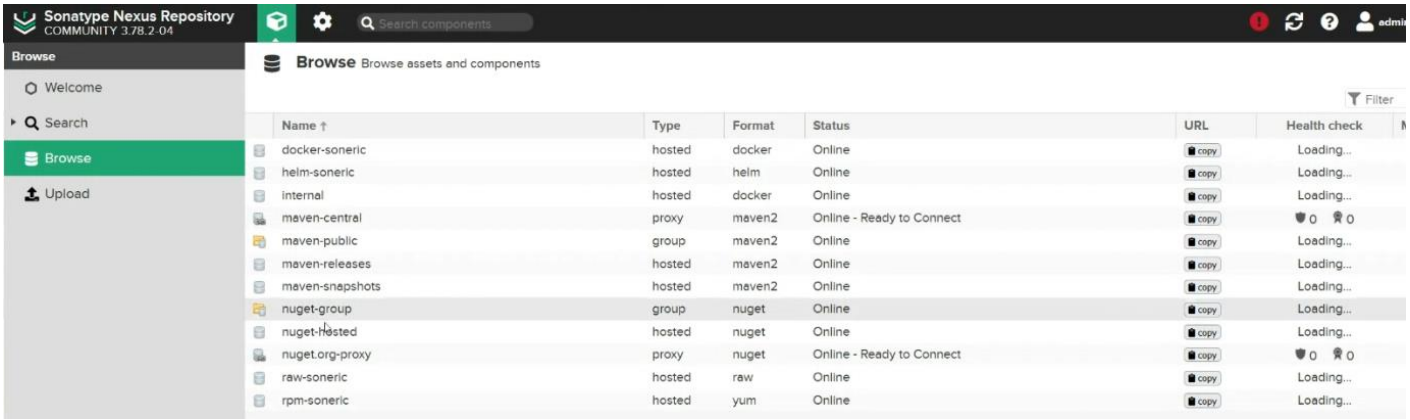

Рисунок 2-5 Графический интерфейс Nexus

В Nexus находятся все необходимые для установки модули и конфигурационные файлы.

Скачиваем из Nexus архив с файлами конфигурации HA_config.tgz на “master-0” хост и разархивируем его.

```
####
curl http://nexus:8081/repository/raw-soneric/k8s_install/HA_config.tgz -o
/home/redos/HA_config.tgz.tgz
tar -xvzf HA_config.tgz
####
```

Инициируем инсталляцию K8S на “master-0” хосте.

```
####
sudo kubeadm init -v7 --ignore-preflight-errors Swap --config
k8sinstall/1_ha_kubeadm-config_v1_laba.yaml --upload-certs
####
```

Пример процесса инсталляции на “master-0” показан на Рисунке 2-6.

```
[redos@soneric-k8s-master-0 ~]$ sudo kubeadm init -v7 --ignore-preflight-errors Swap --config 1_ha_kubeadm-config_v1_laba.yaml --upload-certs
[sudo] password for redos:
I0804 12:46:54.806668   11945 initconfiguration.go:260] loading configuration from "1_ha_kubeadm-config_v1_laba.yaml"
[init] Using Kubernetes version: v1.30.10
[preflight] Running pre-flight checks
I0804 12:46:54.822655   11945 checks.go:561] validating Kubernetes and kubeadm version
I0804 12:46:54.822672   11945 checks.go:166] validating if the firewall is enabled and active
I0804 12:46:56.185552   11945 checks.go:201] validating availability of port 6443
I0804 12:46:56.186444   11945 checks.go:201] validating availability of port 10259
I0804 12:46:56.186560   11945 checks.go:201] validating availability of port 10257
I0804 12:46:56.186635   11945 checks.go:278] validating the existence of file /etc/kubernetes/manifests/kube-apiserver.yaml
I0804 12:46:56.186695   11945 checks.go:278] validating the existence of file /etc/kubernetes/manifests/kube-controller-manager.yaml
I0804 12:46:56.186718   11945 checks.go:278] validating the existence of file /etc/kubernetes/manifests/kube-scheduler.yaml
I0804 12:46:56.186737   11945 checks.go:278] validating the existence of file /etc/kubernetes/manifests/etcd.yaml
I0804 12:46:56.186760   11945 checks.go:428] validating if the connectivity type is via proxy or direct
I0804 12:46:56.186813   11945 checks.go:467] validating http connectivity to first IP address in the CIDR
I0804 12:46:56.186854   11945 checks.go:467] validating http connectivity to first IP address in the CIDR
I0804 12:46:56.186882   11945 checks.go:102] validating the container runtime
I0804 12:46:56.246025   11945 checks.go:637] validating whether swap is enabled or not
        [WARNING Swap]: swap is supported for cgroup v2 only; the NodeSwap feature gate of the kubelet is beta but disabled by default
I0804 12:46:56.246235   11945 checks.go:368] validating the presence of executable crictl
I0804 12:46:56.246293   11945 checks.go:368] validating the presence of executable conntrack
I0804 12:46:56.246368   11945 checks.go:368] validating the presence of executable ip
I0804 12:46:56.246402   11945 checks.go:368] validating the presence of executable iptables
I0804 12:46:56.246501   11945 checks.go:368] validating the presence of executable mount
I0804 12:46:56.246559   11945 checks.go:368] validating the presence of executable nsenter
I0804 12:46:56.246598   11945 checks.go:368] validating the presence of executable ethtool
        [WARNING FileExisting-ethtool]: ethtool not found in system path
I0804 12:46:56.246695   11945 checks.go:368] validating the presence of executable tc
        [WARNING FileExisting-tc]: tc not found in system path
I0804 12:46:56.246786   11945 checks.go:368] validating the presence of executable touch
I0804 12:46:56.246863   11945 checks.go:514] running all checks
I0804 12:46:56.288197   11945 checks.go:399] checking whether the given node name is valid and reachable using net.LookupHost
```

Рисунок 2-6 Инсталляция K8S на “master-0” хосте

Необходимо дождаться завершения инсталляции. Перед использованием K8S кластера необходимо выполнить следующие команды под регулярным пользователем (**НЕ** root). Данные команды выполняются на всех трех хостах “master-0”, “master-1”, “master-2”.

