Работа с контейнерным оркестратором Kubernetes на базе платформы DE

Преподаватель

Игнатенко Филипп

Руководитель разработки PaaS-сервисов российской облачной платформы (БАЗИС)

Преподаватель на курсах DevOps, DevSecOps, Docker, Kubernetes, Linux на платформе онлайн-образования «Otus»

Спикер международных конференций

Scan me! Ignatenko Filipp

Маршрут вебинара

Сравнительный анализ известных оркестраторов

Основные компоненты кластера Kubernetes

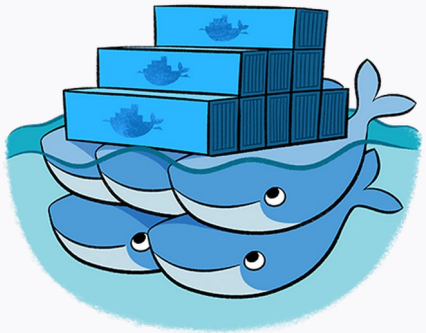
Docker swarm

Docker swarm

Docker swarm — встроенное в docker решение по контейнерной оркестрации

Работает исключительно с **docker**

Нативное и **более легкое** в освоении, чем k8s



Docker swarm. Устройство

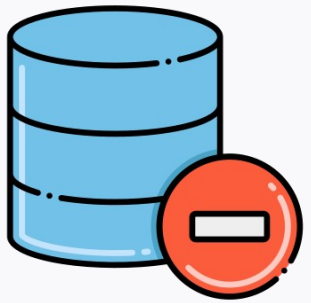
Основу кластера docker swarm представлют управляющие (**master**) и рабочие (**worker**) узлы

(для сравнения: k8s состоит из множества компонентов: etcd, api-server, scheduler, controller-manager и т.д, что труднее в установке и администрировании)

Docker swarm. Ограничения.

Ограничение по количеству узлов и контейнеров:

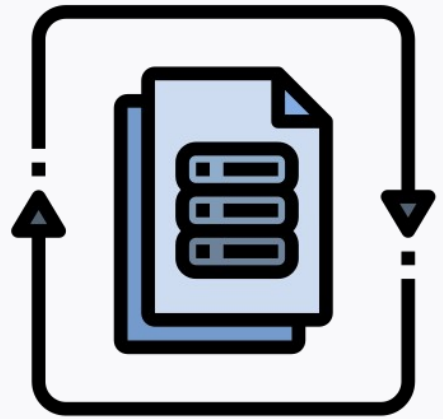
masters: 9 штук*
containers: 100 на 1 ноду



Docker swarm. Шаблонизаторы

Манифесты и шаблонизаторы:

Docker swarm поддерживает **YAML**-формат
И **не поддерживает шаблонизаторы** (такие, как helm в k8s)



Docker swarm. Сети

В docker swarm существует поддержка следующих сетей:

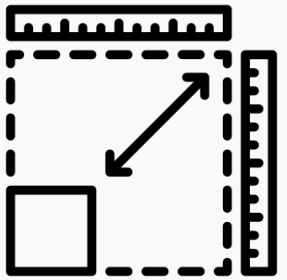
- **overlay** (связь между разными docker daemon)
- **ingress network** (overlay с возможностью балансировки)
- **bridge** (brodge-сеть для соединения overlay'ных сетей)

В docker swarm есть **поддержка service-discovering**’а

Docker swarm. Автомасштаиброване

Автомасштабирование в docker swarm **отсутствует** (в отличии от k8s)

Зато, есть **поддержка rolling update** (постепенное обновление сервисов, как в k8s)



Docker swarm. Мониторинг

С помощью дополнительных инструментов и настроек в docker swarm возможна настройка **мониторинга** и **логирования** (prometheus, grafana, ELK)

Docker swarm. Безопасность

Для хранения секретов и паролей есть возможность интеграции HashiCrop **Vault** (как замена встроенному решению по управлению секретами — **docker secrets**)

