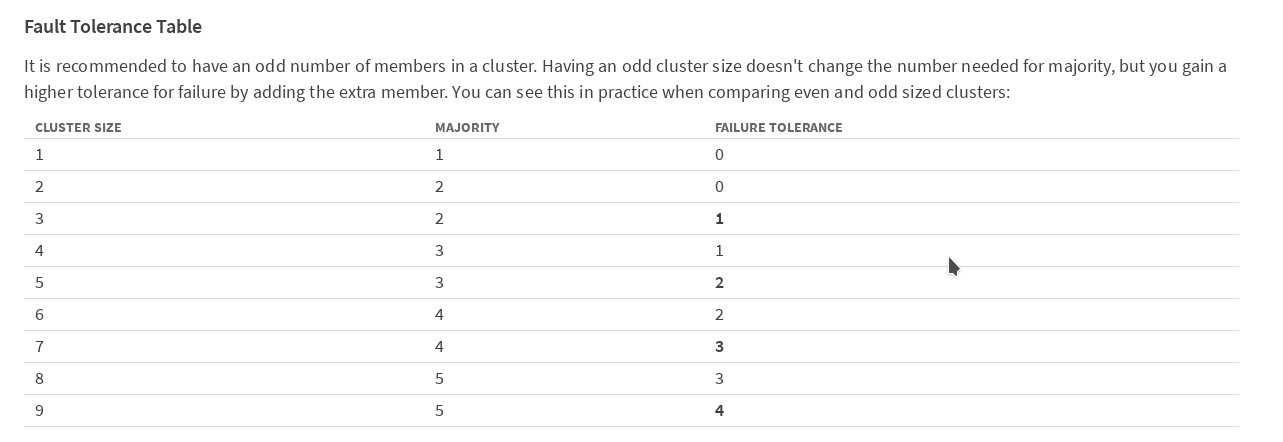
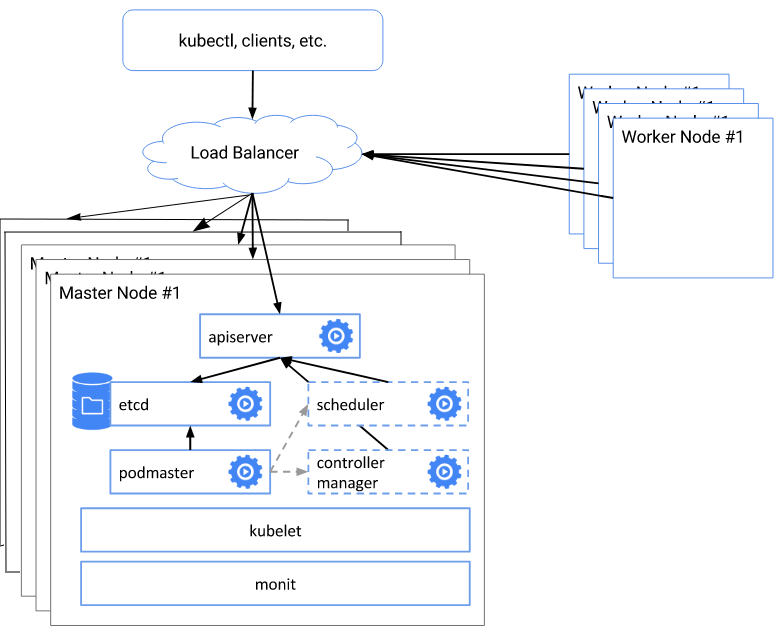
Кластер K8s c 3 мастерами

1. [Список используемого софта](https://habr.com/post/358264/#po)
2. [Список и назначение хостов](https://habr.com/post/358264/#hostslist)
3. [Принцип работы и развертывания](https://habr.com/post/358264/#deployshema)
4. [Подготовка ОС к развертыванию. Установка docker, kubeadm, kubelet и kubectl](https://habr.com/post/358264/#beforebegin)
5. [Подготовка конфигурационного скрипта](https://habr.com/post/358264/#startscript)
6. [Создание etcd кластера](https://habr.com/post/358264/#etcd)
7. [Инициализация мастера с помощью kubeadm](https://habr.com/post/358264/#firstmaster)
8. [Настройка CIDR](https://habr.com/post/358264/#cidr)
9. [Инициализация остальных мастернод](https://habr.com/post/358264/#othermaster)
10. [Настройка keepalived и виртуального IP](https://habr.com/post/358264/#balancer)
11. [Добавление рабочих нод в кластер](https://habr.com/post/358264/#worknode)
12. [Установка ingress-nginx](https://habr.com/post/358264/#ingress)
13. Дополнительно
    * [Dashboard](https://habr.com/post/358264/#dashboard)
    * [Heapster](https://habr.com/post/358264/#heapster)

Задача - поместить проект в отказоустойчивый кластер при помощи kubernetes.   
  