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

Важно! После завершения инсталляции K8S на “master” хосте, на терминале будут выведены команды с секретными ключами, они используются для подключения к K8S кластеру домолнительных “master” и “worker” хостов. Необходимо сохранить эти команды для дальнейшего использования.

Пример распечатки выполнения команды kubeadm init с ключами для подключения к K8S кластеру новых хостов показан на Рисунке 2-7.

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

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

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

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

  export KUBECONFIG=/etc/kubernetes/admin.conf

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

You can now join any number of the control-plane node running the following command on each as root:

  kubeadm join 10.200.127.62:6443 --token dme4g1.u5mq8i6qut52aw85 \
        --discovery-token-ca-cert-hash sha256:8f4477240ea6956d088b295d6677f599a1b8db10bd2a2abeec1b6d7273391f80 \
        --control-plane --certificate-key 5162535a9dd49c600506940dec1a56947a6871a700b421e17b8eb2be3b02c0e7

Please note that the certificate-key gives access to cluster sensitive data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

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

Рисунок 2-7 Процедура подключения к K8S кластеру управляющих “master” хостов

Для включения в K8S кластер дополнительных управляющих хостов “master-1” и “master-2” необходимо на каждом из них выполнить команду от пользователя root с секретным ключом. В инструкции значение секретного ключа указано как пример. Реальное значение следует взять из распечатки результата выполнения команды kubeadm init.

```
#You can now join any number of the control-plane node running the following command
on each as root:
#(взять свою распечатку из kubeadm init)
####
sudo kubeadm join 10.200.127.62:6443 --token dme4g1.u5mq8i6qut52aw85 --discovery-
token-ca-cert-hash
sha256:8f4477240ea6956d088b295d6677f599a1b8db10bd2a2abeec1b6d7273391f80 --control-
plane --certificate-key
5162535a9dd49c600506940dec1a56947a6871a700b421e17b8eb2be3b02c0e7
```

Применить имеющийся конфигурационный файл для Calico. Команда дается на “master-0” хосте. В результате выполнения Calico запускается на всем K8S кластере.

```
#Применить
kubectl apply -f k8sinstall/2__ha_calico3_29_4_ipforw_laba_v1.yaml
####
```

Пример успешного запуска сервиса Calico показан на Рисунке 2-8.

```
[redos@soneric-k8s-master-0 ~]$ kubectl get all -A -o wide
NAMESPACE     NAME                                                READY   STATUS
GATES
kube-system   pod/calico-kube-controllers-795ff58f54-xm8v5        1/1     Running
kube-system   pod/calico-node-4gpkb                               1/1     Running
kube-system   pod/calico-node-56zt5                               0/1     Running
kube-system   pod/calico-node-697cf                               1/1     Running
kube-system   pod/calico-node-fcrzz                               0/1     Running
kube-system   pod/calico-node-j772m                               0/1     Running
kube-system   pod/calico-node-kbgph                               0/1     Running
kube-system   pod/coredns-7b49bdddfb-9dzs6                        0/1     CrashLoopBackOff
kube-system   pod/coredns-7b49bdddfb-kjlt6                        0/1     CrashLoopBackOff
kube-system   pod/etcd-master                                     1/1     Running
kube-system   pod/etcd-soneric-k8s-master-1                       1/1     Running
kube-system   pod/etcd-soneric-k8s-master-2                       1/1     Running
kube-system   pod/kube-apiserver-master                           1/1     Running
kube-system   pod/kube-apiserver-soneric-k8s-master-1             1/1     Running
kube-system   pod/kube-apiserver-soneric-k8s-master-2             1/1     Running
```

Рисунок 2-8 Успешный запуск сервиса Calico на K8S кластере

Применяем конфигурацию для core-dns с помощью имеющегося конфигурационного файла и рестартуем сервис.

```
#поменять core-dns config
####
kubectl apply -f k8sinstall/3_ha_core-dns.yaml
kubectl -n kube-system rollout restart deployment coredns
####
```

Убедиться, что сервис core-dns находится в состоянии RUNNING, как показано на Рисунке 2-9.

```
[redos@soneric-k8s-master-0 ~]$ kubectl get all -A
NAMESPACE     NAME                                                READY   STATUS    RESTARTS        AGE
kube-system   pod/calico-kube-controllers-795ff58f54-xm8v5        1/1     Running   0               2m59s
kube-system   pod/calico-node-4gpkb                               1/1     Running   0               2m59s
kube-system   pod/calico-node-56zt5                               1/1     Running   0               2m59s
kube-system   pod/calico-node-697cf                               1/1     Running   0               2m59s
kube-system   pod/calico-node-fcrzz                               1/1     Running   0               2m59s
kube-system   pod/calico-node-j772m                               1/1     Running   0               2m59s
kube-system   pod/calico-node-kbgph                               1/1     Running   0               2m59s
kube-system   pod/coredns-7999cc8d76-8s5k7                        1/1     Running   0               4s
kube-system   pod/coredns-7999cc8d76-k6q5z                        1/1     Running   0               4s
kube-system   pod/etcd-master                                     1/1     Running   12              17m
kube-system   pod/etcd-soneric-k8s-master-1                       1/1     Running   0               13m
kube-system   pod/etcd-soneric-k8s-master-2                       1/1     Running   0               10m
kube-system   pod/kube-apiserver-master                           1/1     Running   1               17m
kube-system   pod/kube-apiserver-soneric-k8s-master-1             1/1     Running   0               13m
kube-system   pod/kube-apiserver-soneric-k8s-master-2             1/1     Running   0               10m
kube-system   pod/kube-controller-manager-master                  1/1     Running   2 (118s ago)    16m
kube-system   pod/kube-controller-manager-soneric-k8s-master-1    1/1     Running   0               13m
kube-system   pod/kube-controller-manager-soneric-k8s-master-2    1/1     Running   0               10m
kube-system   pod/kube-proxy-hkcbm                                1/1     Running   0               7m12s
kube-system   pod/kube-proxy-k82rm                                1/1     Running   0               5m49s
kube-system   pod/kube-proxy-ptjzl                                1/1     Running   0               16m
kube-system   pod/kube-proxy-s575q                                1/1     Running   0               13m
kube-system   pod/kube-proxy-w96lh                                1/1     Running   0               10m
kube-system   pod/kube-proxy-xx2rw                                1/1     Running   0               6m26s
kube-system   pod/kube-scheduler-master                           1/1     Running   13 (118s ago)   17m
kube-system   pod/kube-scheduler-soneric-k8s-master-1             1/1     Running   7               13m
kube-system   pod/kube-scheduler-soneric-k8s-master-2             1/1     Running   0               10m
```

