

Виртуализация: Kubernetes ч.2. (Внутреннее устройство)



Шило
Петр



Шило Петр

Devops-инженер
Arifonica

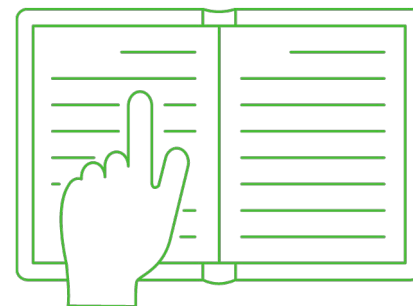


Предисловие

На этом занятии мы поговорим о:

- архитектуре k8s;
- сетевых плагинов;
- пакетном менеджере Helm.

По итогу занятия вы узнаете, как устроен kubernetes и научитесь устанавливать кластер с помощью kubectl, а также использовать helm для создания и установки чартов.



План занятия

1. [Предисловие](#)
2. [Архитектура k8s](#)
3. [Сетевые плагины](#)
4. [Установка с помощью kubeadm](#)
5. [Helm](#)
6. [Итоги](#)
7. [Домашнее задание](#)

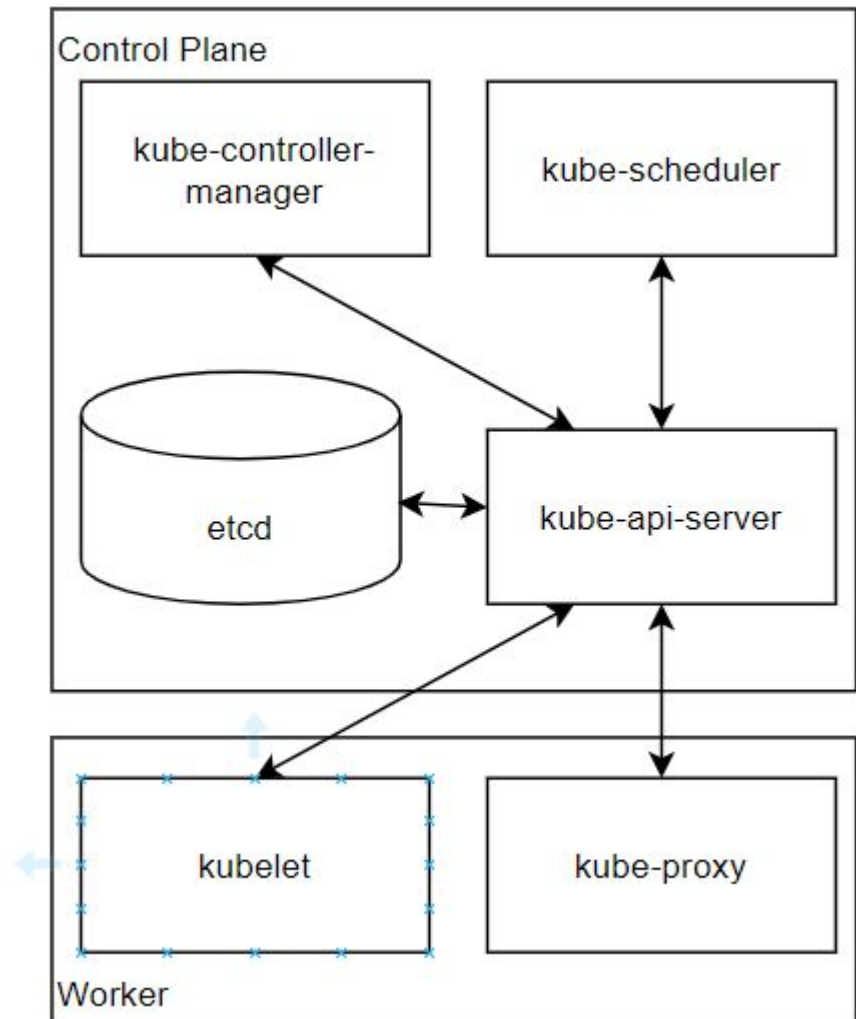


Архитектура k8s

Архитектура k8s

Компоненты:

- kube-api-server
- kube-controller-manager
- kube-scheduler
- etcd
- kube-proxy
- kubelet





Архитектура k8s: kube-api-server

Сервер API — компонент Kubernetes панели управления, который представляет API Kubernetes.

API-сервер — это клиентская часть панели управления Kubernetes.

Основной реализацией API-сервера Kubernetes является kube-apiserver. kube-apiserver предназначен для горизонтального масштабирования (развёртывание на несколько экземпляров). Вы можете запустить несколько экземпляров kube-apiserver и сбалансировать трафик между этими экземплярами.



Архитектура k8s: etcd

— распределенное и высоконадежное хранилище данных в формате "ключ-значение", которое используется как основное хранилище всех данных кластера в Kubernetes.

Вы можете найти подробную информацию о etcd в [официальной документации](#).

Архитектура k8s: kube-controller-manager

— компонент k8s, который запускает процессы контроллера:

- **Контроллер узла (Node Controller):** уведомляет и реагирует на сбой узла.
- **Контроллер репликации (Replication Controller):** поддерживает правильное количество подов для каждого объекта контроллера репликации в системе.
- **Контроллер конечных точек (Endpoints Controller):** заполняет объект конечных точек (Endpoints), то есть связывает сервисы (Services) и поды (Pods).
- **Контроллеры учетных записей и токенов (Account & Token Controllers):** создают стандартные учетные записи и токены доступа API для новых пространств имен.