Есть поддеркжа **mTLS** (взаимное TLS)

Сетевые политики (network policy) **не поддерживаются**

Docker swarm

Итог: Docker swarm. Преимущества и недостатки

Преимущества:
- низкий порог вхождения, простота использования
- только контейнерная оркестрация на основе docker

Недостатки:
- Уступает k8s в широте администрирования
- Отсутствует автомасштабирование
- функционал сильно уступает k8s (если не хочется придумывать и/или использовать костыли)

HashiCorp Nomad

HashiCorp Nomad

HashiCorp Nomad

Nomad — оркестратор от **HashiCorp**, который правильнее назвать фреймворком для построения кластерных решений

Существует **community** и **enterprise** версии

HashiCorp Nomad. Основной концепт

Nomad представляет из себя **фреймворк** для построения кластера, расширяемый за счет использования внешних инструментов (Consul для service-discovering, Vault для секретов, Prometheus для мониторинга, ELK, и так далее)

Установка и управление Nomad **заметно легче**, чем в k8s. Имеет относительно **низкий порог вхождения** (минимальная установка может представлять из себя один бинарь, или набор пакетов для различных ОС)

HashiCorp Nomad. Устройство

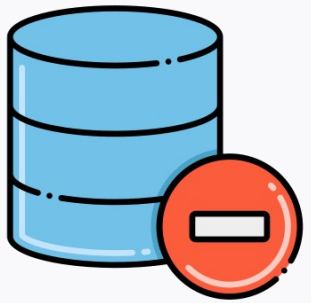
Кластер Nomad представлен из управляющих нод (**server**) и рабочих нод (**clients**)

Использует принцип **Job - Tasks**, где task - атомарная единица работы, выполняемая драйверами (docker, QEMU, Java и тд).

Таким образом, Nomad поддерживает виртуализированные, контейнеризированные, пакетные и другие виды нагрузок)

HashiCorp Nomad. Ограничения

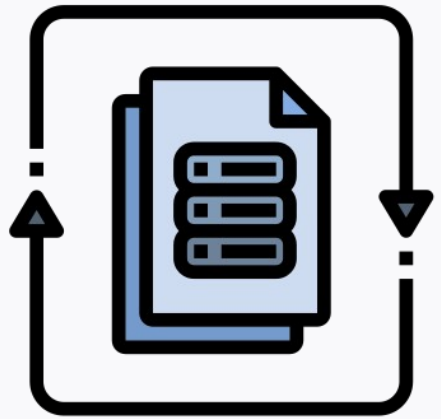
Ограничения:
Количество узлов: 10 000
Количество контейнеров: 2 млн



HashiCorp Nomad. Язык манифестов и шаблонизаторы

Nomad **не использует YAML** (используется собственный язык **HCL**, однако есть конверторы в YAML, JSON (и обратно), но есть ограничения)

Нет поддержки шаблонизаторов (таких, как helm в k8s)



HashiCorp Nomad. Сети и Storage

Nomad по умолчанию поддерживает **хостовые сети**, но есть **поддеркжа CNI** (знакомые по k8s: calico, weaave, cilium)

Также поддерживается **CSI** (Container storage interface), как в k8s

Service discovering на основе DNS **отсутствует** (в отличии от swarm и k8s), однако за счет расширений (Consul) становится **возможным**

HashiCorp Nomad

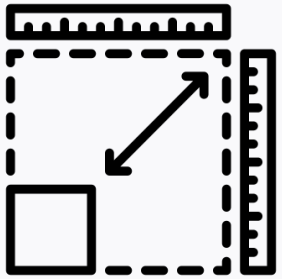
HashiCorp Nomad. Автомасштабирование и обновление рабочих нагрузок

Автомасштабирование на основе **HPA** (кол-во подов) и **cluster autoscaler** (кол-во нод).
Отсутствует VPA (объем ресурсов для подов) — оно есть только в **enterprise-**версии