Рисунок 2-9 Сервис core-dns находится в состоянии RUNNING

Убедиться, что все хосты, входящие в состав K8S кластера, находятся в состоянии Ready, как показано на Рисунке

```
[redos@soneric-k8s-master-0 ~]$ kubectl get node
NAME                   STATUS   ROLES           AGE     VERSION
master                 Ready    control-plane   17m     v1.30.10
soneric-k8s-master-1   Ready    control-plane   13m     v1.30.10
soneric-k8s-master-2   Ready    control-plane   10m     v1.30.10
soneric-k8s-node-0     Ready    <none>          7m29s   v1.30.10
soneric-k8s-node-1     Ready    <none>          6m43s   v1.30.10
soneric-k8s-node-2     Ready    <none>          6m6s    v1.30.10
[redos@soneric-k8s-master-0 ~]$
```

Рисунок 2-10 Хосты K8S кластера находятся в состоянии Ready

Проверить, на всех kube-proxy pod K8S кластера, что используется ipvs.

```
kubectl logs -n kube-system kube-proxy-hkcbm | grep "Using ipvs Proxier"
kubectl logs -n kube-system kube-proxy-k82rm | grep "Using ipvs Proxier"
kubectl logs -n kube-system kube-proxy-ptjzl | grep "Using ipvs Proxier"
kubectl logs -n kube-system kube-proxy-s575q | grep "Using ipvs Proxier"
kubectl logs -n kube-system kube-proxy-w96lh | grep "Using ipvs Proxier"
kubectl logs -n kube-system kube-proxy-xx2rw | grep "Using ipvs Proxier"
```

Пример выполнения проверки ipvs на всех kube-proxy pod показан на Рисунке 2-11.

```
[redos@soneric-k8s-master-0 ~]$ kubectl logs -n kube-system kube-proxy-hkcbm | grep "Using ipvs Proxier"
I0804 09:57:45.101273       1 server_linux.go:233] "Using ipvs Proxier"
[redos@soneric-k8s-master-0 ~]$ kubectl logs -n kube-system kube-proxy-k82rm | grep "Using ipvs Proxier"
I0804 09:59:08.778325       1 server_linux.go:233] "Using ipvs Proxier"
[redos@soneric-k8s-master-0 ~]$ kubectl logs -n kube-system kube-proxy-ptjzl | grep "Using ipvs Proxier"
I0804 09:47:46.990520       1 server_linux.go:233] "Using ipvs Proxier"
[redos@soneric-k8s-master-0 ~]$ kubectl logs -n kube-system kube-proxy-s575q | grep "Using ipvs Proxier"
I0804 09:51:37.017255       1 server_linux.go:233] "Using ipvs Proxier"
[redos@soneric-k8s-master-0 ~]$ kubectl logs -n kube-system kube-proxy-w96lh | grep "Using ipvs Proxier"
I0804 09:54:24.758594       1 server_linux.go:233] "Using ipvs Proxier"
[redos@soneric-k8s-master-0 ~]$ kubectl logs -n kube-system kube-proxy-xx2rw | grep "Using ipvs Proxier"
I0804 09:58:31.873338       1 server_linux.go:233] "Using ipvs Proxier"
[redos@soneric-k8s-master-0 ~]$
```

Рисунок 2-11 Проверка параметра ipvs на всех kube-proxy pod K8S кластера

На всех “master” хостах сконфигурировать таймера “толерантности”, которые будут определять период времени, по истечении которого изменяется статус POD в K8S кластере в случае аварии. Все POD на K8S кластере впоследствии будут создаваться с указанной в параметрах толерантностью.

Важно! После данных изменений происходит перенастройка K8S кластера с его перезагрузкой. Перезагрузка инициируется автоматически, необходимо дождаться окончания процедуры. Для применения настроек на K8S кластере необходимо подождать 1-2 минуты.

```
#Добавить в /etc/kubernetes/manifests/kube-apiserver.yaml  на всех мастерах
sudo su
vi /etc/kubernetes/manifests/kube-apiserver.yaml
####
    - --default-not-ready-toleration-seconds=20
```

```
    - --default-unreachable-toleration-seconds=20
####
```

Пример сессии при настройке параметров толерантности показан на Рисунке 2-12.

```
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 10.200.127.55:6443
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=10.200.127.55
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --secure-port=6443
    - --service-account-issuer=https://kubernetes.default.svc.cluster.local
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --service-account-signing-key-file=/etc/kubernetes/pki/sa.key
    - --service-cluster-ip-range=10.245.0.0/18
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
    - --default-not-ready-toleration-seconds=20
    - --default-unreachable-toleration-seconds=20
    image: nexus:5000/repository/docker-soneric/kube-apiserver:v1.30.10
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 8
      httpGet:
-- INSERT --
```

Рисунок 2-12 Настройка параметров толерантности

Инициировать установку и конфигурацию сервиса loxilb. Команда выполняется на “master-0” хосте.

```
kubectl apply -f k8sinstall/4_ha_kube_and_loxilb-nobgp_laba_egress.yaml
```

Создаем namespace для 2G и 4G для последующей установки сервисных приложений.

```
kubectl create namespace dev-4g
kubectl create namespace dev-2g
```

2.3 Развертывание Worker хостов

На всех “worker” хостах отключаем “неразборчивый” режим на интерфейсах ens32, ens33, ens35.

```
#Делаем Promiscuous mode — «неразборчивый» режим работы сетевого устройства,
например, сетевого адаптера, который позволяет ему видеть и «слушать» все сетевые
пакеты, проходящие через сеть, даже если они не предназначены для него (можно
попробовать без него)
nmcli connection modify ens32 ethernet.accept-all-mac-addresses no
nmcli connection modify ens33 ethernet.accept-all-mac-addresses no
nmcli connection modify ens35 ethernet.accept-all-mac-addresses no
```