Для повышенной отказоустойчивости кластер было решено разворачивать с тремя мастер нодами.  
  
Согласно таблице в документации etcd на [сайте CoreOS](https://coreos.com/etcd/docs/latest/v2/admin_guide.html),   
  
  
  
рекомендуется иметь нечетное число членов в кластере. Для того, чтобы кластер продолжал работать после выхода из строя одного члена (в нашем случае мастера kubernetes), нужно минимум 3 машины.

Краткая схема и описание архитектуры развертывания

  
  
Суть схемы заключается в следующем:

* создаем etcd кластер
* при помощи kubeadm init создаем первого мастера сертификаты, ключи и.т.д.
* с помощью сгенерированых файлов конфигурации инициализируем остальные 2 мастер ноды
* конфигурируем балансировщик nginx на каждой мастер ноде для виртуального адреса
* меняем адрес и порт API сервера на выделенный виртуальный адрес
* Добавляем в кластер рабочие ноды

**Список используемого софта**

* ОС - CentOS 7.5-1804
* контейнеры:  
  docker version 18.06.1-ce, build e68fc7a, docker-compose version 1.22.0
* Kubernetes v1.10.3
* networks add-ons: flannel
* Балансировщик: nginx
* Виртуальный IP: keepalived Version: v1.3.5-6-g6fa32f2

Список хостов

|  |  |  |  |
| --- | --- | --- | --- |
| **IP адрес** | **Имена хостов** | **Описание** | **Компоненты** |
| 10.90.2.41 | hb-master01 | master nodes \* 3 | keepalived, nginx, etcd, kubelet, kube-apiserver, kube-scheduler, kube-proxy, kube-dashboard, heapster |
| 10.90.2.51 | hb-master02 |
| 10.90.2.61 | hb-master03 |
| 10.90.2.42 | hb-node01 | worker nodes \* 3 | kubelet, kube-proxy |
| 10.90.2.52 | hb-node02 |
| 10.90.2.62 | hb-node03 |
| 10.90.2.111 | N\A | keepalived virtual IP | N\A |

Подготовка ОС к развертыванию. Установка docker, kubeadm, kubelet и kubectl, pre & post install steps

Для подготовки мастер хоста запускаем скрипт master-hosts-prepare.sh

ssh root@hb-master01 "bash -s" < ./master-hosts-prepare.sh > mhp01.log

ssh root@hb-master02 "bash -s" < ./master-hosts-prepare.sh > mhp02.log

ssh root@hb-master03 "bash -s" < ./master-hosts-prepare.sh > mhp03.log

Для подготовки воркер хоста запускаем скрипт worker-hosts-prepare.sh

ssh root@hb-node01 "bash -s" < ./worker-hosts-prepare.sh > whp01.log

ssh root@hb-node02 "bash -s" < ./worker-hosts-prepare.sh > whp02.log

ssh root@hb-node03 "bash -s" < ./worker-hosts-prepare.sh > whp03.log

Подготовка конфигурационного скрипта

# настройки на локальной машине каждой из нод (на каждой ноде свои)  
**K8SHA\_IPLOCAL** — IP адрес ноды на которой настраивается скрипт  
**K8SHA\_ETCDNAME** — имя локальной машины в кластере ETCD, соответственно на master01 — etcd1, master02 — etcd2 и т.д.  
**K8SHA\_KA\_STATE** — роль в keepalived. Одна нода MASTER, все остальные BACKUP.   
**K8SHA\_KA\_PRIO** — приоритет keepalived, у мастера 102 у остальных 101, 100. При падении мастера с номером 102, его место занимает нода с номером 101 и так далее.  
**K8SHA\_KA\_INTF** — keepalived network interface. Имя интерфейса который будет слушать keepalived  
  
# общие настройки для всех мастернод одинаковые  
**K8SHA\_IPVIRTUAL**= 10.90.2.111 — виртуальный IP кластера.   
**K8SHA\_IP1...K8SHA\_IP3 — IP** адреса мастеров  
**K8SHA\_HOSTNAME1 ...K8SHA\_HOSTNAME3**— имена хостов для мастернод. Важный пункт, по этим именам kubeadm будет генерировать сертификаты.   
**K8SHA\_KA\_AUTH** — пароль для keepalived. Можно задать произвольный  
**K8SHA\_TOKEN** — токен кластера. Можно сгенерировать командой **kubeadm token generate**  
**K8SHA\_CIDR** — адрес подсети для подов. Используем flannel поэтому CIDR 0.244.0.0/16. Обязательно экранировать — в конфиге должно быть K8SHA\_CIDR=10.244.0.0\\/16

