DOI: 10.18137/RNU.V9187.22.02.P.149 УДК 004.4

В.Г. Мазманян, Э.М. Вихтенко

АЛГОРИТМ ЗАПУСКА МНОГОУЗЛОВОГО КЛАСТЕРА KUBERNETES И РАБОТА С НИМ В ОПЕРАЦИОННОЙ СИСТЕМЕ CENTOS LINUX

Аннотация. Описаны алгоритм запуска двухузлового кластера Kubernetes с использованием среды запуска контейнеров Docker в операционной системе Linux CentOS и алгоритм развертки простого веб-приложения Node.js в этом кластере.

Ключевые слова: CentOS, Docker, Kubernetes, Linux, контейнеризация, многоузловой кластер, развертка приложений.

V.G. Mazmanyan, E.M. Vikhtenko

ALGORITHM FOR STARTING A MULTINODE CLUSTER KUBERNETES AND WORKING WITH IT IN THE CENTOS LINUX OPERATING SYSTEM

Abstract. The article describes the algorithm for running a two-node Kubernetes cluster using the Docker container runtime on Linux CentOS operating system and the algorithm for deploying a simple Node.js web application in this cluster.

Keywords: CentOS, Docker, Kubernetes, Linux, containerization, multi-node cluster, application deployment.

Введение

Все чаще в IT-сфере звучит термин DevOps (англ. Development & Operations – «разработка и эксплуатация»), возникший на пересечении двух направлений работы с информационно-технологическими (далее – ИТ) продуктами. DevOps можно обозначить как методология, согласно которой происходит взаимодействие инженеров по разработке и инженеров по ИТ-эксплуатации и взаимная интеграция их рабочих процессов для обеспечения качественной работы продукта [1; 7; 10]. Практики DevOps используются для улучшения эффективности управленческих и коммуникационных процессов. Одной из важных целей в методологии DevOps является использование единой платформы и окружения для разработки и доставки продуктов. Достигнуть этой цели позволяет использование контейнеризированных приложений.

Целью данной работы является создание и реализация системного решения на основе технологий Kubernetes, которое бы позволило разработчикам самостоятельно разворачивать и сопровождать собственные приложения на промышленном уровне.

Контейнеризация

Контейнеризация – это разделение приложения на самостоятельные микросервисы, которые изолируются друг от друга внутри операционной системы в так называемые контейнеры. Контейнер можно представить как единственный работающий в системе процесс. По сравнению с виртуализацией, когда для изоляции каждого процесса используется отдельная виртуальная машина, контейнеризация позволяет запускать гораздо большее количество процессов на одном и том же оборудовании за счет того, что виртуальной машине требуются дополнительные вычислительные ресурсы для собственной работы.

Мазманян Виктория Германовна
студент. Тихоокеанский государственный университет, город Хабаровск. Сфера научных интересов: проектирование и разработка программно-информационных систем.
Электронный адрес: mail@pnu.edu.ru

Вихтенко Эллина Михайловна
кандидат физико-математических наук, доцент, доцент кафедры программного обеспечения вычислительной техники и автоматизированных систем. Тихоокеанский государственный университет, город Хабаровск. Сфера научных интересов: проектирование и разработка программно-информационных систем. Автор более 80 опубликованных научных работ.
Электронный адрес: mail@pnu.edu.ru

Микросервисы автономны, поэтому разрабатывать, развертывать, обновлять и масштабировать их можно по отдельности, что позволяет заменять компоненты в соответствие с меняющимися потребностями бизнеса.

Однако с увеличением числа развертываемых в контейнерах сервисов увеличивается и сложность настройки, управления и обеспечения бесперебойной работы системы в целом. Для автоматизации настройки компонентов, их контроля и обработки аварийных ситуаций используется инструмент-оркестратор Kubernetes [2; 4–6]. Kubernetes – это платформа с открытым исходным кодом, которая позволяет развертывать контейнеризированные рабочие нагрузки и управлять ими.

Контейнеры внутри Kubernetes разворачиваются в кластере. Кластер состоит из нескольких физических или виртуальных машин, которые в терминологии Kubernetes называются узлами (Nodes).

Узлы в кластере делятся на два типа – ведущие и рабочие. На рабочих узлах запускается полезная нагрузка, то есть контейнеризированные приложения и сервисы. На ведущих узлах запускаются компоненты управляющего слоя Kubernetes, которые отвечают за распределение нагрузки по рабочим узлам.