Важно! В случае использования Multus, “неразборчивый” режим нужно будет включить на интерфейсах ens32, ens33, ens35 “worker” хостов.

```
nmcli connection modify ens32 ethernet.accept-all-mac-addresses yes
nmcli connection modify ens33 ethernet.accept-all-mac-addresses yes
nmcli connection modify ens35 ethernet.accept-all-mac-addresses yes
```

На всех “worker-0” “worker-1” “worker-2” хостах для подключения их к K8S кластеру от имени пользователя root необходимо выполнить команду. Указанные в команде значения секретных ключей даны как пример. Реальные значения секретных ключей необходимо взять из распечатки kubeadm init, полученной при инсталляции K8S на “master-0” хосте.

```
#Подключить worker nodes 1 за одним. (взять свою распечатку из kubeadm init)

kubeadm join 10.200.127.62:6443 --token dme4g1.u5mq8i6qut52aw85 --discovery-token-ca-
cert-hash sha256:8f4477240ea6956d088b295d6677f599a1b8db10bd2a2abeec1b6d7273391f80
```

3 Развертывание UDR

Высокоуровневая архитектура ПК UDR показана на Рисунок 3-1.

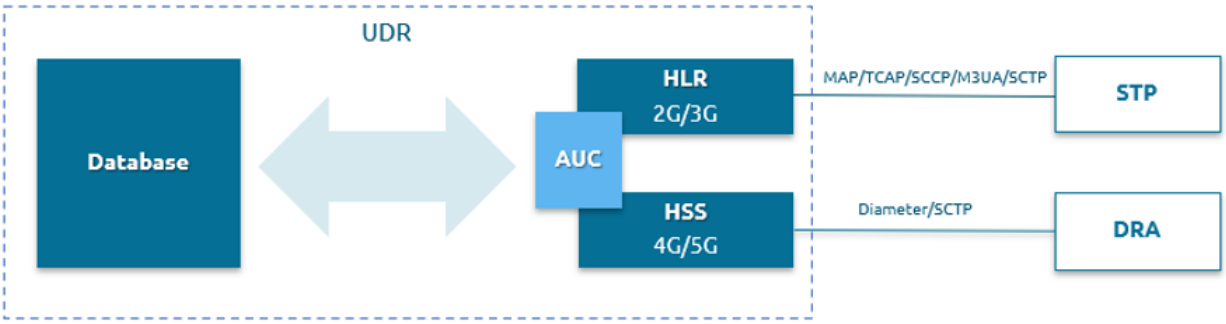

Рисунок 3-1 Высокоуровневая архитектура ПК UDR

3.1 Установка базы данных Redis

В состав ПК UDR входит СУБД Redis с открытой лицензией (open-source edition), являющаяся разрешенным к использованию составным компонентом.

Все команды по установке базы данных Redis выполняются на целевой виртульной машине, на которой разворачивается Redis, если это не оговорено отдельно.

Для установки базы данных Redis необходимо сконфигурировать и подключить репозиторий в ранее установленном сервере Nexus.

Добавляем nexus в /etc/hosts (например IP Nexus = IP.Nexus).

```
sudo su
echo "IP.Nexus nexus nexus" >> /etc/hosts
```

Подключаем репозиторий Nexus.

```
cat > /etc/yum.repos.d/nexus.repo <<EOF
[nexus-rpm]
name=Nexus RPM
baseurl=http://nexus:8081/repository/rpm-soneric/rpms/
enabled=1
gpgcheck=0
EOF
```

Устанавливаем сеанс работы от root user и выключаем репозитории "enabled=0".

```
sudo su
sed -i 's/led=1/led=0/' /etc/yum.repos.d/RedOS-*.repo
```

Проверяем значение enabled=0.

```
grep "enabled" /etc/yum.repos.d/RedOS-*.repo
```

Обновляем кэш.

```
dnf makecache
```

Производим установка сервера Redis.

```
dnf --repo nexus-rpm install -y redis

systemctl enable redis
systemctl start redis
systemctl status redis
```

Создаем директорию для модулей.

```
# Для модулей redis
mkdir -p /opt/redis/modules/
# Для экспортеров
mkdir -p /opt/stat/
```

Скачивваем модули Redis.

```
curl http://nexus:8081/repository/raw-soneric/dirs/librejson.so -o
/opt/redis/modules/librejson.so
curl http://nexus:8081/repository/raw-soneric/dirs/redisearch.so -o
/opt/redis/modules/redisearch.so
```

Сохраняем копию первоначального файла конфигурации redis.conf

```
mv /etc/redis/redis.conf /etc/redis/redis_orig.conf
curl http://nexus:8081/repository/raw-soneric/dirs/redis.conf -o
/etc/redis/redis.conf
```

Выполняем настройки для Redis.

```
setenforce 0
sed -i 's/^SELINUX=enforcing/SELINUX=permissive/' /etc/selinux/config
```

Добавляем параметры в /etc/sysctl.conf.

```
echo "vm.swappiness = 1" >> /etc/sysctl.conf
echo "vm.overcommit_memory = 1" >> /etc/sysctl.conf
sysctl vm.overcommit_memory=1
```

Создаем сервис disable-thp.

```
cat > /etc/systemd/system/disable-thp.service <<EOF
[Unit]
Description=Disable Transparent HugePages (THP)
After=syslog.target network.target

[Service]
Type=simple
ExecStart=/bin/bash -c "echo never | tee /sys/kernel/mm/transparent_hugepage/enabled
> /dev/null"

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

Включаем и запускаем службу disable-thp.

```
sudo systemctl daemon-reload
sudo systemctl enable disable-thp
sudo systemctl start disable-thp
```

Проверяем статус службы disable-thp.

```
sudo systemctl status disable-thp
```

Создаем нужные права и пользователей для Redis.

```
sudo chmod -R 755 /opt/redis/modules/
sudo chown -R redis:redis /opt/redis/modules/
sudo chown -R redis:redis /etc/redis/
```

Изменяем лимиты.