На каждой мастер ноде запускаем соответствующий скрипт create-conf-ma?.sh  
  
create-conf-ma1.sh

create-conf-ma2.sh

create-conf-ma3.sh

Создание etcd кластера

На основании полученных конфигов создаем etcd кластер

docker-compose --file etcd/docker-compose.yaml up -d

После того, как на всех мастерах поднялись контейнеры, проверяем статус etcd

docker exec -ti etcd etcdctl cluster-health

member 3357c0f051a52e4a is healthy: got healthy result from http://172.26.133.24:2379

member 4f9d89f3d0f7047f is healthy: got healthy result from http://172.26.133.21:2379

member 8870062c9957931b is healthy: got healthy result from http://172.26.133.23:2379

member c8923ecd7d317ed4 is healthy: got healthy result from http://172.26.133.22:2379

member cd879d96247aef7e is healthy: got healthy result from http://172.26.133.25:2379

cluster is healthy

docker exec -ti etcd etcdctl member list

3357c0f051a52e4a: name=etcd4 peerURLs=http://172.26.133.24:2380 clientURLs=http://172.26.133.24:2379,http://172.26.133.24:4001 isLeader=false

4f9d89f3d0f7047f: name=etcd1 peerURLs=http://172.26.133.21:2380 clientURLs=http://172.26.133.21:2379,http://172.26.133.21:4001 isLeader=false

8870062c9957931b: name=etcd3 peerURLs=http://172.26.133.23:2380 clientURLs=http://172.26.133.23:2379,http://172.26.133.23:4001 isLeader=false

c8923ecd7d317ed4: name=etcd2 peerURLs=http://172.26.133.22:2380 clientURLs=http://172.26.133.22:2379,http://172.26.133.22:4001 isLeader=true

cd879d96247aef7e: name=etcd5 peerURLs=http://172.26.133.25:2380 clientURLs=http://172.26.133.25:2379,http://172.26.133.25:4001 isLeader=false

Если с кластером, все в порядке, то двигаемся дальше. Если что -то не так, то смотрим логи

docker logs etcd

Инициализация первой мастер ноды с помощью kubeadm

На hb-master01 используя kubeadm выполняем инициализацию кластера kubernetes.

kubeadm init --config=kubeadm-init.yaml

Если будет ошибка по версии Kubelet то к строке нужно добавить ключ

--ignore-preflight-errors=KubeletVersion

После того как мастер инициализируется, kubeadm выведет на экран служебную информацию. В ней будет указан token и хэш для инициализации других членов кластера. Обязательно сохраните строчку вида: **kubeadm join --token XXXXXXXXXXXX 172.26.133.21:6443 --discovery-token-ca-cert-hash sha256:XXXXXXXXXXXXXXXXXXXXXXX**, где нибудь отдельно, так как данная информация выводится один раз; если токены будут утеряны, их придется генерировать заново. 

Your Kubernetes master 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

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 machines by running the following on each node

as root:

kubeadm join --token XXXXXXXXXXXX 172.26.133.21:6443 --discovery-token-ca-cert-hash sha256:XXXXXXXXXXXXXXXXXXXXXXX

Далее нужно установить переменную окружения, для возможности работать с кластером от root

vim ~/.bashrc

export KUBECONFIG=/etc/kubernetes/admin.conf

source ~/.bashrc

Если нужно работать под обычным пользователем, то следуем инструкции, которая появилась на экране при инициализации мастера. 

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

Проверяем, что все сделали правильно:

kubectl get node

NAME STATUS ROLES AGE VERSION

hb-master01 NotReady master 22m v1.9.5

Мастер будет находится в статусе NotReady до того как мы не поднимем сеть cidr, это нормально.

Настройка CIDR

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/v0.9.1/Documentation/kube-flannel.yml

clusterrole "flannel" created

clusterrolebinding "flannel" created

serviceaccount "flannel" created

configmap "kube-flannel-cfg" created

daemonset "kube-flannel-ds" created

Проверяем, что все ОК

kubectl get pods --all-namespaces

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-system kube-apiserver-hb-master01 1/1 Running 0 1h

kube-system kube-controller-manager-hb-master01 1/1 Running 0 1h

kube-system kube-dns-6f4fd4bdf-jdhdk 3/3 Running 0 1h

kube-system kube-flannel-ds-hczw4 1/1 Running 0 1m

kube-system kube-proxy-f88rm 1/1 Running 0 1h

kube-system kube-scheduler-hb-master01 1/1 Running 0 1h

Инициализация остальных мастернод

Теперь после того, как наш кластер работает с одной нодой, настало время ввести в кластер оставшиеся мастерноды.   
  
