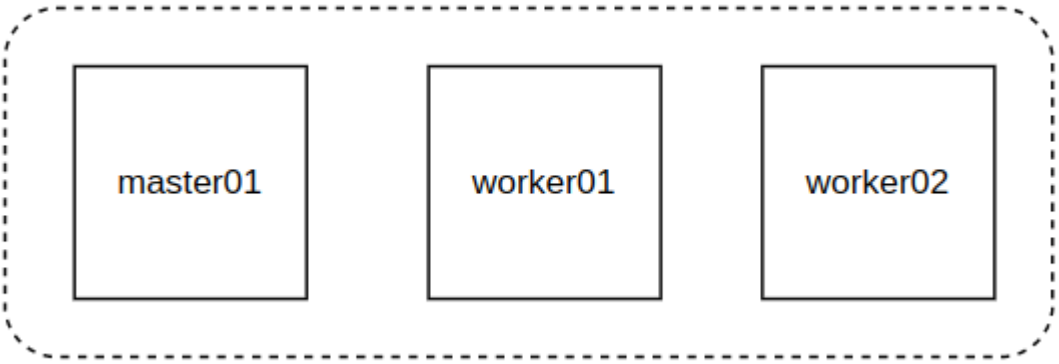
Настройка кластера Kubernetes

ОТ РОМАН 27.01.2023 KUBERNETES

Содержание [скрыть]

В этой публикации я покажу, как выполняется настройка кластера Kubernetes. Развертывание будет включать один мастер узел и два узла для рабочей нагрузки. Сразу скажу, что это руководство не полностью моё. За основу я брал вот эту статью, но адаптировал её под свою конфигурацию. Что и побудило меня написать данную статью, чтобы в следующий раз можно было быстро по материалам статьи подготовить кластер Kubernetes.

Если изобразить схематично, то итоговый кластер будет выглядеть следующим образом:



Да, на производственный кластер с отказоустойчивостью всех узлов такая архитектура не претендует, но для целей разработки или изучения Kubernetes этого должно быть вполне достаточно. Если вам не пока хочется заниматься развертыванием такого кластера, а поработать с Kubernetes есть желание, то вы можете использовать развертывание на базе Minikube.

Я не претендую на звание гуру Kubernetes, поэтому, если вы нашли какие-то огрехи в конфигурации, то пишите в комментариях, что можно улучшить/оптимизировать.

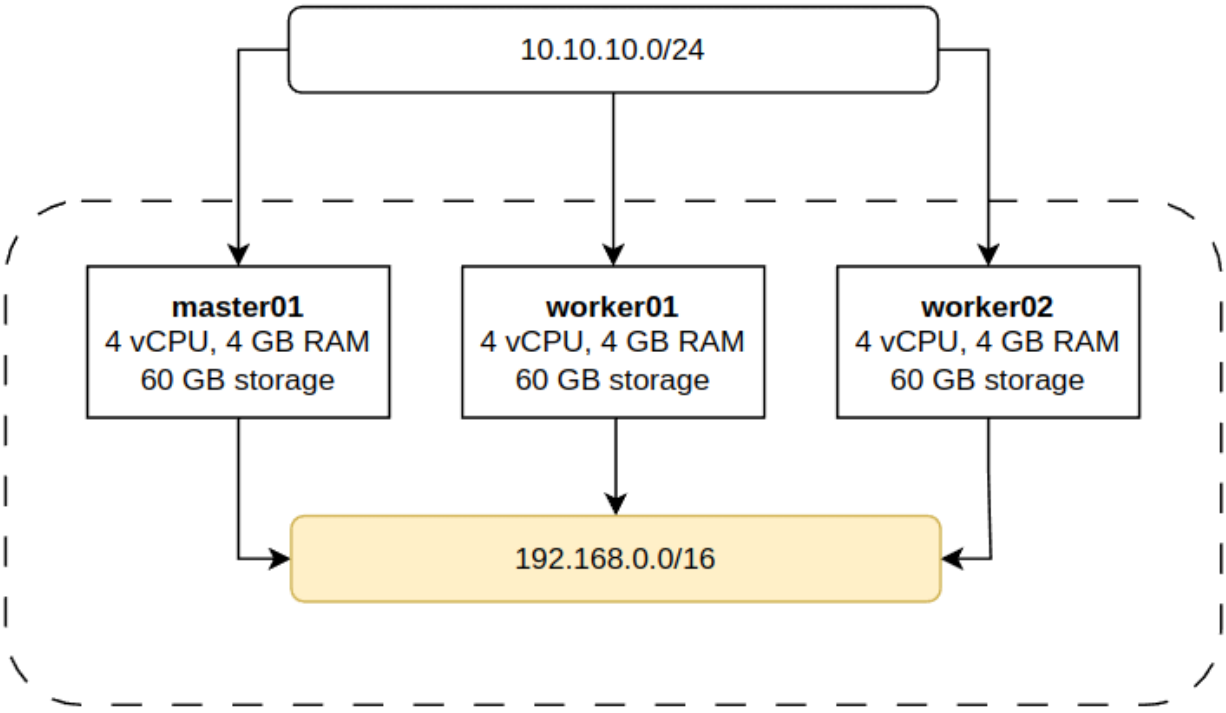
Предварительные требования к операционным системам

В качестве операционной системы я буду использовать Ubuntu Server 22.04. О том, как её развернуть есть статья на моем блоге, либо есть много других руководств на просторах интернета. Вы можете использовать другой Linux дистрибутив, но я написал эту статью с прицелом исключительно на Ubuntu Server 22.04. Для вашего дистрибутива какие-то из шагов могут отличаться. Например, настройки исключений брандмауэра.

Для всех узлов кластера я буду использовать одинаковую конфигурацию аппаратных ресурсов:

- 4 vCPU.
- 4 ГБ оперативной памяти.
- 40 ГБ диск для операционной системы.