```
vi /etc/systemd/system/redis.service.d/limit.conf

# [Service]
LimitNOFILE=100032
TimeoutStartSec=300s
```

Увеличиваем значение somaxconn для улучшения производительности.

```
echo "net.core.somaxconn = 65535" | sudo tee -a /etc/sysctl.conf

sysctl -p
```

Запускаем Redis.

```
systemctl daemon-reload
systemctl restart redis
```

Выполняем дополнительные настройки.

```
mount -o remount,noatime /

redis-cli module list
redis-cli info
```

На этом установка базы данных Redis завершена.

3.2 Установка компонентов мониторинга (Graphana)

Подсистема мониторинга уснанавливается на отдельном сервере и включает в себя следующие программные компоненты:

1) Prometheus - основной инструмент для сбора метрик. Он собирает данные о состоянии сервисов и оборудования с помощью экспортёров (например, Node Exporter);
2) Alertmanager - используется для обработки и отправки уведомлений, когда условия тревоги активируются (например, превышен порог нагрузки на CPU);
3) Node Exporter - экспортер метрик, который позволяет собирать данные о нагрузке на сервере (например, нагрузка CPU, использование памяти);
4) Grafana - графический интерфейс для отображения собранных метрик. Позволяет строить дашборды и настраивать визуализации;
5) Loki - система для централизованного хранения и анализа логов. Данная функция обрабатывает лог-файлы и используется для последующей диагностики проблем в приложения;
6) Promtail - агент, собирающий лог-файлы и отвечающий за дальнейшую отправку их в систему централизованного хранения Loki.

Для установки компонентов мониторига необходимо:

```
####
sudo su
sed -i 's/led=1/led=0/' /etc/yum.repos.d/RedOS-*.repo
grep "enabled" /etc/yum.repos.d/RedOS-*.repo
####
```

Добавляем nexus в /etc/hosts (например IP nexus 10.200.127.14)

```
####
echo "10.200.127.14 nexus nexus" >> /etc/hosts
####
```

Репозиторий для установки пакетов

```
####
cat > /etc/yum.repos.d/nexus.repo <<EOF
[nexus-rpm]
name=Nexus RPM
baseurl=http://nexus:8081/repository/rpm-soneric/rpms/
enabled=1
gpgcheck=0
EOF
####
```

```
####
dnf makecache
####
```

Устанавливаем Docker из репозитория nexus-rpm.

```
Установка docker, docker-compose:
####bash
sudo dnf --repo nexus-rpm install -y docker-ce docker-ce-cli docker-compose
####
```

Запускаем приложение Docker.

```
####bash
sudo systemctl start docker
sudo systemctl enable --now docker
####
```

Добавляем пользователя в группу Docker.

```
####bash
sudo usermod -aG docker ${USER}
####
```

Для подключения Docker реестра Nexus необходимо добавить в файл daemon.json строку `"insecure-registries": ["nexus:5000"].

```
не забыть поставить , в предыдущей строке.
####bash
vi /etc/docker/daemon.json

{
```

```
  "insecure-registries": ["nexus:5000"]
}
####
```

После редактирования файл daemon.json должен иметь следующее содержание.

```
{
"log-driver": "journald",
"live-restore": true,
"icc": false,
"disable-validation": true,
"insecure-registries": [ "nexus:5000" ]
}
```

Выполнить команды.

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

Установить бинарные файлы "node exporter", "promtail", "redis exporter", на Redis VM.

```
#Перейти на VM redis
####
mkdir /home/redos/for_install
curl http://nexus:8081/repository/raw-soneric/dirs/monitoring.tgz -o
/home/redos/for_install/monitoring.tgz
cd for_install
tar -xvzf monitoring.tgz

tar -xzf redis_exporter-v1.45.0.linux-amd64.tar.gz

sudo mkdir /opt/stat
sudo mkdir /opt/stat/node_exporter
sudo mkdir /opt/stat/promtail
sudo mkdir /opt/stat/redis

sudo mv redis_exporter-v1.45.0.linux-amd64/redis_exporter /opt/stat/redis/
sudo chmod +x /opt/stat/redis/redis_exporter
sudo vi /etc/systemd/system/redis-exporter.service

[Unit]
Description=Redis Exporter
After=network.target

[Service]
User=redis_exporter
Group=redis_exporter
ExecStart=/opt/stat/redis/redis_exporter --redis.addr=redis://localhost:6379
Restart=always

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

Выполнить команду.

```
####

sudo useradd -rs /bin/false redis_exporter

####
```

```
####

sudo useradd -rs /bin/false node_exporter

sudo cp promtail/promtail* /opt/stat/promtail/
sudo cp node_exporter/node* /opt/stat/node_exporter/

sudo rm /opt/stat/promtail/*service
sudo rm /opt/stat/node_exporter/*service

sudo chown node_exporter:node_exporter /opt/stat/node_exporter/*

sudo cp node_exporter/node-exporter.service /etc/systemd/system/
sudo cp promtail/promtail-exporter.service /etc/systemd/system/

####
```

Исправить путь запуска на "ExecStart=/opt/stat/node_exporter/node_exporter".

```
sudo vi /etc/systemd/system/node-exporter.service

#вместо
ExecStart=/opt/stat/node_exporter
#Сделать
ExecStart=/opt/stat/node_exporter/node_exporter

sudo chmod +x  /opt/stat/promtail/*
sudo chmod +x  /opt/stat/node_exporter/*
```

Поменять в promtail.yaml IP address на адрес мониторинговой машины.

```
####
cd /opt/stat/promtail/
sudo vi promtail.yaml
Заменить
- url: http://10.82.0.79:3100/loki/api/v1/push
на
- url: http://10.200.127.5:3100/loki/api/v1/push

sudo systemctl daemon-reload
```

```
sudo systemctl enable node-exporter
sudo systemctl enable redis-exporter
sudo systemctl enable promtail-exporter

sudo systemctl start node-exporter
sudo systemctl start redis-exporter
sudo systemctl start promtail-exporter

sudo systemctl status node-exporter
sudo systemctl status redis-exporter
sudo systemctl status promtail-exporter
```