Для этого с hb-master01 нужно скопировать каталог /etc/kubernetes/pki в удаленный каталог /etc/kubernetes/ каждого мастера. Для копирования в настройках ssh я временно разрешил подключение руту. После копирования файлов, естественно, данную возможность отключил.   
  
На каждой из оставшихся мастернод настраиваем ssh сервер

vim /etc/ssh/sshd\_config

PermitRootLogin yes

systemctl restart ssh

Копируем файлы

scp -r /etc/kubernetes/pki 172.26.133.22:/etc/kubernetes/ \

&& scp -r /etc/kubernetes/pki 172.26.133.23:/etc/kubernetes/ \

&& scp -r /etc/kubernetes/pki 172.26.133.24:/etc/kubernetes/ \

&& scp -r /etc/kubernetes/pki 172.26.133.25:/etc/kubernetes/

Теперь на hb-master02 используйте kubeadm для запуска кластера, убедитесь, что pod kube-apiserver- находится в рабочем состоянии. 

kubeadm init --config=kubeadm-init.yaml

Your Kubernetes master 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

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 machines by running the following on each node

as root:

kubeadm join --token xxxxxxxxxxxxxx 172.26.133.22:6443 --discovery-token-ca-cert-hash sha256:xxxxxxxxxxxxxxxxxxxxxxxxx

Повторяем на hb-master03, hb-master04, hb-master05  
  
Проверяем, что все мастера инициализировались и работают в кластере

kubectl get nodes

NAME STATUS ROLES AGE VERSION

hb-master01 Ready master 37m v1.9.5

hb-master02 Ready master 33s v1.9.5

hb-master03 Ready master 3m v1.9.5

hb-master04 Ready master 17m v1.9.5

hb-master05 Ready master 19m v1.9.5

kubectl get pods --all-namespaces

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-system kube-apiserver-hb-master01 1/1 Running 0 6m

kube-system kube-apiserver-hb-master02 1/1 Running 0 1m

kube-system kube-apiserver-hb-master03 1/1 Running 0 1m

kube-system kube-apiserver-hb-master04 1/1 Running 0 1m

kube-system kube-apiserver-hb-master05 1/1 Running 0 10s

kube-system kube-controller-manager-hb-master01 1/1 Running 0 6m

kube-system kube-controller-manager-hb-master02 1/1 Running 0 1m

kube-system kube-controller-manager-hb-master03 1/1 Running 0 1m

kube-system kube-controller-manager-hb-master04 1/1 Running 0 1m

kube-system kube-controller-manager-hb-master05 1/1 Running 0 9s

kube-system kube-dns-6f4fd4bdf-bnxl8 3/3 Running 0 7m

kube-system kube-flannel-ds-j698p 1/1 Running 0 6m

kube-system kube-flannel-ds-mf9zc 1/1 Running 0 2m

kube-system kube-flannel-ds-n5vbm 1/1 Running 0 2m

kube-system kube-flannel-ds-q7ztg 1/1 Running 0 1m

kube-system kube-flannel-ds-rrrcq 1/1 Running 0 2m

kube-system kube-proxy-796zl 1/1 Running 0 1m

kube-system kube-proxy-dz25s 1/1 Running 0 7m

kube-system kube-proxy-hmrw5 1/1 Running 0 2m

kube-system kube-proxy-kfjst 1/1 Running 0 2m

kube-system kube-proxy-tpkbt 1/1 Running 0 2m

kube-system kube-scheduler-hb-master01 1/1 Running 0 6m

kube-system kube-scheduler-hb-master02 1/1 Running 0 1m

kube-system kube-scheduler-hb-master03 1/1 Running 0 1m

kube-system kube-scheduler-hb-master04 1/1 Running 0 48s

kube-system kube-scheduler-hb-master05 1/1 Running 0 29s

Создадим реплики службы kube-dns. На hb-master01 выполнить

kubectl scale --replicas=5 -n kube-system deployment/kube-dns

На всех мастернодах в файл конфигурации внести строчку с количеством api серверов   
Если вы используете kubernetes версии больше 1.9 этот шаг можно пропустить. 

vim /etc/kubernetes/manifests/kube-apiserver.yaml

- --apiserver-count=5

systemctl restart docker && systemctl restart kubelet

Настройка keepalived и виртуального IP

На всех мастернодах настраиваем keepalived и nginx в качестве балансировщика 

systemctl restart keepalived

docker-compose -f nginx-lb/docker-compose.yaml up -d

Тестируем работу

curl -k https://172.26.133.21:16443 | wc -1

wc: invalid option -- '1'

Try 'wc --help' for more information.