В качестве внутренней сети кластера я буду использовать подсеть 192.168.0.0/16. Трафик к узлам кластера будет приходить из моей домашней подсети 10.10.10.0/24.



При необходимости можно уменьшите количество vCPU и оперативной памяти для узлов.

Порты для сетевого взаимодействия

Для корректной работы Kubernetes необходимо, чтобы между узлами кластера был доступен определенный набор портов, который приведен в документации. Продублирую перечень портов.

Перечень необходимых доступных портов для корректной работы плоскости управления:

Протокол	Направление	Порты	Назначение	Кем используется
TCP	Входящее	6443	Kubernetes API server	Все узлы
TCP	Входящее	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Входящее	10250	Kubelet API	Kubelet API, плоскость управления

TCP	Входящее	10259	kube-scheduler	kube-scheduler
TCP	Входящее	10257	kube-controller-manager	kube-controller-manager

Перечень необходимых доступных портов для корректной работы узлов рабочей нагрузки:

Протокол	Направление	Порты	Назначение	Кем используется
TCP	Входящее	10250	Kubelet API	Kubelet API, плоскость управления
TCP	Входящее	30000-32767	NodePort Services	Все узлы

Настройка кластера Kubernetes

Как я уже говорил выше – для всех узлов кластера я буду использовать ОС Ubuntu Server 22.04. Поэтому перед началом последующих шагов я выполню её установки и минимальную первоначальную настройку.

Также перед выполнение последующий действий необходимо отключить файл подкачки на всех узлах кластера:

```
sudo swapoff -a
```

Чтобы при последующий перезагрузках узлов кластера файл подкачки не активировался вновь необходимо добавить соответствующее задание в файл crontab на всех узлах кластера:

```
sudo crontab -e
```

Добавим следующее простое задание:

```
@reboot /sbin/swapoff -a
```

```
  GNU nano 6.2
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
@reboot /sbin/swapoff -a
```

Сохраним внесенные изменения и закроем файл crontab. В случае успешного добавления задания должен отобразиться соответствующий вывод на консоль:

```
crontab: installing new crontab
roman@master01:~$
```

```
crontab: installing new crontab
roman@master01:~$
```

Теперь приступим к выполнению других шагов.

Установка и настройка cri-o

В качестве интерфейса выполнения контейнеров (CRI) я буду использовать cri-o, т.к. начиная с версии Kubernetes 1.24 с docker не все так просто.

Теперь все готово для установки cri-o. Установим необходимые пакеты и подготовим переменные:

```
sudo apt update
sudo apt install -y apt-transport-https ca-certificates curl gnupg2
software-properties-common
export OS_VERSION=xUbuntu_20.04
export CRIO_VERSION=1.23
```

Установим gpg ключи:

```
curl -fsSL
https://download.opensuse.org/repositories/devel:/kubic:/libcontain
ers:/stable/$OS_VERSION/Release.key | sudo gpg --dearmor -o
/usr/share/keyrings/libcontainers-archive-keyring.gpg
curl -fsSL
https://download.opensuse.org/repositories/devel:/kubic:/libcontain
ers:/stable:/cri-o:/$CRIO_VERSION/$OS_VERSION/Release.key | sudo
gpg --dearmor -o /usr/share/keyrings/libcontainers-crio-archive-
keyring.gpg
```

Добавим файлы репозитория:

```
echo "deb [signed-by=/usr/share/keyrings/libcontainers-archive-
keyring.gpg]
https://download.opensuse.org/repositories/devel:/kubic:/libcontain
ers:/stable/$OS_VERSION/ /" | sudo tee
/etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
echo "deb [signed-by=/usr/share/keyrings/libcontainers-crio-
archive-keyring.gpg]
https://download.opensuse.org/repositories/devel:/kubic:/libcontain
ers:/stable:/cri-o:/$CRIO_VERSION/$OS_VERSION/ /" | sudo tee
/etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-
o:$CRIO_VERSION.list
```

Запустим установку пакетов для cri-o:

```
sudo apt update
sudo apt install -y cri-o cri-o-runc
```

Создадим конфигурационные файлы для cri-o:

```
cat <<EOF | sudo tee /etc/modules-load.d/crio.conf
overlay
br_netfilter
EOF

# Set up required sysctl params, these persist across reboots.
cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
```

Загрузим модули для корректной работы сети нашего кластера:

```
sudo modprobe overlay
sudo modprobe br_netfilter
```

Перечитаем параметры конфигурации ядра со всех конфигурационных файлов:

```
sudo sysctl --system
```

Запустим службы:

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

Установка kubeadm, kubelet и kubectl

Установим gpg ключи и добавим репозитории:

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-
keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" |
sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Установим kubelet, kubeadm и kubectl:

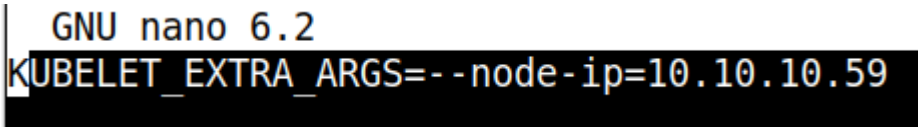
```
sudo curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg
sudo apt update -y
sudo apt install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Добавим IP-адрес каждого узда в параметр KUBELET_EXTRA_ARGS:

```
sudo nano /etc/default/kubelet
```

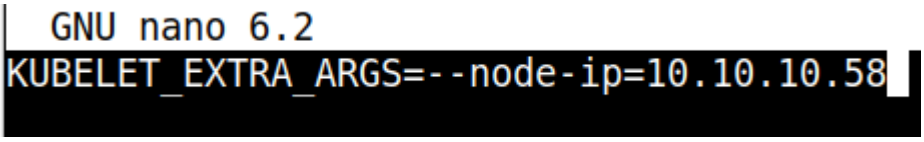
Для каждого узла кластера я укажу его IP-адрес. Конфигурация master01:

```
KUBELET_EXTRA_ARGS=--node-ip=10.10.10.59
```



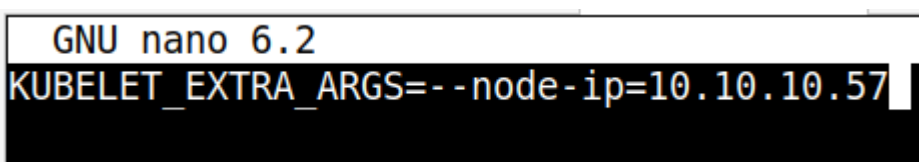
Для worker01:

```
KUBELET_EXTRA_ARGS=--node-ip=10.10.10.58
```



Для worker02:

```
KUBELET_EXTRA_ARGS=--node-ip=10.10.10.57
```



Сохраняем внесенные изменения.

Инициализация кластера

Подготовим первый узел кластера, которые будет заниматься плоскостью управления. Напомню, что основная сеть для общения между узлами кластера – подсеть 10.10.10.0/24. Для взаимодействия подов кластера я буду использовать подсеть 192.168.0.0/16. IP-адрес master01 – 10.10.10.59/24. Подготовим соответствующие переменные окружения на master01:

```
IPADDR="10.10.10.59"
NODENAME=$(hostname -s)
POD_CIDR="192.168.0.0/16"
```

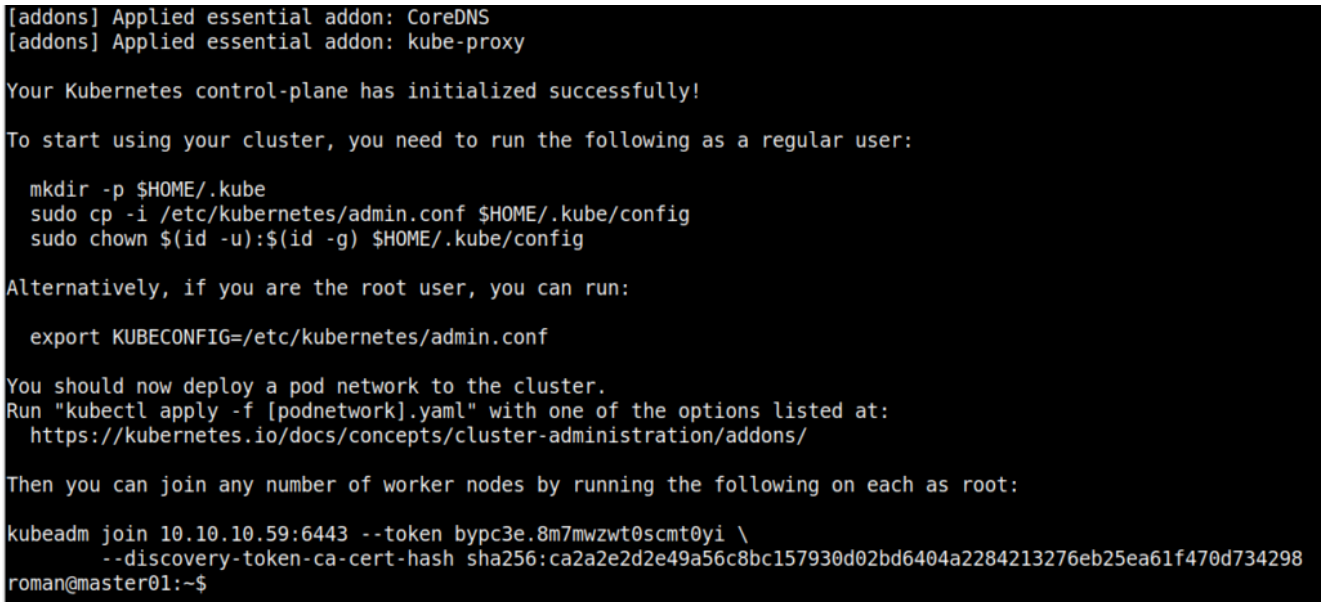
Инициализируем первый узел кластера:

```
sudo kubeadm init --apiserver-advertise-address=$IPADDR  --
apiserver-cert-extra-sans=$IPADDR  --pod-network-cidr=$POD_CIDR --
node-name $NODENAME --ignore-preflight-errors Swap
```

Пример сообщения об успешном завершении инициализации кластера:

```
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

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

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

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

  export KUBECONFIG=/etc/kubernetes/admin.conf

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

```
administration/addons/

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

kubeadm join 10.10.10.59:6443 --token bypc3e.8m7mwzwt0scmt0yi \
        --discovery-token-ca-cert-hash
sha256:ca2a2e2d2e49a56c8bc157930d02bd6404a2284213276eb25ea61f470d73
4298
roman@master01:~$
```



Здесь же мы видим сообщение о том, как мы можем настроить подключение к нашему кластеру:

```
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
```

Также kubeadm подготовил для нас команду для присоединения дополнительных узлов кластера (в вашем случае команда будет немного отличаться):

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

```
kubeadm join 10.10.10.59:6443 --token bypc3e.8m7mwzwt0scmt0yi \
        --discovery-token-ca-cert-hash
sha256:ca2a2e2d2e49a56c8bc157930d02bd6404a2284213276eb25ea61f470d734298
```