Перейти на Виртуальную Машину (VM) Nexus.

```
####
mkdir /home/redos/for_install
curl http://nexus:8081/repository/raw-soneric/dirs/monitoring.tgz -o
/home/redos/for_install/monitoring.tgz
cd for_install
tar -xvzf monitoring.tgz
####

sudo mkdir /opt/stat
sudo mkdir /opt/stat/node_exporter
sudo mkdir /opt/stat/promtail

####
```

```
####

sudo useradd -rs /bin/false node_exporter

sudo cp promtail/promtail* /opt/stat/promtail/
sudo cp node_exporter/node* /opt/stat/node_exporter/

sudo rm /opt/stat/promtail/*service
sudo rm /opt/stat/node_exporter/*service

sudo chown node_exporter:node_exporter /opt/stat/node_exporter/*

sudo cp node_exporter/node-exporter.service /etc/systemd/system/
sudo cp promtail/promtail-exporter.service /etc/systemd/system/

####
```

Исправить путь запуска на "ExecStart=/opt/stat/node_exporter/node_exporter".

```
sudo vi /etc/systemd/system/node-exporter.service

#вместо
ExecStart=/opt/stat/node_exporter
#Сделать
ExecStart=/opt/stat/node_exporter/node_exporter
```

```
sudo chmod +x  /opt/stat/promtail/*
sudo chmod +x  /opt/stat/node_exporter/*
```

Поменять в promtail.yaml IP address на адрес мониторинговой машины.

```
####
cd /opt/stat/promtail/
sudo vi promtail.yaml
Заменить
- url: http://10.82.0.79:3100/loki/api/v1/push
на
- url: http://10.200.127.5:3100/loki/api/v1/push

sudo systemctl daemon-reload

sudo systemctl enable node-exporter
sudo systemctl enable promtail-exporter

sudo systemctl start node-exporter
sudo systemctl start promtail-exporter

sudo systemctl status node-exporter
sudo systemctl status promtail-exporter
```

Перейти на Виртуальную Машину (VM) мониторинга.

```
mkdir /home/redos/for_install
curl http://nexus:8081/repository/raw-soneric/dirs/docker_mon.tgz -o
/home/redos/for_install/docker_mon.tgz
cd for_install
tar -xvzf docker_mon.tgz

mkdir /home/redos/alertmanager
mkdir /home/redos/prometheus
mkdir /home/redos/loki
mkdir /home/redos/promtail
mkdir /home/redos/grafana
```

```
cd ~

docker-compose up -d
#Проверка
docker ps -a
docker logs --details alertmanager
#Если менять настройки, то остановить созданные контейнеры.
docker compose down
```

```
mkdir /home/redos/grafana/provisioning/datasources
cp dashboards/* /home/redos/grafana/dashboards/
cp provisioning/datasources/* ../../grafana/provisioning/datasources/
```

Поменять настройки.

```
docker compose up -d
```

Перезапуск Docker.

```
docker compose down && docker compose up -d
```

Произвести установку компонента (plugin) для Redis.

```
####
sudo dnf install zip

docker compose down
curl http://nexus:8081/repository/raw-soneric/dirs/redis-datasource-2.2.0.zip -o
/home/redos/for_install/redis-datasource-2.2.0.zip
cd for_install
#Unzip an archive to the plugins/ folder for local Grafana installation or Docker's
volume.
####
unzip redis-datasource-2.2.0.zip -d /home/redos/grafana/plugins
####
#Check permissions for the redis-datasource binaries:
ls -lrt /home/redos/grafana/plugins/redis-datasource*
```

Установить компонента (plugin) Redis Application.

```
#Redis Application plugin
#Download the latest version from the Releases "redis-app-2.2.1.zip".
####
curl http://nexus:8081/repository/raw-soneric/dirs/redis-app-2.2.1.zip -o
/home/redos/for_install/redis-app-2.2.1.zip
####
#Unzip an archive to the plugins/ folder for local Grafana installation or Docker's
volume.
####
unzip redis-app-2.2.1.zip -d /home/redos/grafana/plugins
####
```

Установить компонента (plugin) Redis Explorer.

```
#Redis Explorer plugin
#Dependencies
#Redis Explorer plugin requires Redis Data Source and Redis Application plugin.
#Download the latest version from the Releases. "redis-explorer-app-2.1.1.zip"
####
curl http://nexus:8081/repository/raw-soneric/dirs/redis-explorer-app-2.1.1.zip -o
/home/redos/for_install/redis-explorer-app-2.1.1.zip
####
```

```
#Unzip an archive to the plugins/ folder for local Grafana installation or Docker's
volume.
####
unzip redis-explorer-app-2.1.1.zip -d /home/redos/grafana/plugins
####
cd /home/redos/
docker compose up -d
```

Произвести проверку установленных компонентов.

```
#Проверка
docker ps -a
docker logs --details grafana


!grafana
admin/admin
new= admin/Soneric1@
```

Сконфигурировать управление k8s с мониторинговой Виртуальной Машины (VM).

```
#Если нужно управление k8s с мониторинг VM
sudo su
mkdir -p /home/redos/.kube/
scp redos@10.200.127.53:/home/redos/.kube/config /home/redos/.kube/config
curl http://nexus:8081/repository/raw-soneric/dirs/kubectl_1.32.0 -o
/home/redos/for_install/kubectl
cd /home/redos/for_install
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
kubectl version --client
sudo chown redos:redos  /home/redos/.kube/config
```

Перейти на master K8s Виртуальную Машину (VM).

```
#Перейти на master K8s VM
####
mkdir /home/redos/for_install
curl http://nexus:8081/repository/raw-soneric/dirs/mon_manif.tgz -o
/home/redos/for_install/mon_manif.tgz
cd for_install
tar -xvzf mon_manif.tgz
```