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 233 100 233 0 0 15281 0 --:--:-- --:--:-- --:--:-- 15533

Если 100 % — то все ОК.   
  
После того, как мы получили работающий виртуальный адрес, укажем его как адрес API сервера.  
  
На hb-master01 

kubectl edit -n kube-system configmap/kube-proxy

server: https://172.26.133.20:16443

Удалим все kube-proxy pod для их рестарта с новыми параметрами.

kubectl get pods --all-namespaces -o wide | grep proxy

kubectl delete pod -n kube-system kube-proxy-XXX

Проверим, что все рестартанули

kubectl get pods --all-namespaces -o wide | grep proxy

kube-system kube-proxy-2q7pz 1/1 Running 0 28s 172.26.133.22 hb-master02

kube-system kube-proxy-76vnw 1/1 Running 0 10s 172.26.133.23 hb-master03

kube-system kube-proxy-nq47m 1/1 Running 0 19s 172.26.133.24 hb-master04

kube-system kube-proxy-pqqdh 1/1 Running 0 35s 172.26.133.21 hb-master01

kube-system kube-proxy-vldg8 1/1 Running 0 32s 172.26.133.25 hb-master05

Добавление рабочих нод в кластер

На каждой рабочей ноде устанавливаем docke, kubernetes и kubeadm, по аналогии с мастерами.  
Добавляем ноду в кластер, используя токены сгенерированные при инициализации hb-master01

kubeadm join --token xxxxxxxxxxxxxxx 172.26.133.21:6443 --discovery-token-ca-cert-hash sha256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

[preflight] Running pre-flight checks.

[WARNING FileExisting-crictl]: crictl not found in system path

[discovery] Trying to connect to API Server "172.26.133.21:6443"

[discovery] Created cluster-info discovery client, requesting info from "https://172.26.133.21:6443"

[discovery] Requesting info from "https://172.26.133.21:6443" again to validate TLS against the pinned public key

[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will use API Server "172.26.133.21:6443"

[discovery] Successfully established connection with API Server "172.26.133.21:6443"

This node has joined the cluster:

\* Certificate signing request was sent to master and a response

was received.

\* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.

Проверяем, что все рабочие ноды вошли в кластер и они доступны. 

kubectl get nodes

NAME STATUS ROLES AGE VERSION

hb-master01 Ready master 20h v1.9.5

hb-master02 Ready master 20h v1.9.5

hb-master03 Ready master 20h v1.9.5

hb-master04 Ready master 20h v1.9.5

hb-master05 Ready master 20h v1.9.5

hb-node01 Ready <none> 12m v1.9.5

hb-node02 Ready <none> 4m v1.9.5

hb-node03 Ready <none> 31s v1.9.5

**Только на рабочих нодах** в файлах /etc/kubernetes/bootstrap-kubelet.conf и /etc/kubernetes/kubelet.conf меняем  
значение переменной server на наш виртуальный IP

vim /etc/kubernetes/bootstrap-kubelet.conf

server: https://172.26.133.20:16443

vim /etc/kubernetes/kubelet.conf

server: https://172.26.133.20:16443

systemctl restart docker kubelet

Далее Вы можете увеличивать производительность Вашего кластера добавляя новые рабочие ноды, по мере надобности.  
  
**Установка ingress-nginx**  
Ntgthm нам осталось установить ingress.  
В документации kubernetes про Ingress написано следующее:  
*Объект API, который управляет внешним доступом к службам в кластере, обычно HTTP.  
Ingress может обеспечивать балансировку нагрузки, завершение SSL и виртуальный хостинг на основе имен.*  
  
В общем то более подробно я вряд ли смогу описать. Настройка ingress это материал для отдельной статьи, в контексте установки кластера я лишь опишу его установку. 

kubectl apply -f kube-ingress/mandatory.yaml

namespace "ingress-nginx" created

deployment.extensions "default-http-backend" created

service "default-http-backend" created

configmap "nginx-configuration" created

configmap "tcp-services" created

configmap "udp-services" created

serviceaccount "nginx-ingress-serviceaccount" created

clusterrole.rbac.authorization.k8s.io "nginx-ingress-clusterrole" configured

role.rbac.authorization.k8s.io "nginx-ingress-role" created

rolebinding.rbac.authorization.k8s.io "nginx-ingress-role-nisa-binding" created

clusterrolebinding.rbac.authorization.k8s.io "nginx-ingress-clusterrole-nisa-binding" configured

deployment.extensions "nginx-ingress-controller" created

kubectl apply -f kube-ingress/service-nodeport.yaml

service "ingress-nginx" created

Проверяем что ingress поднялся:

kubectl get all -n ingress-nginx

NAME READY STATUS RESTARTS AGE

pod/default-http-backend-5c6d95c48-j8sd4 1/1 Running 0 5m

pod/nginx-ingress-controller-58c9df5856-vqwst 1/1 Running 0 5m

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/default-http-backend ClusterIP 10.109.216.21 <none> 80/TCP 5m

service/ingress-nginx NodePort 10.96.229.115 172.26.133.20 80:32700/TCP,443:31211/TCP 4m