Поддержка **rolling update, blue-green и canary** обновлений с сохранением истории (как в k8s)

Nomad успешно участвовал в стресс-тестах на масштабируемость: 2 миллиона контейнеров

Производительность Nomad сравнима с k8s



HashiCorp Nomad

Итог: HashiCorp Nomad. Преимущества и недостатки

Преимущества:
- простота в установке и администрировании
- достойный вендор (Hashicorp)
- поддеривает как контейнерные, так и не-контейнерные нагрузки

Недостатки:
- из коробки, в "голом" виде сильно ограничен функционал. Требуется установка и интеграция сторонних решений
- функционал бесплатной версии уступает функционалу enterprise-решения
- комьюнити слабее, чем у k8s

Apache Mesos

Apache Mesos

Apache Mesos (Aurora, Maraphon, DS/OS)

Apache Mesos - менеджер кластера с поддержкой различных типов нагрузок

Apache MESOS™

Apache Mesos. Устройство

Кластер Mesos состоит из управляющих нод (**master**), рабочих нод (**agent**), и «фреймворков» (**frameworks**).

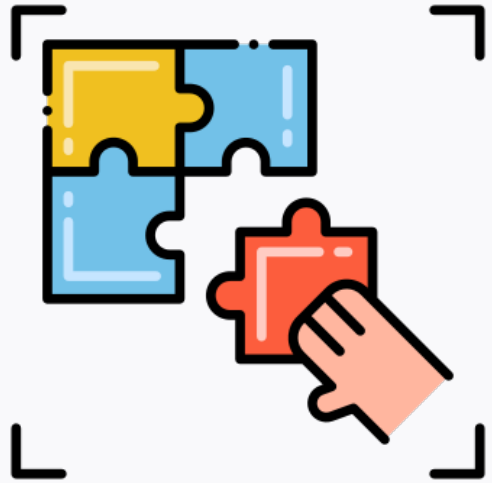
Фреймворков может быть одновременно несколько, и именно фреймворки запускают рабочие нагрузки.

Выбором лидера среди управляющих master-нод занимается инструмент **ZooKeeper**.

Apache Mesos. Фреймворки

Фреймворк состоит из 2-х компонентов:
- **планировщик** (**Scheduler**) - устанавливается на каждой ноде
- **исполнитель** (**Executor**) - запускается на рабочих нодах (agent) и выполняет задачи фрейморка

Master предлагает доступные ресурсы **agent**'ов имеющимся **фреймворкам**, а **scheduler**'ы фреймворков **выбирают**, какие из предложенных ресурсов агентов можно использовать для запуска рабочих нагрузок



Apache Mesos. Популярные фреймворки

Apache Mesos - это главный менеджер кластера, а конечная функциональность по запуску рабочих нагрузок определяются установленными **фреймворками**.

Самые популярные фреймворки:
- **Maraphon** (управление контейнерами)
- **Apache Aurora** (legacy, deprecated!) для cron-заданий различных служб
- **Mesospere DC/OS** (ОС на основе Apache Mesos+Maraphon - legacy, deprecated!) для расширения ряда возможностей, которые частично доступны в бесплатной версии. (Прекратил развитие в пользу создания k8s-сместимого фреймворка)

Apache Mesos. Виды поддерживаемых нагрузок

Mesos поддерживает как контейнерные, так и не-контейнерные нагрузки

В качестве **Scheduler** (компонент фреймворка) могут быть использованы:
- **Hadoop** (для раочих нагрузок big data)
- **MPI** (Message Passing Interface) для систем обмена сообщениями
- **Jenkins** - для заданий CI/CD
- возможность разработать **собственный** scheduler

Apache Mesos. Производительность

Развернуть Apache Mesos легко, но дальнейшая поддержка и администрирование связано с высокими накладными расходами по работе с **ZooKeeper**, администрированием Java (тот же hadoop от apache, написанный на Java), поддержке фреймворков