На всех master K8s Виртуальных Машинах (VM) выполнить.

```
####
sudo su
cat /etc/systemd/system/kubelet.service.d/kubeadm.conf
cat /var/lib/kubelet/kubeadm-flags.env

cd /etc/systemd/system/kubelet.service.d
vi kubeadm.conf
```

```
Add  "--authentication-token-webhook=true"  into  "KUBELET_AUTHZ_ARGS"
After that, it would be like
Environment="KUBELET_AUTHZ_ARGS=--authorization-mode=Webhook --client-ca-
file=/etc/kubernetes/pki/ca.crt --authentication-token-webhook=true"

cd /var/lib/kubelet/
vi kubeadm-flags.env
Найдите строку:
KUBELET_KUBEADM_ARGS="--container-runtime-endpoint=unix:///var/run/crio/crio.sock --
pod-infra-container-image=registry.k8s.io/pause:3.9"
Замените на:
KUBELET_KUBEADM_ARGS="--container-runtime-endpoint=unix:///var/run/crio/crio.sock --
pod-infra-container-image=registry.k8s.io/pause:3.9 --authentication-token-
webhook=true --authorization-mode=Webhook"
```

В качестве альтернативного выриант можно использовать “sed”.

```
#или через sed
###
sed -i 's/ca.crt/ca.crt --authentication-token-webhook=true/'
/etc/systemd/system/kubelet.service.d/kubeadm.conf
sed -i 's/pause:3.9/pause:3.9 --authentication-token-webhook=true --authorization-
mode=Webhook/' /var/lib/kubelet/kubeadm-flags.env
#проверка
cat /etc/systemd/system/kubelet.service.d/kubeadm.conf
cat /var/lib/kubelet/kubeadm-flags.env
###
systemctl daemon-reload
systemctl restart kubelet.service
exit
####
```

На всех необходимо внести изменения в файл kube-scheduler.yamlследующим образом.

```
#На мастерах
kube-scheduler.yaml
sudo vi /etc/kubernetes/manifests/kube-scheduler.yaml
Найдите строку:

--bind-address=127.0.0.1
Замените на:

--bind-address=0.0.0.0

kube-controller-manager

sudo vi /etc/kubernetes/manifests/kube-controller-manager.yaml
внести изменения в строчке

Найдите строку:

--bind-address=127.0.0.1
Замените на:
```

```
--bind-address=0.0.0.0


sudo vi /etc/kubernetes/manifests/etcd.yaml
Найдите строку:

--listen-metrics-urls=http://127.0.0.1:2381
Замените на:

--listen-metrics-urls=http://0.0.0.0:2381
Исправить kube-proxy, для отдачи метрик

#только на master-0
kubectl -n kube-system edit configmap kube-proxy
Найдите строку:

metricsBindAddress: 127.0.0.1:10249
Замените на:

metricsBindAddress: 0.0.0.0:10249
```

Для установки prometheus-stack в k8s необходимо выполнить.

```
####
#Установка prometheus-stack в k8s
####
cd /home/redos/for_install
kubectl apply --server-side -f manifests/setup
kubectl wait \
      --for condition=Established \
      --all CustomResourceDefinition \
      --namespace=monitoring
kubectl apply -f manifests/
####
#Если удалить
kubectl delete --ignore-not-found=true -f manifests/ -f manifests/setup
####
#другой вариант запуска
kubectl create -f manifests/setup
kubectl get ns monitoring
kubectl create -f manifests/
####
kubectl apply -f promtail-serviceaccount.yml
```

```
kubectl get pods -n monitoring -w
kubectl get svc -n monitoring

kubectl get networkpolicies -n monitoring
kubectl -n monitoring delete networkpolicies.networking.k8s.io --all
```

```
#Prometheus авто патч не делаем:
#kubectl --namespace monitoring patch svc prometheus-k8s -p '{"spec": {"type":
"NodePort"}}'
```

```
#kubectl get svc -n monitoring  prometheus-k8s -o yaml
kubectl apply -f prometheus_svc.yaml

#Alertmanager авто патч не делаем:
#kubectl --namespace monitoring patch svc alertmanager-main -p '{"spec": {"type":
"NodePort"}}'
#kubectl get svc -n monitoring  alertmanager-main -o yaml
kubectl apply -f alertmanager_svc.yaml
```

```
#kubectl get secrets
#kubectl get secrets -n monitoring alertmanager-main -o "jsonpath={
.data['alertmanager\.yaml'] }" | base64 -d

#kubectl get clusterrolebinding prometheus-operator -o=yaml
#kubectl get clusterrole prometheus-operator -o=yaml
#kubectl describe clusterrole prometheus-operator
#kubectl describe clusterrole kube-state-metrics
kubectl get events -n monitoring --field-selector involvedObject.name=alertmanager-
main-0
```

Добавить paused:true перед image: nexus:5000/repository/docker-soneric/alertmanager:v0.27.0.

```
#alertmanager fix
##add paused:true перед image: nexus:5000/repository/docker-
soneric/alertmanager:v0.27.0

spec:
  paused: true
  image: nexus:5000/repository/docker-soneric/alertmanager:v0.27.0
  nodeSelector:
    kubernetes.io/os: linux
```

```
kubectl -n monitoring edit alertmanagers.monitoring.coreos.com

#kubectl -n monitoring get statefulset.apps/alertmanager-main -o yaml > dump.yaml
kubectl -n monitoring delete statefulsets.apps alertmanager-main
```

```
#добавить
spec:
      hostNetwork: true
      dnsPolicy: ClusterFirst
      initContainers:
      - args:
        - --watch-interval=0

hostNetwork: true
#remove both liveness and readiness probe (for the time being)
#Подготовленный файл. "alert_dump.yaml"
#Закоментировать в файле параметр  hostNetwork: true
vi for_install/alert_dump.yaml

поменять
```

```
hostNetwork: true
на
#hostNetwork: true
```

```
####
kubectl create -f alert_dump.yaml
####
```

После установки компонентов мониторинга необходимо установить сервисы UDR.

