

# Установка и настройка кластера Kubernetes на Ubuntu Server

🕒 Обновлено: 28.09.2022 🕒 Опубликовано: 07.02.2021

Используемые термины: [Kubernetes](#), [Ubuntu](#), [Docker](#).

Есть различные готовые реализации кластера Kubernetes (K8S), например:

- Minikube — готовый кластер, который разворачивается на один компьютер. Хороший способ познакомиться с Kubernetes.
- Kubespray — набор [Ansible](#) ролей.
- Готовые кластеры в облаке, например AWS, Google Cloud, Yandex Cloud и так далее.

Использовать одну из готовых реализаций — быстрый и надежный способ развертывания системы оркестрации контейнеров Docker. Однако, мы рассмотрим ручное создание кластера Kubernetes из 3-х нод — один мастер (управление) и две рабочие ноды (запуск контейнеров).

Для этого выполним следующие шаги:

[Готовим систему к развертыванию кластера](#)

[Устанавливаем Docker](#)

[Устанавливаем пакеты Kubernetes](#)

[Создаем кластер](#)

[Control-plane](#)

[Worker](#)

[Работа с подами](#)

[Работа с развертываниями](#)

[Работа с сервисами](#)

[Монтирование конфигураций](#)

[Немного об Ingress](#)

[Установка и настройка веб-интерфейса](#)

[Как удалить ноду из кластера](#)

## Подготовка системы

Данные действия выполняем на всех узлах будущего кластера. Это необходимо, чтобы удовлетворить программные

системные требования для нашего кластера.

## Настройка системы

1. Задаем имена узлам. Для этого выполняем команды на соответствующих серверах:

```
hostnamectl set-hostname k8s-  
master1.dmosk.local
```

```
hostnamectl set-hostname k8s-  
worker1.dmosk.local
```

```
hostnamectl set-hostname k8s-  
worker2.dmosk.local
```

*\* в данном примере мы зададим имя **k8s-master1** для мастера и, соответственно, **k8s-worker1** и **k8s-worker2** — для первого и второго рабочих нод. Каждая команда выполняется на своем сервере.*

Необходимо, чтобы наши серверы были доступны по заданным именам. Для этого необходимо на сервере DNS добавить соответствующие A-записи. Или на каждом сервере открываем hosts:

```
vi /etc/hosts
```

И добавляем записи на подобие:

```
192.168.0.15      k8s-master1.dmosk.local k8s-  
master1  
192.168.0.20      k8s-worker1.dmosk.local k8s-  
worker1  
192.168.0.25      k8s-worker2.dmosk.local k8s-  
worker2
```

*\* где, **192.168.0.15**, **192.168.0.20**, **192.168.0.25** — IP-адреса наших серверов, **k8s-master1**, **k8s-worker1**, **k8s-worker2** — имена серверов, **dmosk.local** — наш внутренний домен.*

2. Устанавливаем необходимые компоненты — дополнительные пакеты и утилиты. Для начала, обновим список пакетов и саму систему:

```
apt update && apt upgrade
```

Выполняем установку пакетов:

```
apt install curl apt-transport-https  
git iptables-persistent
```

\* где:

- **git** — утилита для работы с GIT. Понадобится для загрузки файлов из репозитория *git*.
- **curl** — утилита для отправки GET, POST и других запросов на http-сервер. Понадобится для загрузки ключа репозитория *Kubernetes*.
- **apt-transport-https** — позволяет получить доступ к АРТ-репозиториям по протоколу *https*.
- **iptables-persistent** — утилита для сохранения правил, созданных в *iptables* (не обязательна, но повышает удобство).

В процессе установки *iptables-persistent* может запросить подтверждение сохранить правила брандмауэра — отказываемся.

3. Отключаем файл подкачки. С ним *Kubernetes* не запустится.

Выполняем команду для разового отключения:

```
swapoff -a
```

Чтобы *swap* не появился после перезагрузки сервера, открываем на редактирование файл:

```
vi /etc/fstab
```

И комментируем строку:

```
#!/swap.img      none      swap      sw          0
0
```

4. Загружаем дополнительные модули ядра.

```
vi /etc/modules-load.d/k8s.conf
```

```
br_netfilter
overlay
```

\* модуль **br\_netfilter** расширяет возможности *netfilter* ([подробнее](#)); **overlay** необходим для *Docker*.

Загрузим модули в ядро:

```
modprobe br_netfilter
```

```
modprobe overlay
```

Проверяем, что данные модули работают:

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

Мы должны увидеть что-то на подобие:

```
overlay          114688  10
br_netfilter     28672   0
bridge          176128  1 br_netfilter
```

5. Изменим параметры ядра.