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

deployment.apps/default-http-backend 1 1 1 1 5m

deployment.apps/nginx-ingress-controller 1 1 1 1 5m

NAME DESIRED CURRENT READY AGE

replicaset.apps/default-http-backend-5c6d95c48 1 1 1 5m

replicaset.apps/nginx-ingress-controller-58c9df5856 1 1 1 5m

На этом шаге настройка кластера закончена. Если вы все сделали правильно, то должны получить отказоустойчивый, рабочий кластер Kubernetes c отказоустойчивой точкой входа и балансировщиком на виртуальном адресе.   
  
Спасибо за внимание, буду рад комментариям, или указаниям на неточности. Также можно создавать issue на github, я постараюсь оперативно реагировать на них.   
  
С уважением,  
Евгений Родионов

Дополнительно

|  
**Установка панели управления Kubernetes Dashboard**  
У Kubernetes кроме cli, имеется не плохая панель инструментов. Устанавливается она очень просто, инструкция и документация есть на [GitHub](https://github.com/kubernetes/dashboard)  
  
Команды можно выполнять на любом из 5 мастеров. Я работаю с hb-master01

kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml

Проверяем:

kubectl get pods --all-namespaces -o wide | grep kubernetes-dashboard

kube-system kubernetes-dashboard-5bd6f767c7-cz55w 1/1 Running 0 1m 10.244.7.2 hb-node03

Панель теперь доступна по адресу:

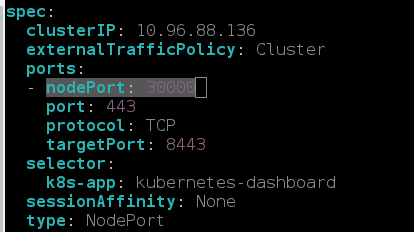
<http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/>

Но чтобы на нее попасть, нужно пробрасывать proxy c локальной машины с помощью команды

kubectl proxy

Мне это не удобно, поэтому я использую [NodePort](https://kubernetes.io/docs/concepts/services-networking/service/#type-nodeport)и размещу панель по адресу[https://172.26.133.20:30000](https://172.26.133.20:30000/)на первый доступный порт из диапазона, выделенного для NodePort. 

kubectl -n kube-system edit service kubernetes-dashboard

Заменяем значение type: ClusterIP на type: NodePort и в секцию port: добавляем значение nodePort: 30000  
  
  
  
Далее создадим пользователя с именем admin-user и полномочиями администратора кластера.

kubectl apply -f kube-dashboard/dashboard-adminUser.yaml

serviceaccount "admin-user" created

clusterrolebinding "admin-user" created

Получаем токен для пользователя admin-user

kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')