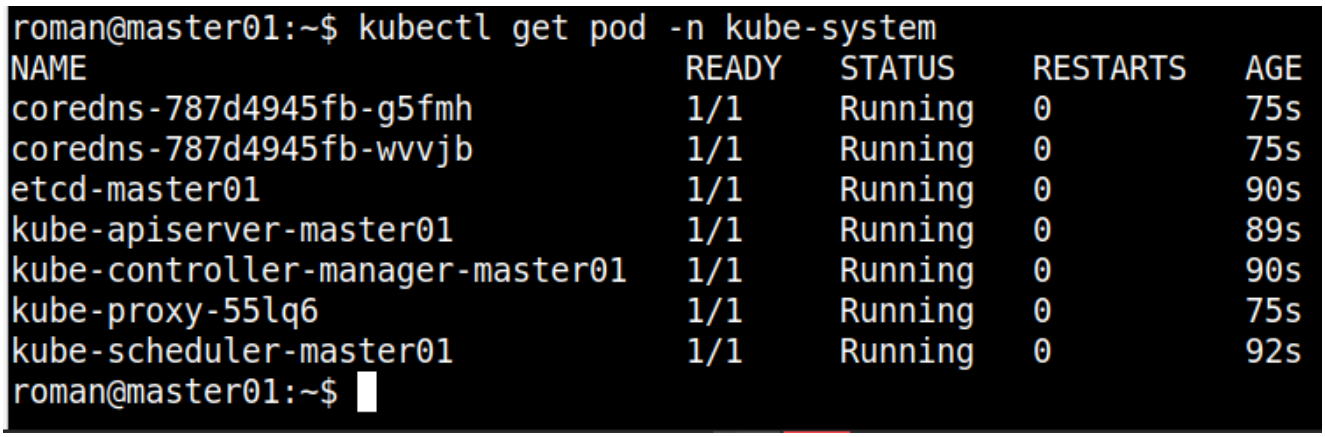
Выполним настройку подключения:

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

Попробуем получить информация об узлах нашего кластера Kubernetes:

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

```
roman@master01:~$ kubectl get pod -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-787d4945fb-g5fmh            1/1     Running   0          75s
coredns-787d4945fb-wvvjb            1/1     Running   0          75s
etcd-master01                       1/1     Running   0          90s
kube-apiserver-master01             1/1     Running   0          89s
kube-controller-manager-master01    1/1     Running   0          90s
kube-proxy-55lq6                    1/1     Running   0          75s
kube-scheduler-master01             1/1     Running   0          92s
roman@master01:~$
```



Успешный ответ от кластера говорит о том, что онг успешно инициализирован и мы можем выполнять подключение дополнительных узлов.

При использовании стандартных настроек на сервер управления (master01) не будет размещаться рабочая нагрузка. В моем случае такое поведение и было запланировано изначально. Если же вы планируете размещение рабочей нагрузки на мастер узлах, то выполните следующую команду:

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

Установка плагина Calico для работы с сетью

Установка очень проста:

> Если вы используете сеть для контейнеров (переменная POD_CIDR) отличную от 192.168.0.0/16, то вам предварительно необходимо загрузить файл манифеста и раскомментировать переменную CALICO_IPV4POOL_CIDR, в которой указать CIDR вашей сети для подов:
> 1. curl https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/calico.yaml -O
> 2. [отредактировать манифест]
> 3. kubectl apply -f calico.yaml

> Ссылку на инструкцию по установке последенй версии плагина можно найти на сайте вендора.

```
kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/calico.yaml
```

```
roman@kube-control-plane:~$ kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
roman@kube-control-plane:~$
```

Дожидаемся окончания выполнения всех заданий по развертыванию ресурсов. Проверяем:

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

```
roman@kube-control-plane:~$ kubectl get pod -n kube-system
NAME                                       READY   STATUS    RESTARTS   AGE
calico-kube-controllers-674fff74c8-wbckt   1/1     Running   0          48s
calico-node-nm2jk                          0/1     Running   0          48s
coredns-5d78c9869d-49mq7                   1/1     Running   0          3m46s
coredns-5d78c9869d-mrjd4                   1/1     Running   0          3m46s
etcd-kube-control-plane                    1/1     Running   0          4m
kube-apiserver-kube-control-plane          1/1     Running   0          4m
kube-controller-manager-kube-control-plane 1/1     Running   0          4m
kube-proxy-8wkrx                           1/1     Running   0          3m46s
kube-scheduler-kube-control-plane          1/1     Running   0          4m
roman@kube-control-plane:~$
```

Подключение дополнительных узлов кластера

Осталось совсем немного – добавить два узла кластера для размещения рабочей нагрузки.

Как я уже говорил выше, при завершении инициализации кластера kubeadm отображает команду для подключения дополнительных узлов. Если вы по каким-то причинам не скопировали это сообщение, то сгенерировать новый токен для подключения вы можете вот такой командой:

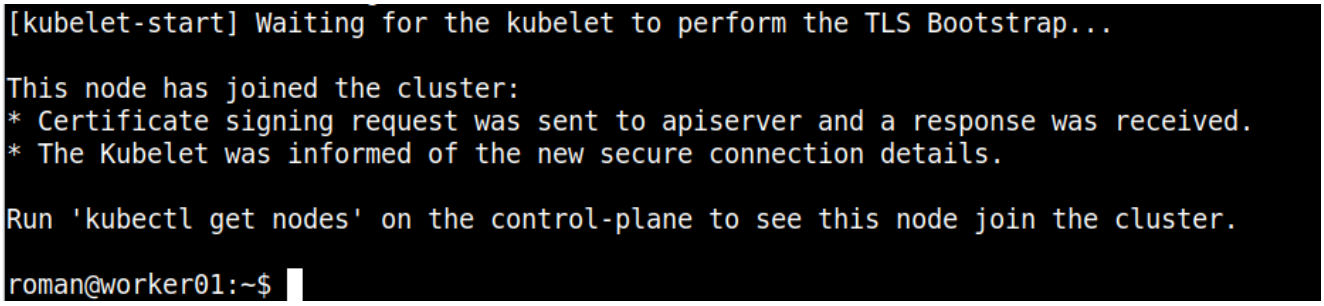
```
kubeadm token create --print-join-command
```