Создаем конфигурационный файл:

```
vi /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

\* ***net.bridge.bridge-nf-call-iptables*** контролирует возможность обработки трафика через bridge в netfilter. В нашем примере мы разрешаем данную обработку для IPv4 и IPv6.

Применяем параметры командой:

```
sysctl --system
```

## Брандмауэр

Для мастер-ноды и рабочей создаем разные наборы правил.

По умолчанию, в Ubuntu брандмауэр настроен на разрешение любого трафика. Если мы настраиваем наш кластер в тестовой среде, настройка брандмауэра не обязательна.

### 1. На мастер-ноде (Control-plane)

Выполняем команду:

```
iptables -I INPUT 1 -p tcp --match multiport --dports 6443,2379:2380,10250:10252 -j ACCEPT
```

\* в данном примере мы открываем следующие порты:

- **6443** — подключение для управления (Kubernetes API).
- **2379:2380** — порты для взаимодействия мастера с воркерами (etcd server client API).
- **10250:10252** — работа с kubelet (соответственно API, scheduler, controller-manager).

Для сохранения правил выполняем команду:

```
netfilter-persistent save
```

2. На рабочей ноде (Worker):

На нодах для контейнеров открываем такие порты:

```
iptables -I INPUT 1 -p tcp --match multiport --dports 10250,30000:32767 -j ACCEPT
```

\* где:

- **10250** — подключение к kubelet API.
- **30000:32767** — рабочие порты по умолчанию для подключения к подам (NodePort Services).

Сохраняем правила командой:

```
netfilter-persistent save
```

## Установка и настройка Docker

На все узлы кластера выполняем установку Docker следующей командой:

```
apt install docker docker.io
```

После установки разрешаем автозапуск сервиса docker:

```
systemctl enable docker
```

Создаем файл:

```
vi /etc/docker/daemon.json
```

```
{
  "exec-opts":
  ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
```

\* для нас является важной настройкой **cgroupdriver** — она должна быть выставлена в значение **systemd**. В противном случае, при создании кластера Kubernetes выдаст предупреждение. Хоть на возможность работы последнего

*это не влияет, но мы постараемся выполнить развертывание без ошибок и предупреждений со стороны системы.*

И перезапускаем docker:

```
systemctl restart docker
```

## Установка Kubernetes

Установку необходимых компонентов выполним из репозитория. Добавим его ключ для цифровой подписи:

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

Создадим файл с настройкой репозитория:

```
vi /etc/apt/sources.list.d/kubernetes.list  
  
deb https://apt.kubernetes.io/ kubernetes-  
xenial main
```

Обновим список пакетов:

```
apt update
```

Устанавливаем пакеты:

```
apt install kubelet kubeadm kubectl
```

\* где:

- **kubelet** — сервис, который запускается и работает на каждом узле кластера. Следит за работоспособностью подов.
- **kubeadm** — утилита для управления кластером Kubernetes.
- **kubectl** — утилита для отправки команд кластеру Kubernetes.

Нормальная работа кластера сильно зависит от версии установленных пакетов. Поэтому бесконтрольное их обновление может привести к потере работоспособности всей системы. Чтобы этого не произошло, запрещаем обновление установленных компонентов:

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

Установка завершена — можно запустить команду:

```
kubectl version --client --output=yaml
```

... и увидеть установленную версию программы:

```
clientVersion:
  buildDate: "2022-07-13T14:30:46Z"
  compiler: gc
  gitCommit:
aeef86a93758dc3cb2c658dd9657ab4ad4afc21cb
  gitTreeState: clean
  gitVersion: v1.24.3
  goVersion: go1.18.3
  major: "1"
  minor: "24"
  platform: linux/amd64
kustomizeVersion: v4.5.4
```

Наши серверы готовы к созданию кластера.

## Создание кластера

По-отдельности, рассмотрим процесс настройки мастер ноды (control-plane) и присоединения к ней двух рабочих нод (worker).

### Настройка control-plane (мастер ноды)

Выполняем команду на мастер ноду:

```
kubeadm init --pod-network-cidr=10.244.0.0/16
```

*\* данная команда выполнит начальную настройку и подготовку основного узла кластера. Ключ **--pod-network-cidr** задает адрес внутренней подсети для нашего кластера.*

Выполнение займет несколько минут, после чего мы увидим что-то на подобие:

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

...

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

kubeadm join 192.168.0.15:6443 --token
f7sihu.wmgzwxkvbr8500al \
--discovery-token-ca-cert-hash
sha256:6746f66b2197ef496192c9e240b31275747734cf
74057e04409c33b1ad280321
```

*\* данную команду (выделена желтым) нужно вводить на worker нодах, чтобы присоединить их к нашему кластеру. Можно ее скопировать, но позже мы будем генерировать данную команду по новой.*

В окружении пользователя создаем переменную KUBECONFIG, с помощью которой будет указан путь до файла конфигурации kubernetes:

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

Чтобы каждый раз при входе в систему не приходилось повторять данную команду, открываем файл:

```
vi /etc/environment
```

И добавляем в него строку:

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

Посмотреть список узлов кластера можно командой:

```
kubectl get nodes
```

На данном этапе мы должны увидеть только мастер ноду:

NAME	AGE	VERSION	STATUS	ROLES
k8s-master.dmosk.local	10m	v1.20.2	NotReady	<none>

Чтобы завершить настройку, необходимо установить CNI (Container Networking Interface) — в моем примере это flannel:

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

*\* краткий обзор и сравнение производительности CNI можно почитать в [статье на хабре](#).*

Узел управления кластером готов к работе.

## Настройка worker (рабочей ноды)

Мы можем использовать команду для присоединения рабочего узла, которую мы получили после инициализации мастер ноды или вводим (на первом узле):



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

Данная команда покажет нам запрос на присоединения новой ноды к кластеру, например:

```
kubeadm join 192.168.0.15:6443 --token  
f7sihu.wmgzwxkvbr8500al \  
--discovery-token-ca-cert-hash  
sha256:6746f66b2197ef496192c9e240b31275747734cf  
74057e04409c33b1ad280321
```

Копируем его и используем на двух наших узлах. После завершения работы команды, мы должны увидеть:

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

На мастер ноде вводим:

```
kubectl get nodes
```

Мы должны увидеть:

NAME	AGE	VERSION	STATUS	ROLES
k8s-master1.dmosk.local	18m	v1.20.2	Ready	control-plane,master
k8s-worker1.dmosk.local	79s	v1.20.2	Ready	<none>
k8s-worker2.dmosk.local	77s	v1.20.2	NotReady	<none>

*\* обратите внимание, что нода k8s-worker2 имеет статус **NotReady**. Это значит, что настройка еще выполняется и необходимо подождать. Как правило, в течение 2 минут статус меняется на **Ready**.*

Наш кластер готов к работе. Теперь можно создавать поды, развертывания и службы. Рассмотрим эти процессы подробнее.

## Pods

Поды — неделимая сущность объекта в Kubernetes. Каждый Pod может включать в себя несколько контейнеров (минимум, 1). Рассмотрим несколько примеров, как работать с подами. Все команды выполняем на мастере.

## Создание

Поды создаются командой `kubectl`, например:

```
kubectl run nginx --image=nginx:latest --port=80
```

*\* в данном примере мы создаем под с названием **nginx**, который в качестве образа Docker будет использовать **nginx** (последнюю версию); также наш под будет слушать запросы на порту **80**.*

Чтобы получить сетевой доступ к созданному поду, создаем `port-forward` следующей командой:

```
kubectl port-forward nginx --address 0.0.0.0 8888:80
```

*\* в данном примере запросы к кластеру `kubernetes` на порт **8888** будут вести на порт **80** (который мы использовали для нашего пода).*

Команда **`kubectl port-forward`** является интерактивной. Ее мы используем только для тестирования. Чтобы пробросить нужные порты в Kubernetes используются `Services` — об этом будет сказано ниже.

Можно открыть браузер и ввести адрес <http://<IP-адрес мастера>:8888> — должна открыться страница приветствия для NGINX.

## Просмотр

Получить список всех подов в кластере можно командой:

```
kubectl get pods
```

Например, в нашем примере мы должны увидеть что-то на подобие:

NAME	READY	STATUS	RESTARTS	AGE
nginx	1/1	Running	0	3m26s

Посмотреть подробную информацию о конкретном поде можно командой:

```
kubectl describe pods nginx
```

## Запуск команд внутри контейнера

Мы можем запустить одну команду в контейнере, например, такой командой:

```
kubectl exec nginx -- date
```

*\* в данном примере будет запущена команда **date** внутри контейнера **nginx**.*

Также мы можем подключиться к командной строке контейнера командой:

```
kubectl exec --tty --stdin nginx -- /bin/bash
```

*\* обратите внимание, что не у всех образов для контейнеров установлена оболочка **bash**, например для образов на основе *alpine*. Если мы получим ошибку **error**: **Internal error occurred: error executing command in container**, то можно вместо **bash** попробовать ввести **sh**.*

## Удаление

Для удаления пода вводим:

```
kubectl delete pods nginx
```

## Использование манифестов

В продуктивной среде управление подами выполняется с помощью специальных файлов с описанием того, как должен создаваться и настраиваться под — манифестов. Рассмотрим пример создания и применения такого манифеста.