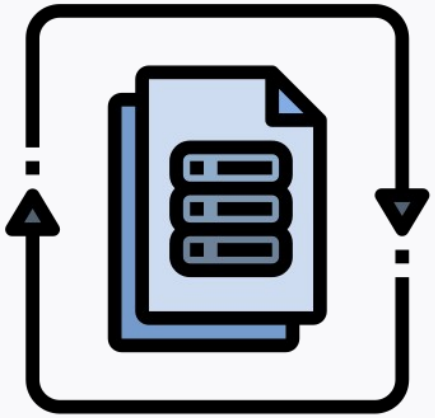
По производительности не уступает k8s, за счет ориентированности на big data (в основном используется для работы с фреймворками, типа hadoop)

Apache Mesos поддерживает несколько десяток тысяч нод.
При использовании **Mesosphere DC/OS** как фреймоврка, получаем практически **неограниченную** масштабируемость (~50000 нод).
Однако, разработка Mesosphere DC/OS **прекращена** и был взят курс на интеграцию с Kubernetes)

Apache Mesos. Язык манифестов и шаблонизаторы

Apache Mesos использует **JSON**-нотацию для описания манифестов

Имеется поддержка **проприетарных шаблонизаторов** для Mesosphere DC/OS (благодаря инструменту Maraphon-LB) и Apache Aurora



Apache Mesos. Сети

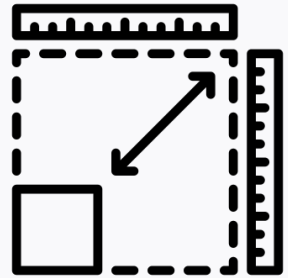
В Apache Mesos имеется поддержка **CNI** (calico, cilium, weave)

service-discovering доступен за счет собственного решения Mesos-DNS (тесно связан с функциональностью ZooKeeper)

При использовании **Maraphon-LB** возможно обнаружение портов с помощью HAProxy.
При использовании **Apache Aurora** доступна поддержка инструмента Mesos DiscoveryInfo (отвязывает нас от ZooKeeper)

Apache Mesos

Apache Mesos. Автомасштабирование, обновление и наблюдаемость



Автомасштабирование **доступно** в комбинации **с Mesosphere DC/OS**

Поддерживается **Blue-green** обновление

За счет распределенной системы управления (**masters - agents - frameworks**) и централизованного управления **ZooKeeper** обеспечивается высокая доступность и надежность.

При использовании Mesosphere DC/OS появляется возможность выполнять **health-check**

Есть поддержка ELK, Prometheus + Grafana, но довольно нетривиально в установке и настройке

Apache Mesos. Безопасность

Apache Mesos + Maraphon, Apache Mesos + Apache Aurora требуют дополнительных средств обеспечения информационной безопасности, таких как Hashicorp **Vault** (встроенное решение по работе с секретами осталось недореализованным)

Лучше дела обстоят при использовании Apache Mesos + DC/OS (но только в платной **enterprise-**версии, которая к тому же и не развивается)

Apache Mesos

Итоги по Apache Mesos. Преимущества и недостатки

Достоинства:
- Высокая масштабируемость
- Поддержка как контейнерных, так и не-контейнерных рабочих нагрузок
- Возможность совместного использования нескольких фреймворков

Недостатки:
- Сложность в поддержке и администрировании
- Всё сложно с разработкой и поддержкой фреймворков
- Высокий порог вхождения

Kubernetes

Kubernetes

Kubernetes (**k8s**) - самодостаточный инструмент контейнерной оркестрации со множеством встроенных сервисов.

Разрабатывался **Google** и передан на поддержку в фонд **CNCF**. Обладает впечатляющим комьюнити. **Полностью бесплатен**.

Де-факто **стандарт индустрии** по контейнерной оркестрации.