На узле worker01 я выполню команду для подключения к кластеру Kubernetes:

```
sudo kubeadm join 10.10.10.59:6443 --token bypc3e.8m7mwzwt0scmt0yi
\
        --discovery-token-ca-cert-hash
sha256:ca2a2e2d2e49a56c8bc157930d02bd6404a2284213276eb25ea61f470d73
4298
```

Пример сообщения об успешном завершении операции присоединения к кластеру:

```
[kubelet-start] Waiting for the kubelet to perform the TLS
Bootstrap...

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

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

roman@worker01:~$
```



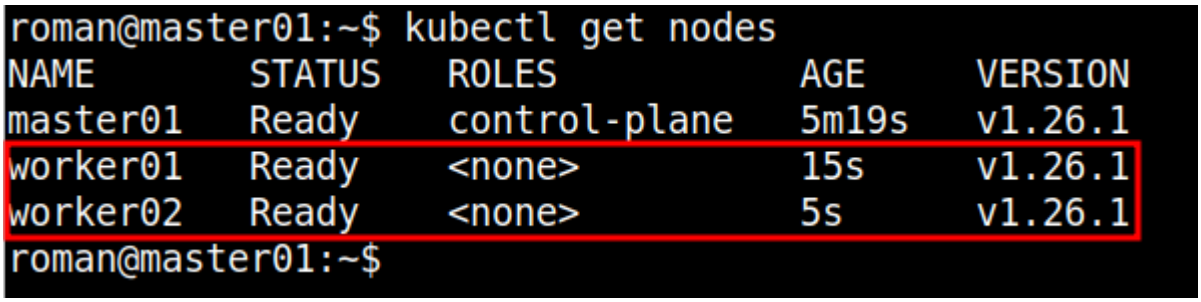
Выполним аналогичную команду присоединения на втором рабочем узле кластера – worker02:

```
sudo kubeadm join 10.10.10.59:6443 --token bypc3e.8m7mwzwt0scmt0yi
\
        --discovery-token-ca-cert-hash
sha256:ca2a2e2d2e49a56c8bc157930d02bd6404a2284213276eb25ea61f470d73
4298
```

Теперь проверим информацию об узлах кластера. На сервере master01 выполним следующую команду:

```
kubectl get nodes
```

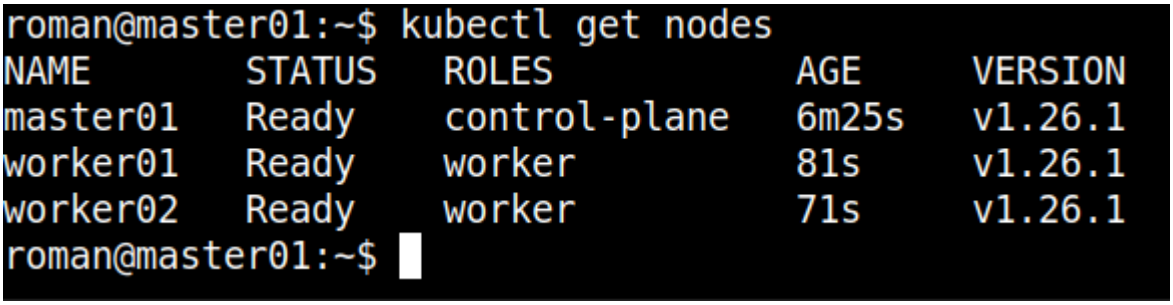
```
roman@master01:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE     VERSION
master01   Ready    control-plane   5m19s   v1.26.1
worker01   Ready    <none>          15s     v1.26.1
worker02   Ready    <none>          5s      v1.26.1
roman@master01:~$
```



Видим, что оба узла кластера добавлены и находят в состоянии “Ready”, т.е. узлы видят мастер узел и наоборот.

Присвоить роль узлам для рабочей нагрузки можно следующими командами:

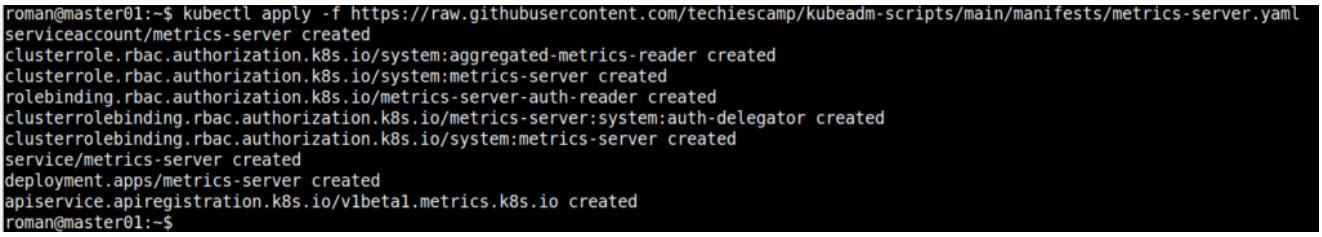
```
kubectl label node worker01  node-role.kubernetes.io/worker=worker
kubectl label node worker02  node-role.kubernetes.io/worker=worker
```



Установка сервера метрики

Установка сервера метрики тоже выполняется не сложно:

```
kubectl apply -f
https://raw.githubusercontent.com/techiescamp/kubeadm-
scripts/main/manifests/metrics-server.yaml
```