Приложения в кластере Kubernetes разворачиваются внутри среды запуска контейнеров (ContainerRuntime Interface), где каждый контейнер запускается в специально созданном модуле (Pod).

Алгоритм развертывания простого приложения

Продемонстрируем алгоритм развертывания простого приложения Node.js в кластере Kubernetes на операционной системе Linux CentOS 7 [8; 9]. В качестве среды запуска контейнеров в Kubernetes выбран инструмент Docker. Алгоритм включает в себя настройку операционной системы, установку компонентов Docker и Kubernetes, а также создание контейнерного образа приложения.

В качестве узлов кластера использованы две виртуальные машины со следующими минимальными характеристиками (см. Таблицу):

Характеристики виртуальных машин

Роль	Хост	IP адрес	Операционная система	ОЗУ	Процессор
Ведущий	kmaster	192.168.88.72	CentOS 7	2 Гб	2 CPU
Рабочий	kworker	192.168.88.73	CentOS 7	1 Гб	1 CPU

Алгоритм развертывания кластера включает в себя ряд шагов.

Шаг 1. Настройка операционной системы.

Чтобы контейнеры имели доступ к файловой системе, необходимо отключить службу SELinux, выполнив команды

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

Для обеспечения связи между контейнерами и подами, необходимо добавить набор правил в службу брандмауэра, выполнив следующие команды:

```
# firewall-cmd --permanent --add-port=6443/tcp
# firewall-cmd --permanent --add-port=2379-2380/tcp
# firewall-cmd --permanent --add-port=10250/tcp
# firewall-cmd --permanent --add-port=10251/tcp
# firewall-cmd --permanent --add-port=10252/tcp
# firewall-cmd --permanent --add-port=10255/tcp
# firewall-cmd –-reload
```

Для обеспечения корректной работы служб Kubernetes в системе должен быть отключен swap, а также включен параметр net.bridge.bridge-nf-call-iptables при помощи следующих команд:

```
# swapoff -a; sed -i '/swap/d' /etc/fstab
 # cat >>/etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
# sysctl–system
```

Шаг 2. Установка компонентов Docker и Kubernetes.

Далее на всех узлах должны быть установлены компоненты служб Kubernetes и среда запуска контейнеров Docker. Их установка происходит из официальных репозиториев. Эти репозитории не доступны из операционной системы по умолчанию, поэтому перед установкой их необходимо подключить. Репозиторий Docker доступен в официальном наборе репозиториев CentOS и подключается при помощи команды

```
# yum-config-manager \ --add-repo \
https://download.docker.com/linux/centos/docker-ce.repo
```

Репозиторий Kubernetes не входит в официальный набор, поэтому чтобы его подключить, необходимо самостоятельно создать файл с описанием репозитория.

Листинг 1. Содержимое файла /etc/yum.repos.d/kubernetes.repo

```
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
```

gpgcheck=0
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

После подключения репозиториев можно перейти к установке необходимых компонентов. Сначала устанавливаются все необходимые зависимости:

yum install -y yum-utils device-mapper-persistent-data lvm2

Затем устанавливаются компоненты Kubernetes и Docker:

yum install -y docker-cekubeadmkubeletkubectl

Выполнение указанной выше команды установит пакеты docker-ce (среда запуска контейнеров), kubeadm (инструмент для создания многоузловых кластеров), kubelet (контролер корректной работы контейнеров), kubectl (инструмент командной строки для взаимодействия с Kubernetes).

Когда установка всех компонентов закончится, необходимо запустить службу Docker при помощи команды# systemctlenable --nowdocker

На этом подготовка узлов к созданию кластера завершена и можно переходить к инициализации кластера.

Шаг 3. Создание кластера Kubernetes.