Kubernetes. Устройство и виды нагрузок

K8s работает **только с контейнерными нагрузками**, но поддерживает **не только docker**

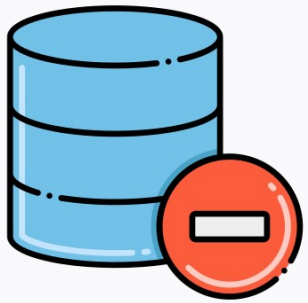
Высокая производительность, распределенная и отказустойчивая система развертывания за счет использования управляющих (**master**) нод и рабочих (**worker**) нод.

Имеет множество компонентов, изучить которые необходимо - по этой причине, k8s имеет относительно **высокий порог вхождения**

Kubernetes. Ограничения

K8s имеет **ограничения** по количеству нод и контейнеров:

- не более 110 подов на узел, при этом параметр можно настраивать,
- не более 5 000 узлов,
- всего не более 150 000 подов,
- всего не более 300 000 контейнеров



Kubernetes. Основные преимущества

Поддеркжа **YAML** и **JSON** форматов

Поддержка мощного шаблонизатора **HELM**

Поддержка множества **CNI**-плагинов (calico, flannel, weave, cilium и проч)

Поддержка **rolling update, blue-green и canary** обновлений, с сохранением истории

Поддержка автомасштабирования **кластера, HPA, VPA**

Возможность подключения систем **мониторинга**, **трасировки**, **логирования** и проч.

Поддержка **service-discovering** за счет встроенного DNS-сервиса

Высокий уровень безопасности, обеспеченный внутренними сервисами с возможностью расширения

Итоги по Kubernetes. Преимущества и недостатки

Достоинства:
- Самодостаточный инструмент для контейнерной оркестрации с возможностью расширения
- Работает не только с docker (CRI)
- Огромное неравнодушное комьюнити
- Полностью бесплатен

Недостатки:
- Высокий порог вхождения
- Предназначен только для контейнерной оркестрации
- Имеет ограничения по количеству узлов и подов

Применимость

Применимость

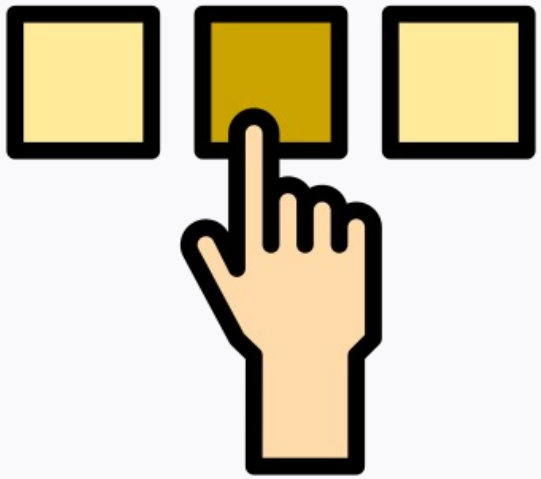
Для каких задач подходят рассмотренные оркестраторы

Docker swarm:
- для маленьких проектов с небольшим кластером
- для небольших команд

Hashicorp Nomad:
- для сочетания контейнерных и не-контейнерных нагрузок
- для небольших команд
- для несложных сетевых архитектур

Apache Mesos:
- для сложных распределенных масштабируемых систем
- для работы с big data
- для сочетания контейнерных и не-контейнерных нагрузок
- для знающих, погруженных и опытных команд

k8s:
- для контейнерной оркестрации
- для распределенных масштабируемых систем
- для больших и небольших команд
- стандарт индустрии де-факто