3.3 Установка сервисов UDR

Общим репозиторием в ПК UDR является компонент Nexus. Установка сервисов ПК UDR производится из репозитория Nexus. При помощи команды curl необходимо скачать архив из репозитория Nexus. Архив содержит все необходимые для установки модули. Необходимо также разархивировать содержимое, как показано в примере.

```
####
curl http://nexus:8081/repository/raw-soneric/k8s_install/HA_dev2g4g_install.tgz -o
/home/redos/HA_dev2g4g_install.tgz
tar -xvzf HA_dev2g4g_install.tgz
####
```

Далее необходимо выполнить команду kubectl apply для создания ресурсов, указав при этом соответствующий конфигурационный yaml-файл, как показано в примере.

```
kubectl apply -f 2g4g_install/manifest_2Gauth.yaml -f 2g4g_install/2gauthservice-
config.yaml
kubectl apply -f 2g4g_install/manifest_2Gclient.yaml -f 2g4g_install/2gclient-
config.yaml
kubectl apply -f 2g4g_install/manifest_2Glinkstatus.yaml -f
2g4g_install/2glinkstatushandler-config.yaml
kubectl apply -f 2g4g_install/manifest_2Grouter.yaml -f 2g4g_install/2grouterservice-
config.yaml
kubectl apply -f 2g4g_install/manifest_2Gtcap.yaml -f 2g4g_install/2gtcaphandler-
config.yaml
kubectl apply -f 2g4g_install/manifest_2Gupdate.yaml -f
2g4g_install/2gupdatelocationservice-config.yaml
kubectl apply -f 2g4g_install/manifest_4Gauth.yaml -f 2g4g_install/4gauthservice-
config.yaml
kubectl apply -f 2g4g_install/manifest_4Gclient.yaml -f 2g4g_install/4gclient-
config.yaml
kubectl apply -f 2g4g_install/manifest_4Gnotify.yaml -f 2g4g_install/4gnotifyservice-
config.yaml
kubectl apply -f 2g4g_install/manifest_4Grouter.yaml -f 2g4g_install/4grouterservice-
config.yaml
kubectl apply -f 2g4g_install/manifest_4gupdate.yaml -f
2g4g_install/4gupdatelocationservice-config.yaml
```

Прописать сервисы ПК UDR, как показано на примере, с использованием команды kubectl apply.

```
kubectl apply -f k8sinstall/5_ha_4g_serv_v1_laba.yaml
kubectl apply -f k8sinstall/6_ha_2g_svc_laba.yaml
kubectl apply -f k8sinstall/7_ha_2glinkstatushandler.yaml
```

3.4 Проверка и тестирование UDR после установки

4 История изменений

История изменений фиксируется в Таблице 4-1.

Таблица 4-1 История изменений

Дата	Версия документации	Версия UDR	Комментарии
20.06.2025	0.1	PoC	Первая версия документа
06.08.2025	0.2	PoC	Добавлена секция K8S
11.08.2025	0.3	PoC	Добавлена секция ПК UDR
06.09.2025	0.4	PoC	Добавлена секция по установке Graphana
26.09.2025	05.	PoC	Пройдена проверка для PoC

Список сокращений

Сокращение	Определение
UDR	User Data Repository – домашний репозиторий абонентов;
ПК	Программный комплекс;
API	Application Programming Interface - набор правил и протоколов, позволяющих приложениям взаимодействовать друг с другом;
HLR	Home Location Register - домашний регистр местоположения мобильных абонентов сетей стандартов 2G/3G;
HSS	Home Subscriber Server – сервер абонентских данных сети стандарта 4G;
AUC	Authentication Center – центр аутентификации мобильных абонентов;
Database	Централизованная база данных;
2G	Стандарт мобильной связи второго поколения;
3G	Стандарт мобильной связи третьего поколения;
4G	Стандарт мобильной связи четвертого поколения;
5G	Стандарт мобильной связи пятого поколения;

SCTP	Stream Control Transmission Protocol – транспортный протокол для сигнализации;
M3UA	MTP-3 User Adaptation Layer - протокол адаптации пользовательского уровня;
SCCP	Signalling Connection Control Part — протокол сетевого уровня, используемый в телекоммуникациях для передачи сигнальных сообщений по сети;
TCAP	Transaction Capabilities Application Part - часть Общеканальной системы сигнализации №7 (SS7);
SS7	Signalling System 7- это набор протоколов телефонной сигнализации;
MAP	Mobile Application Part – протокол сигнализации для мобильных сетей 2G/3G;
Diameter	Протокол сигнализации для сетей 4G;
S6a	Интерфейс между HSS и MME в сети 4G;
STP	Signaling Transfer Point - компонент в телекоммуникационных сетях, который обеспечивает маршрутизацию и передачу сигнальных сообщений между различными элементами сети;
DRA	Протокол обмена сообщениями аутентификации, авторизации для приложений доступа к сетям и мобильности данных в сетях 3G, 4G, IMS и LTE;
3GPP	The 3rd Generation Partnership Project - содружество телекоммуникационных ассоциаций, которые разрабатывают протоколы для мобильных телекоммуникаций;
SMS	Short Message Service - технология приёма и передачи коротких текстовых сообщений с помощью мобильного телефона;
IMSI	International Mobile Subscriber Identity – международный идентификатор мобильного абонента;
CLI	Command line interface - интерфейс командной строки, который позволяет взаимодействовать с компьютером с помощью текстовых команд заданного формата;
MSC	Mobile Switching Center - мобильный коммутационный центр, или мобильный коммутационный сервер сотовых сетей 2G и 3G;

SGSN	Serving GPRS Support Node - узел обслуживания абонентов пакетной сети передачи данных сетей GSM и UMTS. Это аналог MSC для пакетной сети передачи данных;
MSISDN	Mobile Subscriber Integrated Services Digital Number - номер мобильного абонента;
GPRS	General Packet Radio Service - надстройка над технологией GSM, осуществляющая пакетную передачу данных;
MME	Mobility Management Entity – узел управления мобильностью в сети 4G;
TPS	Time per second - число транзакций в секунду;
SCP	Service Control Point - пункт управления услугами;
VLR	Visitor Location Register - гостевой регистр местоположения;
SSH	Secure Shell - протокол прикладного уровня для удалённого обмена данными между устройствами;
SFTP	Secure File Transfer Protocol - протокол прикладного уровня передачи файлов, работающий поверх безопасного канала;
NTP	Network Time Protocol - протокол сетевого времени, который помогает синхронизировать время устройств;
CI\CD	Continuous Integration/Continuous Delivery - методология разработки, которая позволяет автоматизировать процесс создания ПК;