Инициализация кластера выполняется на будущем ведущем узле при помощи команды

sudokubeadminit --pod-network-cidr=10.0.0.0/23

Данная команда обозначит машину, на которой она запущена, в качестве ведущего узла в кластере и произведет его полную настройку, развернув все необходимые компоненты управляющего слоя. После выполнения данной команды в консоли появится текст, в котором будет представлена команда kubeadmjoin с указанием персонализированного токена (см. Рисунок 1). Полученная команда позволяет добавлять другие машины в качестве новых узлов. При этом стоит учесть, что токен действует 24 часа, а сгенерировать новый можно, выполнив команду

kubeadm token create --print-join-command

```
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 192.168.88.72:6443 --token cakmar.yw0nvpnmrshg7vqp \
        --discovery-token-ca-cert-hash sha256:3086e110eda25103131948a96ee43b4aa92f96cce1482ba67a2720ef5342b846
```

Рисунок 1. Результат выполнения команды kubeadminit

Полученную команду kubeadmjoin необходимо выполнить на машине, выделенной под рабочий узел. После присоединения нового узла к кластеру требуется явно обозначить его роль в качестве рабочего, выполнив команду

kubectl label node kworker node-role.kubernetes.io/worker=worker

Для взаимодействия с кластером через инструмент kubelet на ведущем узле необходимо создать директорию кластера и назначить права пользователю, выполнив команды

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudochown $(id -u):$(id -g) $HOME/.kube/config

Завершает развертывание кластера установка плагина контейнерного сетевого взаимодействия, который позволит узлам и контейнерам корректного общаться друг с другом. В данном примере устанавливается плагин Antrea при помощи команды

kubectl apply -f https://raw.githubusercontent.com/antrea-io/antrea/main/build/yamls/antrea.yml

Теперь, выполнив на ведущем узле команды kubectlgetnode и kubectlgetpod, можно увидеть состояние всех узлов кластера и статус развернутых на нем модулей (см. Рисунок 2).

```
[vvoyd@kmaster ~]$ kubectl get node
NAME                  STATUS   ROLES           AGE    VERSION
kmaster.k8s.example.ru   Ready    control-plane   11m    v1.24.1
kworker-k8s-example-ru   Ready    worker          5m0s   v1.24.1
[vvoyd@kmaster ~]$ kubectl get -A pods
NAMESPACE     NAME                                                READY   STATUS    RESTARTS        AGE
kube-system   antrea-agent-rkkhc                                  2/2     Running   2 (4m31s ago)   5m16s
kube-system   antrea-agent-sgxz9                                  2/2     Running   0               9m30s
kube-system   antrea-controller-7d67dcc8fd-v5p9q                  1/1     Running   0               9m30s
kube-system   coredns-6d4b75cb6d-497dq                            1/1     Running   0               11m
kube-system   coredns-6d4b75cb6d-68pdp                            1/1     Running   0               11m
kube-system   etcd-kmaster.k8s.example.ru                         1/1     Running   0               11m
kube-system   kube-apiserver-kmaster.k8s.example.ru               1/1     Running   0               11m
kube-system   kube-controller-manager-kmaster.k8s.example.ru      1/1     Running   0               11m
kube-system   kube-proxy-6lrk2                                    1/1     Running   0               11m
kube-system   kube-proxy-j96xz                                    1/1     Running   0               5m16s
kube-system   kube-scheduler-kmaster.k8s.example.ru               1/1     Running   0               11m
```

Рисунок 2. Списки узлов кластера и запущенных на нем модулей

Среди модулей можно увидеть и элементы управляющего слоя Kubernetes, запущенные на ведущем узле (antrea-agent, antrea-controller, coredns, etcd, kube-apiserver, kube-controller-manager, kube-proxy, kube-scheduler), и агенты, запущенные на рабочем узле (antrea-agent, kube-proxy). Таким образом, мы получили работающий двухузловой кластер Kubernetes.

Шаг 4. Создание контейнерного образа приложения.

Данный и последующий шаги выполняются на ведущем узле. Для примера развертывания используем простое приложение Node.js [3], состоящее из одного файла app.js, содержимое которого представлено листингом 2.

Листинг 2. Простое приложение Node.js: app.js

```
const http = require('http');
const os = require('os');
console.log(«Server starting...»);
var handler = function(request, response) {
console.log(«Received request from « + request.connection.remoteAddress);
response.writeHead(200);
response.end(«You've hit « + os.hostname() + «\n»);
};
var www = http.createServer(handler);
www.listen(8080);
```

Данное приложение принимает HTTP-запросы и отвечает текстом с именем хоста, на котором оно запущено. Развернув такое приложение в кластере Kubernetes, в ответе можно увидеть имя модуля, в котором запущен контейнерный образ приложения.

Чтобы развернуть приложение в кластере, необходимо создать его контейнерный образ. Для создания контейнерного образа, необходимо предоставить список инструкций, которые Docker будет выполнять при сборке. Инструкции указываются в специальном файле Dockerfile, который расположен в одной директории с файлами приложения. Содержимое файла Dockerfile для текущего примера представлено в Листинге 3.