После завершения развертывания ресурсов для сервера метрики мы сможет отображать статистику по всем подам кластера:

```
kubectl top pod -n kube-system
```

```
roman@master01:~$ kubectl top pod -n kube-system
NAME                                      CPU(cores)
MEMORY(bytes)
calico-kube-controllers-57b57c56f-tz9td   2m           15Mi
calico-node-62g6j                         26m          80Mi
calico-node-67dc4                         26m          80Mi
calico-node-8zwbh                         27m          81Mi
coredns-787d4945fb-g5fmh                  2m           12Mi
coredns-787d4945fb-wvvjb                  2m           12Mi
etcd-master01                             25m          34Mi
kube-apiserver-master01                   42m          331Mi
kube-controller-manager-master01          17m          52Mi
kube-proxy-55lq6                          6m           10Mi
kube-proxy-k9d7x                          6m           11Mi
kube-proxy-pjwjf                          5m           11Mi
kube-scheduler-master01                   3m           17Mi
metrics-server-7b67f8d5d6-744k7           6m           13Mi
roman@master01:~$
```

```
roman@master01:~$ kubectl top pod -n kube-system
NAME                                      CPU(cores)   MEMORY(bytes)
calico-kube-controllers-57b57c56f-tz9td   2m           15Mi
calico-node-62g6j                         26m          80Mi
calico-node-67dc4                         26m          80Mi
calico-node-8zwbh                         27m          81Mi
coredns-787d4945fb-g5fmh                  2m           12Mi
coredns-787d4945fb-wvvjb                  2m           12Mi
etcd-master01                             25m          34Mi
kube-apiserver-master01                   42m          331Mi
kube-controller-manager-master01          17m          52Mi
kube-proxy-55lq6                          6m           10Mi
kube-proxy-k9d7x                          6m           11Mi
kube-proxy-pjwjf                          5m           11Mi
kube-scheduler-master01                   3m           17Mi
metrics-server-7b67f8d5d6-744k7           6m           13Mi
roman@master01:~$
```

Добавление ingress контроллера

Ingress контроллер выступает в качестве реверс прокси и балансировщика сетевой нагрузки. В тоже время он позволяет внешним сервисам за пределами кластера Kubernetes получать доступ к вашим сервисам внутри кластера. Поскольку в моем примере развертывание Kubernetes кластера осуществляет, скажем так, на голом железе (bare metal), то для ingress контроллера нужно использовать NodePort. Я буду использовать NGINX Ingress Controller для bare metal сценария.

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-
v1.5.1/deploy/static/provider/baremetal/deploy.yaml
```

Если развертывание завершилось успешно, то вы должны увидеть три пода – два в статусе Completed и один в состоянии Running:

```
kubectl get pods -n ingress-nginx
```

```
roman@master01:~$ kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS
RESTARTS   AGE
ingress-nginx-admission-create-2qr9l        0/1     Completed   0
2m57s
ingress-nginx-admission-patch-pqqqb         0/1     Completed   0
```

```
2m57s
ingress-nginx-controller-64f79ddbcc-kr5bv   1/1     Running     0
2m57s
roman@master01:~$
```

```
roman@master01:~$ kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-2qr9l        0/1     Completed   0          2m57s
ingress-nginx-admission-patch-pqqqb         0/1     Completed   0          2m57s
ingress-nginx-controller-64f79ddbcc-kr5bv   1/1     Running     0          2m57s
roman@master01:~$
```

Также у нас должны появиться два сервиса:

```
kubectl get services -n ingress-nginx
```

```
roman@master01:~$ kubectl get services -n ingress-nginx
NAME                                 TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)                      AGE
ingress-nginx-controller             NodePort    10.98.233.3
<none>        80:30053/TCP,443:30621/TCP   6m1s
ingress-nginx-controller-admission   ClusterIP   10.104.189.166
<none>        443/TCP                      6m1s
roman@master01:~$
```

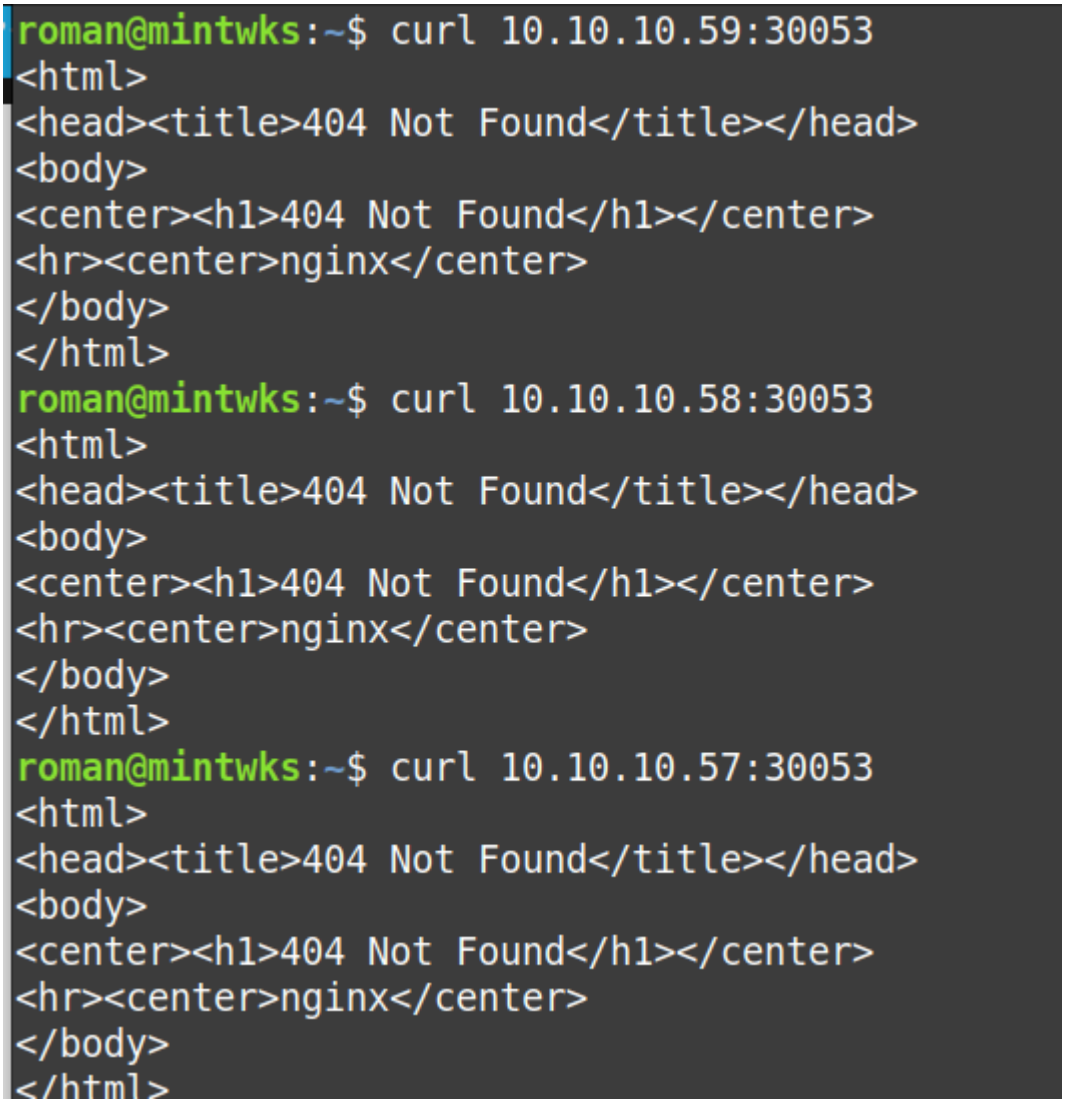
```
roman@master01:~$ kubectl get services -n ingress-nginx
NAME                                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
ingress-nginx-controller             NodePort    10.98.233.3     <none>        80:30053/TCP,443:30621/TCP   6m1s
ingress-nginx-controller-admission   ClusterIP   10.104.189.166  <none>        443/TCP                      6m1s
roman@master01:~$
```

Именно сервис с типом NodePort обеспечивает доступ до ресурсов кластера. Обратите внимание на выделенные порты – именно по ним мы будет обращаться к ресурсам кластера (в вашем случа порты будут отличаться) – они наша входная точка. Для доступа к порту 80 нужно будет отправлять запросы на порт 30053. Для доступа к порту 443 нужно будет обращаться на порт 30621. Причем запрос может обработать любой из узлов кластера.

Например, попробуем получить доступ до HTTP сервиса внутри кластера Kubernetes. Все эти запросы будут равнозначны:

```
curl http://10.10.10.59:30053
curl http://10.10.10.58:30053
```

```
curl http://10.10.10.57:30053
```



Пример доступа по HTTPS:

```
curl -k https://10.10.10.59:30621
curl -k https://10.10.10.58:30621
curl -k https://10.10.10.57:30621
```

```
roman@mintwks:~$ curl -k https://10.10.10.57:30621
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
roman@mintwks:~$ curl -k https://10.10.10.58:30621
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
roman@mintwks:~$ curl -k https://10.10.10.59:30621
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
roman@mintwks:~$
```

Соответственно, вы можете установить, например, nginx перед кластером Kubernetes и примерно вот такой конфигурацией балансировать запросы к кластеру. Пример конфигурации nginx для публикации HTTP сервиса нашего кластера Kubernetes:

```
upstream kubernetes {
    server 10.10.10.59:30053;
    server 10.10.10.58:30053;
    server 10.10.10.57:30053;
}

server {
    listen 80;
    server_name web.itproblog.ru;

    location / {
        proxy_pass http://kubernetes;
```

```
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Пример развертывания

В заключении я хотел бы показать как можно выполнить тестовое развертывание простого сервиса, доступ до которого можно будет получить по HTTP.

Пример для развертывания:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-appv1
spec:
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web-appv1
        image: gcr.io/google-samples/hello-app:1.0
---
apiVersion: v1
kind: Service
metadata:
  name: web-service-appv1
spec:
  selector:
```

```
    app: web
  ports:
    - protocol: TCP
      port: 50001
      targetPort: 8080
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: web-ingress-appv1
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/force-ssl-redirect: "false"
spec:
  ingressClassName: nginx
  rules:
  - host: web.itproblog.ru
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: web-service-appv1
            port:
              number: 50001
```

Сохраним конфигурацию выше в файл svc.yaml.

Попробуем развернуть этот сервис:

```
kubectl apply -f svc.yaml
```

```
roman@master01:~$ kubectl apply -f svc.yaml
deployment.apps/web-appv1 created
service/web-service-appv1 created
ingress.networking.k8s.io/web-ingress-appv1 created
roman@master01:~$
```

```
roman@master01:~$ kubectl apply -f svc.yaml
deployment.apps/web-appv1 created
service/web-service-appv1 created
ingress.networking.k8s.io/web-ingress-appv1 created
roman@master01:~$
```

Проверим поды:

```
kubectl get pod
```

```
roman@master01:~$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-appv1-579c48ddb6-6dp6f    1/1     Running   0          112s
roman@master01:~$
```


```
roman@master01:~$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-appv1-579c48ddb6-6dp6f    1/1     Running   0          112s
roman@master01:~$
```


Убедимся, что сервис был создан успешно:

```
kubectl get svc
```

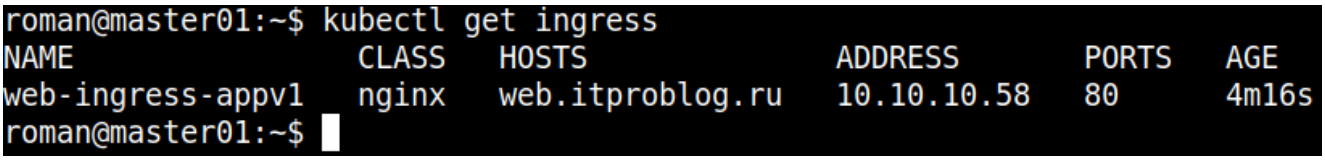
```
roman@master01:~$ kubectl get svc
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)     AGE
kubernetes          ClusterIP   10.96.0.1        <none>
443/TCP     58m
web-service-appv1   ClusterIP   10.109.176.137   <none>
50001/TCP   2m59s
roman@master01:~$
```