Создадим файл формата yaml:

```
vi manifest_pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web-srv
  labels:
    app: web_server
    owner: dmosk
    description: web_server_for_site
spec:
  containers:
    - name: nginx
      image: nginx:latest
      ports:
        - containerPort: 80
        - containerPort: 443
```

```
- name: php-fpm
  image: php-fpm:latest
  ports:
    - containerPort: 9000

- name: mariadb
  image: mariadb:latest
  ports:
    - containerPort: 3306
```

*\* в данном примере будет создан под с названием **web-srv**; в данном поде будет развернуто 3 контейнера — **nginx**, **php-fpm** и **mariadb** на основе одноименных образов.*

Для объектов Kubernetes очень важное значение имеют метки или labels. Необходимо всегда их описывать. Далее, данные метки могут использоваться для настройки сервисов и развертываний.

Чтобы применить манифест выполняем команду:

```
kubectl apply -f manifest_pod.yaml
```

Мы должны увидеть ответ:

```
pod/web-srv created
```

Смотрим поды командой:

```
kubectl get pods
```

Мы должны увидеть:

NAME	READY	STATUS	RESTARTS	AGE
web-srv	3/3	Ready	0	3m11s

*\* для **Ready** мы можем увидеть 0/3 или 1/3 — это значит, что контейнеры внутри пода еще создаются и нужно подождать.*

## Deployments

Развертывания позволяют управлять экземплярами подов. С их помощью контролируется их восстановление, а также балансировка нагрузки. Рассмотрим пример использования Deployments в Kubernetes.

## Создание

Deployment создаем командой со следующим синтаксисом:

```
kubectl create deploy <название для  
развертывания> --image <образ, который должен  
использоваться>
```

Например:

```
kubectl create deploy web-set --image  
nginx:latest
```

*\* данной командой мы создадим deployment с именем **web-set**;  
в качестве образа будем использовать **nginx:latest**.*

## Просмотр

Посмотреть список развертываний можно командой:

```
kubectl get deploy
```

Подробное описание для конкретного развертывания мы можем посмотреть так:

```
kubectl describe deploy web-set
```

*\* в данном примере мы посмотрим описание **deployment** с  
названием **web-set**.*

## Scaling

Как было написано выше, deployment может балансировать нагрузкой. Это контролируется параметром scaling:

```
kubectl scale deploy web-set --replicas 3
```

*\* в данном примере мы указываем для нашего созданного  
ранее deployment использовать 3 реплики — то есть  
Kubernetes создаст 3 экземпляра контейнеров.*

Также мы можем настроить автоматическую балансировку:

```
kubectl autoscale deploy web-set --min=5 --  
max=10 --cpu-percent=75
```

*В данном примере Kubernetes будет создавать от 5 до 10  
экземпляров контейнеров — добавление нового экземпляра  
будет происходить при превышении нагрузки на процессор  
до 75% и более.*

Посмотреть созданные параметры балансировки можно командой:

```
kubectl get hpa
```

## Редактирование

Для нашего разворачивания мы можем изменить используемый образ, например:

```
kubectl set image deploy/web-set  
nginx=httpd:latest --record
```

*\* данной командой для deployment web-set мы заменим образ nginx на httpd; ключ **record** позволит нам записать действие в историю изменений.*

Если мы использовали ключ **record**, то историю изменений можно посмотреть командой:

```
kubectl rollout history deploy/web-set
```

Перезапустить deployment можно командой:

```
kubectl rollout restart deploy web-set
```

## Манифест

Как в случае с подами, для создания разворачиваний мы можем использовать манифесты. Попробуем рассмотреть конкретный пример.

Создаем новый файл:

```
vi manifest_deploy.yaml
```

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: web-deploy  
  labels:  
    app: web_server  
    owner: dmitriy_mosk  
    description: web_server_for_site  
spec:  
  replicas: 5  
  selector:  
    matchLabels:  
      project: myweb  
  template:  
    metadata:
```

```
    labels:
      project: myweb
      owner: dmitriy_mosk
      description: web_server_pod
  spec:
    containers:
      - name: myweb-httpd
        image: httpd:latest
        ports:
          - containerPort: 80
          - containerPort: 443

---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: web-deploy-autoscaling
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myweb-autoscaling
  minReplicas: 5
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 75
    - type: Resource
      resource:
        name: memory
        target:
          type: Utilization
          averageUtilization: 80
```

*\* в данном манифесте мы создадим deployment и autoscaling.*

*Итого, мы получим 5 экземпляров подов для развертывания web-deploy, которые могут быть расширены до 10 экземпляров. Добавление нового будет происходить при превышении нагрузки на процессор более чем на 75% или потреблением оперативной памяти более чем на 80%.*

*\*\* обратите внимание, что в названиях и тегах не должны использоваться символы в верхнем регистре, а также пробелы.*

Чтобы создать объекты с помощью нашего манифеста вводим:

```
kubectl apply -f manifest_deploy.yaml
```

Мы должны увидеть:

```
deployment.apps/web-deploy created
horizontalpodautoscaler.autoscaling/web-deploy-autoscaling created
```

Объекты web-deploy и web-deploy-autoscaling созданы.

## Удаление

Для удаления конкретного развертывания используем команду:

```
kubectl delete deploy web-set
```

Для удаления всех развертываний вместо названия deployment указываем ключ --all:

```
kubectl delete deploy --all
```

Удалить критерии autoscaling для конкретного развертывания можно командой:

```
kubectl delete hpa web-set
```

Удалить все критерии autoscaling можно командой:

```
kubectl delete hpa --all
```

Удалить объекты, созданные с помощью манифеста можно командой:

```
kubectl delete -f manifest_deploy.yaml
```

## Services

Службы позволяют обеспечить сетевую доступность для развертываний. Существует несколько типов сервисов:

- **ClusterIP** — сопоставление адреса с deployments для подключений внутри кластера Kubernetes.
- **NodePort** — для внешней публикации развертывания.
- **LoadBalancer** — сопоставление через внешний балансировщик.
- **ExternalName** — сопоставляет службу по имени (возвращает значение записи CNAME).

Мы рассмотрим первые два варианта.



## Привязка к Deployments

Попробуем создать сопоставления для ранее созданного развертывания:

```
kubectl expose deploy web-deploy --  
type=ClusterIP --port=80
```

\* где **web-deploy** — *deployment*, который мы развернули с помощью манифеста. Публикация ресурса происходит на внутреннем порту **80**. Обращаться к контейнерам можно внутри кластера *Kubernetes*.

Для создания сопоставления, с помощью которого можно будет подключиться к контейнерам из внешней сети выполняется командой:

```
kubectl expose deploy web-deploy --  
type=NodePort --port=80
```

\* данная команда отличается от команды выше только типом **NodePort** — для данному *deployment* будет сопоставлен порт для внешнего подключения, который будет вести на внутренний (в нашем примере, **80**).

## Просмотр

Чтобы посмотреть созданные нами службы, вводим:

```
kubectl get services
```

Мы можем увидеть что-то на подобие:

NAME	TYPE	CLUSTER-IP
EXTERNAL-IP	PORT(S)	AGE
web-deploy	NodePort	10.111.229.132
<none>	80:30929/TCP	21s

\* в данном примере указано, что у нас есть служба типа **NodePort**, а к сервису можно подключиться по порту **30929**.

Можно попробовать открыть браузер и ввести <http://<IP-адрес мастера>:30929> — мы должны увидеть нужную нам страницу (в наших примерах, либо NGINX, либо Apache).

Посмотреть список сервисов с указанием селектором можно командой:

```
kubectl get services -o wide
```

## Удаление

Удаляем созданную службу командой:

```
kubectl delete services web-deploy
```

*\* в данном примере будет удалена служба для развертывания **web-deploy**.*

Удалить все службы можно командой:

```
kubectl delete services --all
```

## Манифест

Как в случае с подами и развертываниями, мы можем использовать манифест-файлы. Рассмотрим небольшой пример.

```
vi manifest_service.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
  labels:
    app: web_server
    owner: dmosk
    description: web_server_for_site
spec:
  selector:
    project: myweb
  type: NodePort
  ports:
    - name: app-http
      protocol: TCP
      port: 80
      targetPort: 80

    - name: app-smtp
      protocol: TCP
      port: 25
      targetPort: 25
```

*\* в данном примере мы создадим службу, которая будем связываться с развертыванием по лейболу **project: myweb**.*

## ConfigMaps

ConfigMap позволяет нам хранить конфигурационные файлы в K8S и, при необходимости, монтировать их к определенным подам. Рассмотрим процесс их создания и использования.

## Создание

Мы можем в качестве источника данных использовать файлы, директории, значения командной строки или env-файл (в формате ключ-значение). Разберемся по порядку.

1. Конфиг из файла:

```
kubectl create configmap nginx-base --from-file=/opt/k8s/configmap/nginx.conf
```

*\* в нашем примере мы создадим конфигурацию **nginx-base**, а значением будет содержимое файла **/opt/k8s/configmap/nginx.conf**.*

2. Из директории:

```
kubectl create configmap nginx-base --from-file=/opt/k8s/configmap/
```

*\* в данном примере в конфигурацию **nginx-base** попадут все файлы из каталога **/opt/k8s/configmap**.*

3. Значения можно указать в команде:

```
kubectl create configmap nginx-spec --from-literal=hostname=dmosk.local --from-literal=cores=8
```

4. Используем env-файл:

```
kubectl create configmap nginx-properties --from-env-file=/opt/k8s/configmap/nginx.env
```

## Просмотр

Показать список всех созданных в k8s конфигов:

```
kubectl get configmaps
```

Подробная информация с содержимым:

```
kubectl get configmaps -o yaml nginx-base
```

## Привязка к контейнерам

Значения наших конфигураций могут быть смонтированы в качестве файлов или системных переменных. Также мы можем их применять как к подам, так и разворачиваниям.

Рассмотрим вариант использования с описанием в манифест-файле.

#### 1. Монтирование в качестве конфигурационного файла:

```
...
spec:
  containers:
    - name: myweb-nginx
      image: nginx:latest
      ...
      volumeMounts:
        - name: config-nginx
          mountPath: /etc/nginx
          subPath: nginx.conf
  volumes:
    - name: config-nginx
      configMap:
        name: nginx-base
```

\* давайте разбираться подробнее, что мы настроили:

- обратите внимание, что нам подходит манифест для *Pod* или *Deployment* — работаем с конфигурацией для **containers**.
- мы создали **volume config-nginx**, который берет значение из конфигурации **nginx-base**.
- к нашему контейнеру мы подключили созданный **config-nginx** с помощью опции **volumeMounts**. А при помощи **mountPath** мы указываем путь, куда будет примонтирована конфигурация.
- в качестве файла назначения мы задали **nginx.conf**. Это делается при помощи опции **subPath**.

#### 2, Использование для создания системных переменных:

```
...
spec:
  containers:
    - name: myweb-nginx
      image: nginx:latest
      ...
      envFrom:
        - configMapRef:
            name: nginx-spec
```

\* в данном примере все немного проще — мы указываем с помощью директивы **envFrom**, что мы должны создать системные переменные из конфига **nginx-spec**.

Для применения изменений вводим:

```
kubectl apply -f manifest_file.yaml
```

\* где **manifest\_file.yaml** — файл, в котором мы сделали правки.

## Удаление

Удалить конфиг можно следующей командой:

```
kubectl delete configmap nginx-spec
```

## Ingress Controller

В данной инструкции не будет рассказано о работе с Ingress Controller. Оставляем данный пункт для самостоятельного изучения.

Данное приложение позволяет создать балансировщик, распределяющий сетевые запросы между нашими сервисами. Порядок обработки сетевого трафика определяем с помощью Ingress Rules.

Существует не маленькое количество реализаций Ingress Controller — их сравнение можно найти в документе по ссылке в [Google Docs](#).

Для установки Ingress Controller Contour (среди множества контроллеров, он легко устанавливается и на момент обновления данной инструкции полностью поддерживает последнюю версию кластера Kubernetes) вводим:

```
kubectl apply -f  
https://projectcontour.io/quickstart/contour.yaml
```

## Установка веб-интерфейса

Веб-интерфейс позволяет получить информацию о работе кластера в удобном для просмотра виде.

В большинстве инструкций рассказано, как получить доступ к веб-интерфейсу с того же компьютера, на котором находится кластер (по адресу 127.0.0.1 или localhost). Но мы рассмотрим настройку для удаленного подключения, так как это более актуально для серверной инфраструктуры.

Переходим на страницу веб-интерфейса в [GitHub](#) и копируем ссылку на последнюю версию файла yaml:

## Installation

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.1.0/aio/deploy/recommended.yaml
```

*\* на момент обновления инструкции, последняя версия интерфейса была **2.1.0**.*

Скачиваем yaml-файл командой:

```
wget
https://raw.githubusercontent.com/kubernetes/da
shboard/v2.1.0/aio/deploy/recommended.yaml
```

\*

где <https://raw.githubusercontent.com/kubernetes/dashboard/v2.1.0/aio/deploy/recommended.yaml> — ссылка, которую мы скопировали на портале GitHub.

Открываем на редактирование скачанный файл:

```
vi recommended.yaml
```

Комментируем строки для **kind: Namespace** и **kind: Secret** (в файле несколько блоков с **kind: Secret** — нам нужен тот, что с **name: kubernetes-dashboard-certs**):

```
...
#apiVersion: v1
#kind: Namespace
#metadata:
#  name: kubernetes-dashboard
...
#apiVersion: v1
#kind: Secret
#metadata:
#  labels:
#    k8s-app: kubernetes-dashboard
#  name: kubernetes-dashboard-certs
#  namespace: kubernetes-dashboard
#type: Opaque
```

*\* нам необходимо закомментировать эти блоки, так как данные настройки в Kubernetes мы должны будем сделать вручную.*

Теперь в том же файле находим **kind: Service** (который с **name: kubernetes-dashboard**) и добавляем строки **type: NodePort** и **nodePort: 30001** (выделены красным):

```
kind: Service
apiVersion: v1
```

```
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kubernetes-dashboard
spec:
  type: NodePort
  ports:
    - port: 443
      targetPort: 8443
      nodePort: 30001
  selector:
    k8s-app: kubernetes-dashboard
```

*\* таким образом, мы публикуем наш сервис на внешнем адресе и порту **30001**.*

Для подключения к веб-интерфейсу не через локальный адрес, начиная с версии 1.17, обязательно необходимо использовать зашифрованное подключение (https). Для этого нужен сертификат. В данной инструкции мы сгенерируем самоподписанный сертификат — данный подход удобен для тестовой среды, но в продуктивной среде необходимо купить сертификат или получить его бесплатно в [Let's Encrypt](https://letsencrypt.org/).