Листинг 3. Файл Dockerfile для создания контейнерного образа приложения

```
FROM node:7
ADD app.js /app.js
ENTRYPOINT [“node”, “app.js”]
```

В первой строке указан базовый образ, на основе которого происходят дальнейшие настройки. В данном случае используется контейнерный образ платформы Node.js из официального репозитория Docker. Во второй строке добавляется файл app.js из локального каталога в корневой каталог образа. В третьей строке определяется команда, которая будет выполнена при запуске образа. В данном случае это команда node app.js, которая запускает приложение при помощи инструментов node.

Создание контейнерного образа выполняется командой (test – название образа) # dockerbuild -t test

Для запуска получившегося контейнера на любой другой машине, его нужно загрузить в общедоступное хранилище DockerHub, предварительно зарегистрировавшись на сайте http://hub.docker.com, создав свой собственный DockerID. Далее необходимо протегировать образ в соответствие с правилами Docker, согласно которым имя образа должно начинаться с DockerID. Создать новый тег для образа можно при помощи команды # dockertagttestvvoyd/test (здесь vvoyd – это Docker ID).

После создания тега необходимо авторизоваться в системе Docker под собственным ID командой dockerlogin и отправить образ в хранилище, выполнив команду

dockerpushvvoyd/test

Шаг 4. Развертывание приложения в кластере Kubernetes.

Перед запуском приложения в кластере рекомендуется создать собственное пространство имен для этого приложения, выполнив следующие команды:

kubectl create namespace **test-space**

kubectl config set-context --current --namespace=**test-space**

Когда приложение упаковано в образ контейнера и доступно через реестр Docker Hub, его можно развернуть в кластере Kubernetes при помощи команды# kubectl create deployment test --image=**vvoyd/test**. После завершения этой команды состояние нового модуля с приложением можно будет увидеть, выполнив команду kubectlgetpod, как показано на Рисунке 3.

Шаг 5. Получение доступа к приложению.

Каждый модуль при создании получает свой внутренний IP-адрес в кластере, недоступный из сети Интернет. Доступ извне можно получить при помощи службы NodePort, которая выделит внешний порт для доступа к запущенному приложению. Использовав этот порт вместе с IP-адресом одного из узлов кластера, можно отправить запрос напрямую в приложение. Создать службу и узнать выделенный порт можно при помощи команд

```
# kubectl expose deployment test --type=NodePort --name=test-service --port=8080
# kubectl get services
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
test-service NodePort 10.111.15.196 <none> 8080:31884/TCP 14s
```

```
[vvoyd@kmaster ~]$ kubectl get pod -A
NAMESPACE     NAME                                          READY   STATUS    RESTARTS        AGE
kube-system   antrea-agent-rkkhc                            2/2     Running   10 (5d1h ago)   5d23h
kube-system   antrea-agent-sgxz9                            2/2     Running   8 (5d2h ago)    5d23h
kube-system   antrea-controller-7d67dcc8fd-v5p9q            1/1     Running   4 (5d2h ago)    5d23h
kube-system   coredns-6d4b75cb6d-497dq                      1/1     Running   4 (5d2h ago)    5d23h
kube-system   coredns-6d4b75cb6d-68pdp                      1/1     Running   4 (5d2h ago)    5d23h
kube-system   etcd-kmaster.k8s.example.ru                   1/1     Running   4 (5d2h ago)    5d23h
kube-system   kube-apiserver-kmaster.k8s.example.ru         1/1     Running   4 (5d2h ago)    5d23h
kube-system   kube-controller-manager-kmaster.k8s.example.ru 1/1    Running   4 (5d2h ago)    5d23h
kube-system   kube-proxy-6lrk2                              1/1     Running   4 (5d2h ago)    5d23h
kube-system   kube-proxy-j96xz                              1/1     Running   4 (5d1h ago)    5d23h
kube-system   kube-scheduler-kmaster.k8s.example.ru         1/1     Running   4 (5d2h ago)    5d23h
test-space    test-6b8c746cbc-nrxnn                         1/1     Running   0               75s
```

Рисунок 3. Результат выполнения команды kubectlgetpod после развертывания приложения в кластере

Таким образом, приложение становится доступно из внешней сети по адресу 192.168.88.73:31884,

При отправке к нему запроса при помощи команды curl, можно увидеть ответ, в котором приложение сообщает имя модуля в качестве своего хоста:

curl http://192.168.88.73:31884/

You've hit test-6b8c746cbc-nrxnn

Поскольку каждый модуль ведет себя как отдельная машина с собственным именем хоста, приложение выглядит так, как будто оно запущено на выделенной специально под него машине, несмотря на то, что на самом деле оно запущено на одном из узлов кластера.