```
roman@master01:~$ kubectl get svc
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP     58m
web-service-appv1   ClusterIP   10.109.176.137   <none>        50001/TCP   2m59s
roman@master01:~$
```


Также проверим все ли хорошо с нашим ingress ресурсом:

```
kubectl get ingress
```

```
roman@master01:~$ kubectl get ingress
NAME                CLASS   HOSTS               ADDRESS        PORTS
AGE
web-ingress-appv1   nginx   web.itproblog.ru    10.10.10.58    80
4m16s
roman@master01:~$
```

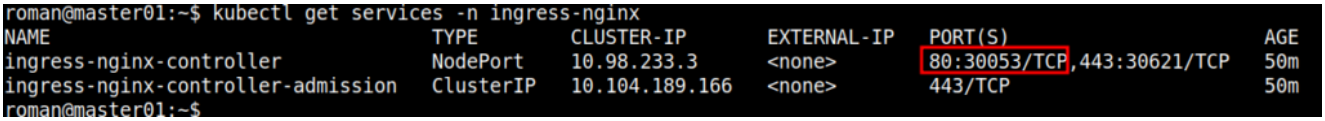


Теперь проверим доступ к сервису. Поскольку наш сервис прослушивает имя web.itproblog.ru, то я добавлю это имя в файл hosts и укажу ip адреса узлов кластера Kubernetes:

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

```
10.10.10.59 web.itproblog.ru
10.10.10.58 web.itproblog.ru
10.10.10.57 web.itproblog.ru
```

Последний шаг – проверка доступности сервиса Kubernetes кластера через ingress контроллер. Напомню, что наша входная точка для HTTP протокола – порт 30053:




```
curl http://web.itproblog.ru:30053
```

```
roman@mintwks:~$ curl http://web.itproblog.ru:30053
Hello, world!
Version: 1.0.0
Hostname: web-appv1-579c48ddb6-6dp6f
roman@mintwks:~$
```

```
roman@mintwks:~$ curl http://web.itproblog.ru:30053
Hello, world!
Version: 1.0.0
Hostname: web-appv1-579c48ddb6-6dp6f
roman@mintwks:~$
```

Причем не важно к какому из трех узлов кластера будет отправлен запрос – мы получим ответ от нашего сервиса в любом случае. Очень похоже на то, что наш кластер Kubernetes и все его компоненты работают корректно.

Это был последний шаг. На этом настройка кластера Kubernetes завершена.

DEVOPS KUBERNETES

‹ Мониторинг Proxmox через Zabbix

Ошибка при получении свойства сертификата (0x00000000) ›

Добавить комментарий

Ваш адрес email не будет опубликован. Обязательные поля помечены *

Комментарий *

Имя *

Email *

Сайт

Сохранить моё имя, email и адрес сайта в этом браузере для последующих моих комментариев.

ОТПРАВИТЬ КОММЕНТАРИЙ

Инфоплекс

Решение IT-задач любой сложности

Поиск…

Архивы

Октябрь 2024

Сентябрь 2024

Июль 2024

Май 2024

Апрель 2024

Март 2024

Февраль 2024

Январь 2024

Декабрь 2023

Ноябрь 2023

Сентябрь 2023

Август 2023

Июль 2023

Июнь 2023

Май 2023

Апрель 2023

Март 2023

Февраль 2023

Январь 2023

Декабрь 2022

Ноябрь 2022

Октябрь 2022

Август 2022

Июль 2022

Июнь 2022

Май 2022

Апрель 2022

Март 2022

Февраль 2022

Январь 2022

Декабрь 2021

Ноябрь 2021

Октябрь 2021

Сентябрь 2021

Август 2021

Май 2021

Апрель 2021

Март 2021

Февраль 2021

Январь 2021

Рубрики

1C (4)

Active Directory Domain Services (4)

Active Directory Federation Services (2)

Ansible (7)

Apache (1)

Astra Linux (3)

Astra Linux Directory (1)

Azure (5)

Cireson (2)

Communigate Pro (20)

Docker (1)

ELK (16)

EVE-NG (2)

Exchange (27)

GitLab (2)

JIRA (1)

Keycloack (1)

Kubernetes (6)

Linux (40)

MySQL (8)

NextCloud (6)

PostgreSQL (10)

Power Automate (4)

Project Server (2)

Project Web App (2)

Proxmox (12)

Scripts (1)

Sendria (1)

SharePoint (4)

System Center (25)

Terraform (1)

Veeam (3)

VirtualHere (3)

VMware (2)

Web Application Proxy (2)

Zabbix (14)

Балансировка сетевого траффика (2)

Без рубрики (1)

Онлайн кассы (2)

Печать (1)

Прочее (6)

Публикация сервисов и приложений (2)

Сертификаты (5)

сети (1)

Система управления проектами (2)

Системы хранения данных (2)

Торговое оборудование (1)

Облако тегов

1C AD DS Ansible Apache Azure Azure AD Password Protection Backup Cireson Communigate Pro DevOps Elasticsearch Exchange 2010 Exchange 2010 to 2019 migration Exchange 2016 Exchange 2019 Filebeat FreeIPA GitLab Kibana Kubernetes Linux Logstash Metricabeat Minikube MySQL MySQL Performance NextCloud Nginx OpenSSL Orchestrator PostgreSQL Power Automate Proxmox SCOM SCSM SharePoint Winlogbeat Zabbix АТОЛ Альт Сервер Виртуализация ККТ Миграция с Exchange на CommuniGate Онлайн-кассы Шпаргалки

Мета

Войти

Лента записей

Лента комментариев

WordPress.org

© 2024 IT Pro Blog. На платформе Sydney