И так, создаем каталог, куда разместим наши сертификаты:

```
mkdir -p /etc/ssl/kubernetes
```

Сгенерируем сертификаты командой:

```
openssl req -new -x509 -days 1461 -nodes -out
/etc/ssl/kubernetes/cert.pem -keyout
/etc/ssl/kubernetes/cert.key -subj
"/C=RU/ST=SPb/L=SPb/O=Global Security/OU=IT
Department/CN=kubernetes.dmosk.local/CN=kuberne
tes"
```

*\* можно не менять параметры команды, а так их и оставить. Браузер все-равно будет выдавать предупреждение о неправильном сертификате, так как он самоподписанный.*

Создаем namespace:

```
kubectl create namespace kubernetes-dashboard
```

*\* это та первая настройка, которую мы комментировали в скачанном файле **recommended.yaml**.*

Теперь создаем настройку для secret с использованием наших сертификатов:

```
kubectl create secret generic kubernetes-  
dashboard-certs --from-  
file=/etc/ssl/kubernetes/cert.key --from-  
file=/etc/ssl/kubernetes/cert.pem -n  
kubernetes-dashboard
```

*\* собственно, мы не использовали настройку в скачанном файле, так как создаем ее с включением в параметры пути до созданных нами сертификатов.*

Теперь создаем остальные настройки с помощью скачанного файла:

```
kubectl create -f recommended.yaml
```

Мы увидим что-то на подобие:

```
serviceaccount/kubernetes-dashboard created  
service/kubernetes-dashboard created  
secret/kubernetes-dashboard-csrf created  
secret/kubernetes-dashboard-key-holder created  
configmap/kubernetes-dashboard-settings created  
role.rbac.authorization.k8s.io/kubernetes-  
dashboard created  
clusterrole.rbac.authorization.k8s.io/kubernet  
e-s-dashboard created  
rolebinding.rbac.authorization.k8s.io/kubernet  
e-s-dashboard created  
clusterrolebinding.rbac.authorization.k8s.io/k  
ubernetes-dashboard created  
deployment.apps/kubernetes-dashboard created  
service/dashboard-metrics-scraper created  
deployment.apps/dashboard-metrics-scraper  
created
```

Создадим настройку для админского подключения:

```
vi dashboard-admin.yaml
```

```
apiVersion: v1  
kind: ServiceAccount  
metadata:  
  labels:  
    k8s-app: kubernetes-dashboard  
  name: dashboard-admin  
  namespace: kubernetes-dashboard  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
  name: dashboard-admin-bind-cluster-role  
  labels:
```



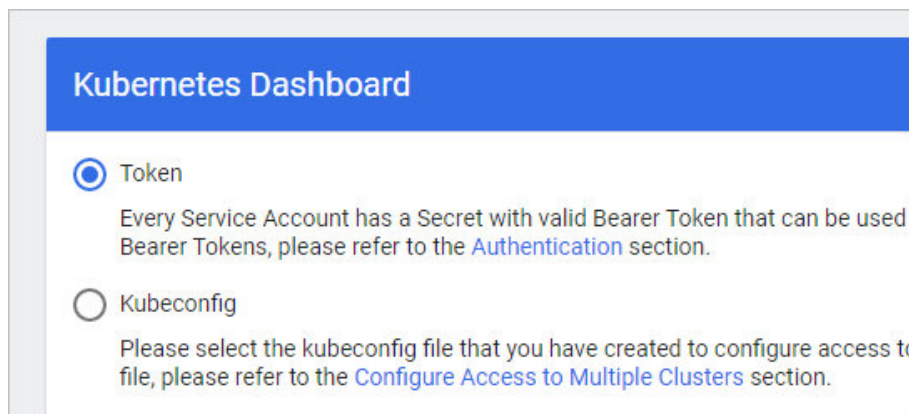
```
k8s-app: kubernetes-dashboard
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: dashboard-admin
  namespace: kubernetes-dashboard
```

Создаем настройку с применением созданного файла:

```
kubectl create -f dashboard-admin.yaml
```

Теперь открываем браузер и переходим по ссылке <https://<IP-адрес мастера>:30001> — браузер покажет ошибку сертификата (если мы настраиваем по инструкции и сгенерировали самоподписанный сертификат). Игнорируем ошибку и продолжаем загрузку.