Архитектура k8s: kube-scheduler

— компонент k8s, который отслеживает созданные поды без привязанного узла и выбирает узел, на котором они должны работать.

При планировании развертывания подов на узлах, учитывается множество факторов, включая требования к ресурсам, ограничения, связанные с аппаратными/программными политиками, принадлежности (affinity) и непринадлежности (anti-affinity) узлов/подов, местонахождения данных, предельных сроков.



Архитектура k8s: kube-proxy

— сетевой прокси, работающий на каждом узле в кластере, и реализующий часть концепции сервис.

kube-proxy конфигурирует правила сети на узлах. При помощи них разрешаются сетевые подключения к вашим подам изнутри и снаружи кластера.

kube-proxy использует уровень фильтрации пакетов в операционной системе, если он доступен. В противном случае, kube-proxy сам обрабатывает передачу сетевого трафика.



Архитектура k8s: kubelet

— агент, работающий на каждом узле в кластера. Он следит за тем, чтобы контейнеры были запущены в поде.

Утилита kubelet принимает набор PodSpecs, и гарантирует работоспособность и исправность определенных в них контейнеров. Агент kubelet не отвечает за контейнеры, не созданные Kubernetes.



Сетевые плагины



CNI


Для реализации сетевого взаимодействия используются сетевые плагины, или другими словами CNI.

CNI (Container Network interface) представляет собой спецификацию для организации универсального сетевого решения для Linux-контейнеров. Кроме того, он включает в себя плагины, отвечающие за различные функции при настройке сети пода. Плагин CNI — это исполняемый файл, соответствующий спецификации.

Плагины CNI

Одной из главных ценностей CNI, конечно же, являются сторонние плагины, обеспечивающие поддержку различных современных решений для Linux-контейнеров. Среди них:

- Project Calico;
- Weave;
- Flannel.



Установка с помощью kubeadm

Установка kubectl, kubelet, kubeadm

Устанавливаем зависимости для apt:

```
sudo apt-get update  
sudo apt-get install -y apt-transport-https ca-certificates curl
```

Настраиваем репозиторий:

```
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg  
https://packages.cloud.google.com/apt/doc/apt-key.gpg  
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
```

Устанавливаем пакеты:

```
sudo apt-get update  
sudo apt-get install -y kubelet kubeadm kubectl
```

Создание кластера

Инициализируем первую мастер ноду:

```
kubeadm init
```

Если все прошло успешно, то в консоль выведется команда для присоединения нод к кластеру, выполняем ее:

```
kubeadm join <control-plane-host>:<control-plane-port> --token <token>  
--discovery-token-ca-cert-hash sha256:<hash>
```

Устанавливаем CNI (Например weave):

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl  
version | base64 | tr -d '\n')"
```



Helm

Helm: обзор и установка

Helm — пакетный менеджер k8s. Позволяет объединять несколько yaml файлов и шаблонизировать их с помощью встроенных функций и параметров. Такой пакет называется Chart.

Имеет функционал отката релиза, тестирования, хранит текущее состояние релиза внутри кластера.

Установка Helm производится с помощью скрипта установки:

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 |  
bash
```

Helm: деплой проекта

Helm можно использовать как для установки чартов из публичных репозиториев:

```
helm repo add brigade https://brigadecore.github.io/charts  
helm install brigade-release brigade/brigade
```

Так и для создания и деплоя своих собственных чартов:

```
helm install my-project
```

Helm: создание собственных чартов

Структура в директории с чартом должна иметь вид:

```
mychart/  
  Chart.yaml  
  values.yaml  
  templates/
```

Chart.yaml содержит метаданные о самом чарте (название, версия, автор и.т.д).

values.yaml — значения для установки по умолчанию.

templates/*.yaml — те объекты, которые будут загружены при установке чарта.

Helm: пример Chart.yaml

apiVersion — указывает версию api для helm.

name — название чарта, именно оно и будет фигурировать в поиске при загрузке чарта в репозиторий;

version — версия чарта;

appVersion — версия приложения внутри чарта (они не всегда могут совпадать)

```
apiVersion: v2
name: frontend-chart
description: A Helm chart for
Kubernetes
type: application
version: 0.1.0
appVersion: 0.1.0
```

Helm: пример values.yaml

В этом файле хранятся любые параметры, которые потребуется переопределять в разных инсталляциях.

Хорошим тоном в публичных чартах считается иметь работоспособные параметры по умолчанию.

```
replicas: 2
database
  host: localhost
  port: 5432
users:
  - root
  - test
  - web
```


Helm: пример шаблона yaml файла

Создадим configmap.yaml с подобным содержанием.

Helm имеет встроенные объекты, такие как Release.Name. Полный список есть в официальной документации.

Использование параметров из values.yaml возможно с помощью {{ .Values. }}

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  test: {{ .Values.replicas | quote }}
```

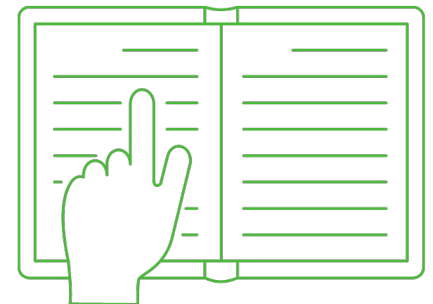


Итоги

Итоги

Сегодня мы рассмотрели архитектуру kubernetes, kubeadm, helm и теперь:

- знаем, как устроен kubernetes;
- можем установить кластер с помощью kubeadm;
- умеем использовать helm для создания и установки чартов.





Домашнее задание

Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Петр Шило

 [Петр Шило](#)