Заключение

Простой развертки приложения на стандартном кластере Kubernetes недостаточно для комфортной работы среднестатистического приложения в промышленной среде. Для отслеживания корректной работы приложения необходим визуальный мониторинг, для реактивной реакции на возможные недоступности нужна система оповещения, а для проактивной реакции может понадобиться автоматическая балансировка нагрузки.

Стандартный кластер Kubernetes не предоставляет подобных компонентов по умолчанию, однако существуют различные инструменты, интегрированные с Kubernetes, которые предоставляют различные сервисы.

Литература

1. *Вехен Д.* Безопасный DevOps. Эффективная эксплуатация систем. СПб.: Питер, 2020. 432 с.

2. *Документация по* Kubernetes [Электронный ресурс]. URL: https://kubernetes.io/ru/docs/concepts/overview/_print (дата обращения 20.05.22).

3. Контейнеризация приложений Java для Kubernetes [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/azure/developer/java/containers/kubernetes (дата обращения 20.05.22).

4. *Лукша М.* Kubernetes в действии. М.: ДМК Пресс, 2019. 672 с.

5. *Маркелов А. А.* Введение в технологии контейнеров и Kubernetes. М.: ДМК Пресс, 2019. 194 с.

6. *Сайфан Д.* Осваиваем Kubernetes. Оркестрация контейнерных архитектур. СПб.: Питер, 2019. 400 с.

7. *Davis J., Daniels K.* (2016) Effective DevOps: Building a Culture of Collaboration, Affinity and Toolong at Scale. O'Reilly, 410 p.

8. *Mallett A.* (2014) CentOS System Administration Essentials.Packt Publishing, 147 p.

9. *Membrey P., Verhoeven T., Angenendt R.* (2009) The Definitive Guide to CentOS. Springer-Verlag New York, 352 p.

10. *Rosso J., Lander R., Brand A.* (2021) Production Kubernetes: Building Successful Application Platforms. O'Reilly Media, 484 p.

References

1. Vekhen D. (2020) *Bezopasnyj DevOps. Effektivnaya ekspluataciya system* [Secure DevOps. Efficient operation of systems]. St. Petersburg, Piter Publishing, 432 p. (in Russian).
2. *Dokumentaciya po Kubernetes* [Cybernet documentation]. Available at: https://kubernetes.io/ru/docs/concepts/overview/_print/ (date of the application: 20.05.22) (in Russian).
3. *Kontejnerizaciya prilozhenij Java dlya Kubernetes* [Java Application Containerization for CyberNet]. Java Application Containerization for CyberNet https://docs.microsoft.com/ru-ru/azure/developer/java/containers/kubernetes (date of the application: 20.05.22) (in Russian).
4. Luksha M. (2019) *Kubernetes v dejstvii* [Cybernetic in action]. Moscow, DMK Press Publishing, 672 p. (in Russian).
5. Markelov A.A. (2019) *Vvedenie v tekhnologii kontejnerov i Kubernetes* [Introduction to container technology and cybernetic]. Moscow, DMK Press Publishing, 194 p. (in Russian).
6. Sajfan D. (2019) *Osvaivaem Kubernetes. Orkestraciya kontejnernyh arhitektur* [Mastering the Cybernet. Orchestration of container architectures]. St. Petersburg, Piter Publishing, 400 p. (in Russian).
7. Davis J., Daniels K. (2016) Effective DevOps: Building a Culture of Collaboration, Affinity, and Toolong at Scale. O'Reilly Media, 410 p.
8. Mallett A. (2014) CentOS System Administration Essentials. Packt Publishing, 147 p.
9. Membrey P., Verhoeven T., Angenendt R. (2009) The Definitive Guide to CentOS. Springer-Verlag New York, 352 p.
10. Rosso J., Lander R., Brand A. (2021) Production Kubernetes: Building Successful Application Platforms. O'Reilly Media, 484 p.