Kubernetes Dashboard потребует пройти проверку подлинности. Для этого можно использовать токен или конфигурационный файл:



На сервере вводим команду для создания сервисной учетной записи:

```
kubectl create serviceaccount dashboard-admin -n kube-system
```

Создадим привязку нашего сервисного аккаунта с Kubernetes Dashboard:

```
kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --serviceaccount=kube-system:dashboard-admin
```

Теперь командой:

```
kubectl describe secrets -n kube-system
$(kubectl -n kube-system get secret | awk
'/dashboard-admin/{print $1}')
```

... получаем токен для подключения (выделен красным):

```
Data
====
ca.crt:      1066 bytes
namespace:   11 bytes
token:
  eyJhbGciOiJSUzI1NiIsImtpZCI6IkpCT0J5TWV2VWxWQU
  otdHZYc1Fuawza2NFTW1IZVNuSlZSN3hwUzFrNTAifQ.eyJ
  Jpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia
  3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3Bh
  Y2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2V
  ydm1jZWJjY291bnQvc2VjcmV0Lm5hbWUiOiJkYXNoYm9hcm
  QtYWRtaW4tdG9rZW4tbnRqNmYiLCJrdWJlcm5ldGVzLm1vL
  3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWVjb3VudC5uYW1l
  IjoizGFzaGJvYXJkLWZkbWluIiwia3ViZXJuZXRlcy5pby9
  zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWJjY291bnQudWlkIj
  OiMzIiwiaWF0Ij0iMjAyMjA0MjYyMjYyMjYyMjYyMjYyMjY
  2VhIiwic3ViIjoic3lzdGVtOnNlcnZpY2VhY2NvdW50Omt1
  YmUtc3lzdGVtOmRhc2hib2FyZC1hZG1pbjI9LmV0dGeNiTCR
  BakDp106PbdqVPH_W2EsBgJjZnTDFlnP3cdXQv6VgBkI8N
  a1p1XDRF-
  lF36KbbC2hpRjbkbLrLW7BemIVwY0znmc8kmrgCSx02FV19
  3NK3biE9TKdlj1Bbd1yf0086L56vteXGP20X0Xs1h3cjAsh
  S-
  70bsnJl6z3MY5GbRvej0yVzq_PWMVYsqvQhssExsJM2tKJW
  G0DnXCW687XH1stbYUoLhxSRoRpMZk-JrguuwgLH5FYIIU-
  ZdTZA6mz-
  _hqrX8PoDvqEfWrSn1M6Q0k8U3TMaDLlduzA7rwLRJBQt3C
  0aD6XFR9wHUqUWd5y953u67wpFPrSA
```

Используя полученный токен, вводим его в панели авторизации:

## Kubernetes Dashboard

☒ Token

Every Service Account has a Secret with valid Bearer Token that can be used. Bearer Tokens, please refer to the [Authentication](#) section.


☐ Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. Please refer to the [Configure Access to Multiple Clusters](#) section.

Enter token \*

Sign in

Мы должны увидеть стартовое окно системы управления:

 **kubernetes**

default

Search


### Overview

**Workloads** N

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs

**Service**

Services

Name	Namespace
 <b>kubernetes</b>	default

## Удаление нод

При необходимости удалить ноду из нашего кластера, вводим 2 команды:

```
kubectl drain k8s-worker2.dmosk.local --ignore-daemonsets
```

```
kubectl delete node k8s-worker2.dmosk.local
```

*\* в данном примере мы удаляем ноду **k8s-worker2.dmosk.local**.*



Была ли полезна вам эта инструкция?

Да

Нет

[Tweet](#)



*Дмитрий Моск — IT-специалист.  
Настройка серверов, компьютерная помощь.*

## Инструкции

[Как собрать свой собственный deb-пакетов с нуля под Linux Debian](#)

[Примеры создания пакетов RPM из исходников или со своими файлами](#)

[Как установить и использовать сервер хранения секретов Hashicorp Vault](#)

[Установка и настройка файлового сервера Samba на Ubuntu](#)

[Установка и настройка кластера Kubernetes на Linux Ubuntu](#)

[Как настроить почту для корпоративной среды на Ubuntu Server](#)

[Установка, настройка и использование системы по сбору логов Grafana Loki на Linux](#)

[Как установить и настроить связку Asterisk + FreePBX на CentOS 8](#)

[Как установить и настроить прокси-сервер Squid на Ubuntu Server](#)

[Как настроить почту для корпоративной среды на CentOS 8](#)

[Весь список ...](#)

Нужна инструкция? Пишите:

Что хотите узнать...

Контактная эл. почта

Запросить инструкцию

Реклама