Name: admin-user-token-p8cxl

Namespace: kube-system

Labels: <none>

Annotations: kubernetes.io/service-account.name=admin-user

kubernetes.io/service-account.uid=0819c99c-2cf0-11e8-a281-a64625c137fc

Type: kubernetes.io/service-account-token

Data

====

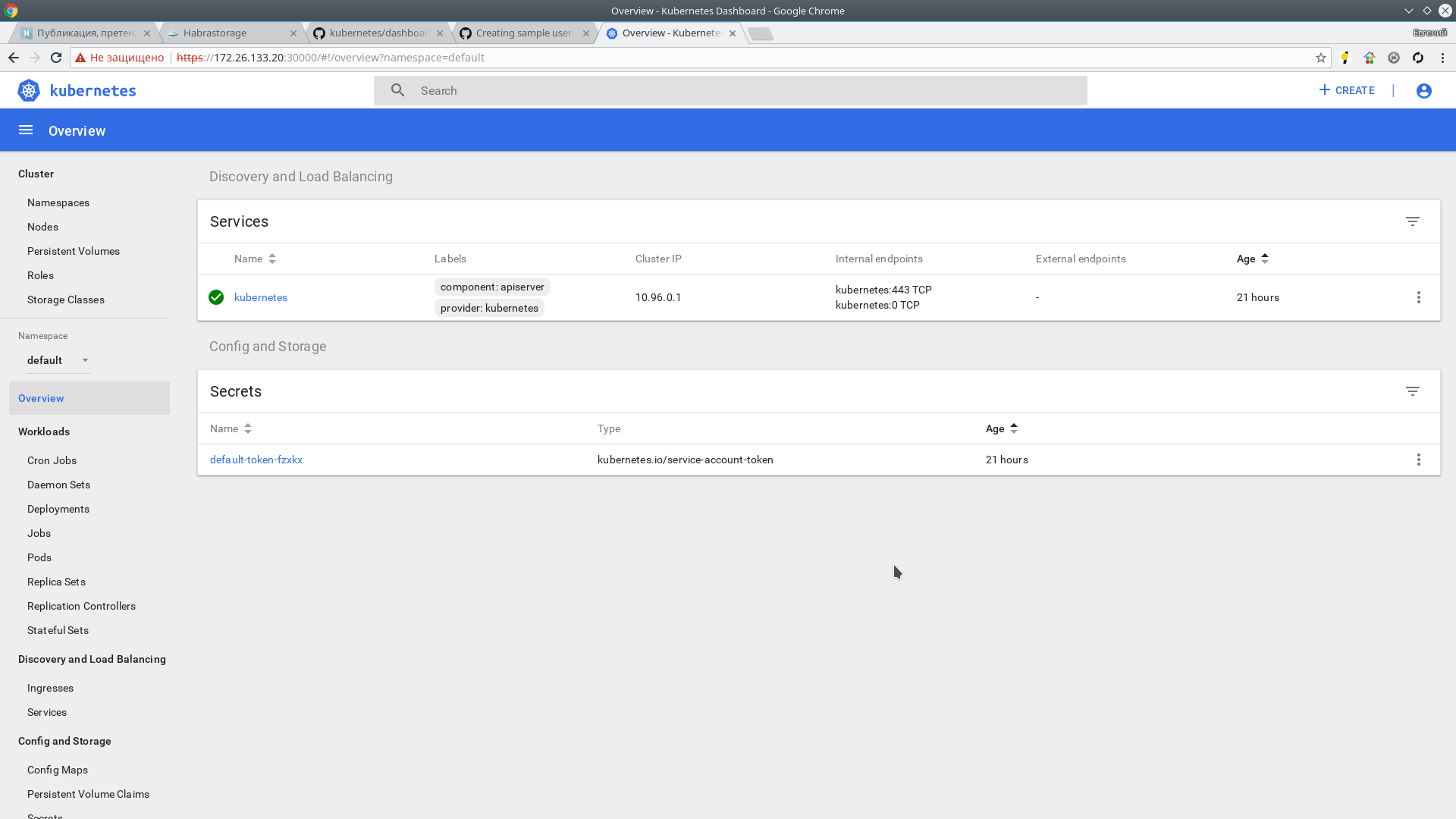
ca.crt: 1025 bytes

namespace: 11 bytes

token: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Копируем token и переходим по адресу:

<https://172.26.133.20:30000/>

Теперь нам доступна панель управления Kubernetes кластера c полномочиями админа.   
  


Heapster

Далее установим Heapster. Это инструмент для мониторинга ресурсов всех составляющих кластера.[Страничка проекта на GitHub](https://github.com/kubernetes/heapster)   
  
Установка:

git clone https://github.com/kubernetes/heapster.git

cd heapster

kubectl create -f deploy/kube-config/influxdb/

deployment "monitoring-grafana" created

service "monitoring-grafana" created

serviceaccount "heapster" created

deployment "heapster" created

service "heapster" created

deployment "monitoring-influxdb" created

service "monitoring-influxdb" created

kubectl create -f deploy/kube-config/rbac/heapster-rbac.yaml

clusterrolebinding "heapster" created

Через пару минут должна пойти информация. Проверяем:

kubectl top nodes

NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%

hb-master01 166m 4% 1216Mi 31%

hb-master02 135m 3% 1130Mi 29%

hb-master03 142m 3% 1091Mi 28%

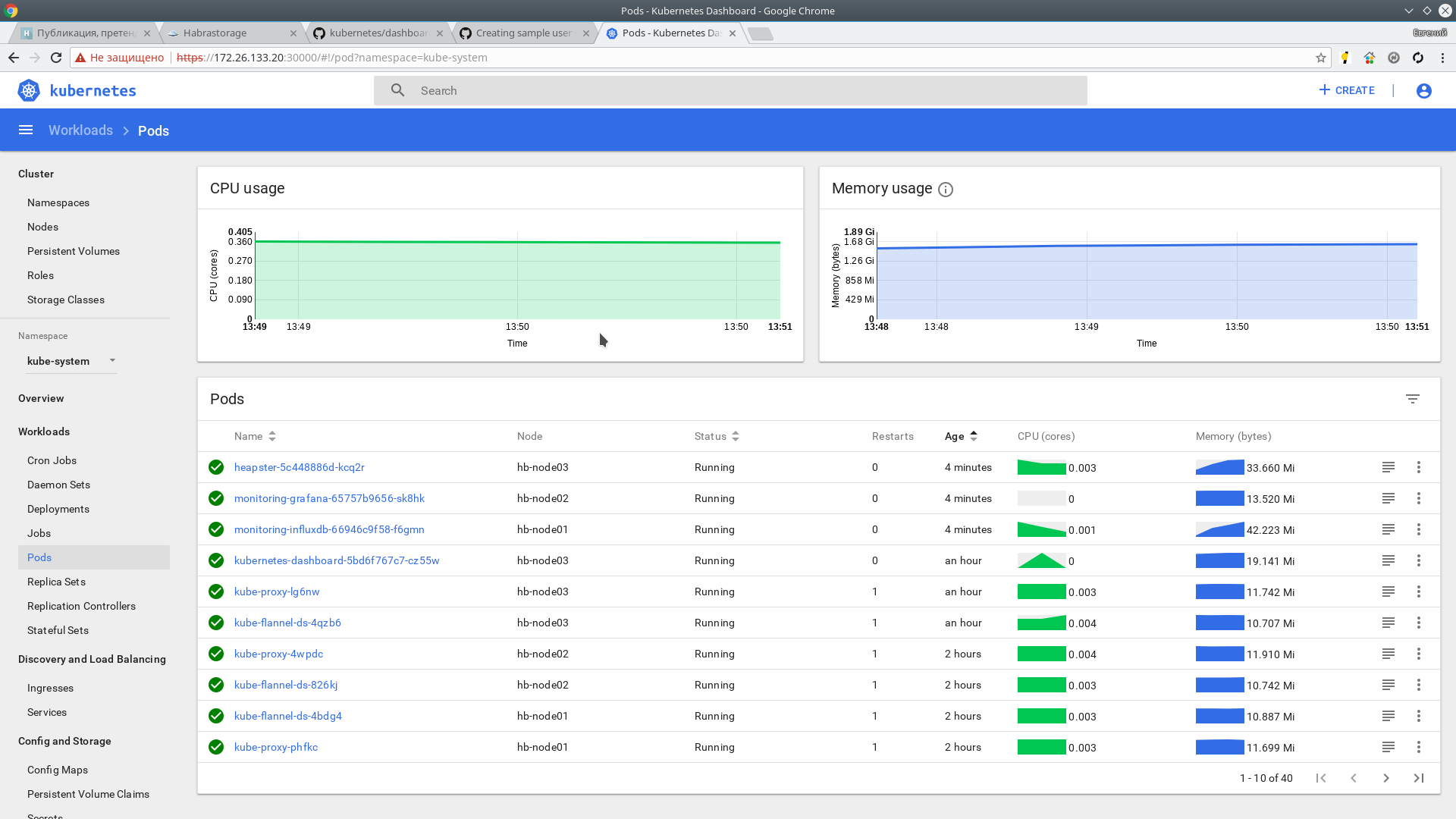
hb-master04 193m 4% 1149Mi 29%

hb-master05 144m 3% 1056Mi 27%

hb-node01 41m 1% 518Mi 3%

hb-node02 38m 0% 444Mi 2%

hb-node03 45m 1% 478Mi 2%

Также метрики доступны в web интерфейсе  
  
  
  
**Использование материалы:**

* [kubeadm-highavailiability — kubernetes high availiability deployment based on kubeadm, for Kubernetes version 1.9.x/1.7.x/1.6.x](https://github.com/cookeem/kubeadm-ha)
* [Kubernetes Documentation](https://kubernetes.io/docs/home/?path=users&persona=app-%20developer&level=foundational)
* [Основы Kubernetes](https://habrahabr.ru/post/258443/)
* [Accessing Kubernetes Pods from Outside of the Cluster](http://alesnosek.com/blog/2017/02/14/accessing-kubernetes-pods-from-outside-of-the-cluster/)

**Метки:**

* [k8s](https://habr.com/search/?q=%5Bk8s%5D&target_type=posts)
* [kubernetes](https://habr.com/search/?q=%5Bkubernetes%5D&target_type=posts)
* [HA](https://habr.com/search/?q=%5BHA%5D&target_type=posts)
* [кластер](https://habr.com/search/?q=%5B%D0%BA%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80%5D&target_type=posts)