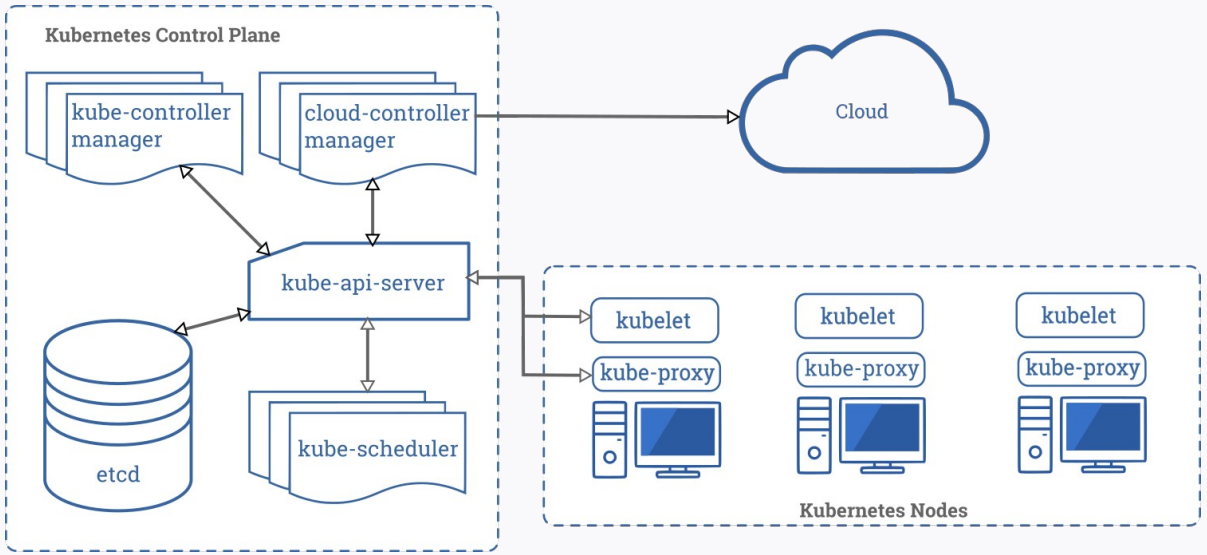
Основные компоненты кластера k8s

Основные компоненты кластера k8s

Из каких компонентов состоит кластер k8s?

Основные компоненты:

etcd
api-server
controller-manager
scheduler



Основные компоненты кластера k8s

etcd

etcd – key/value база данных для хранения конфигурации кластера

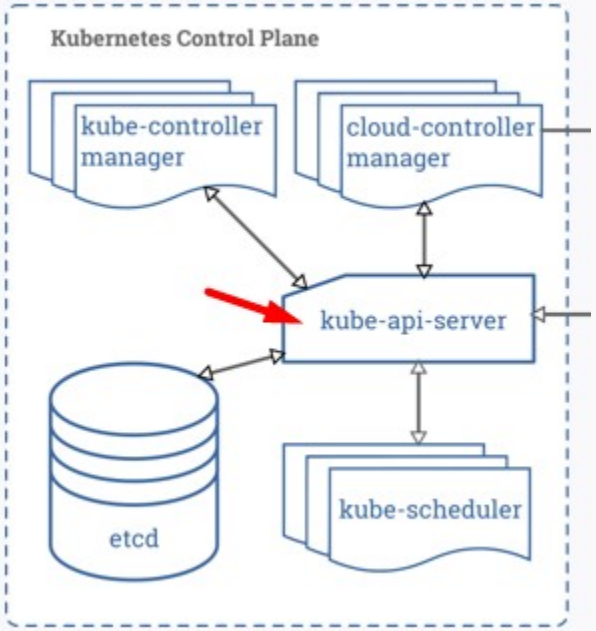
- Работает по алгоритму **raft** (он обеспечивает надежность за счет поддержки кворума)
- Единственная база данных для хранения конфигурации, которую поддерживает k8s
- Единственный stateful-компонент
- На каждую master-ноду устанавливается по ноде **etcd**

Основные компоненты кластера k8s

api-server

api-server – центральный, главный компонент k8s

- Stateless (в отличии от etcd)
- Взаимодействие через kubectl (но можно работать и просто curl’ом)
- Единственный компонент, который общается с etcd
- Работает по REST API
- Обеспечивает авторизацию и аутентификацию (разграничение прав доступа до содержимому кластера)



*При развертывании нового сервиса (например, kubectl apply) мы обращаемся именно к **api-server**, а он уже записывает данные в etcd*

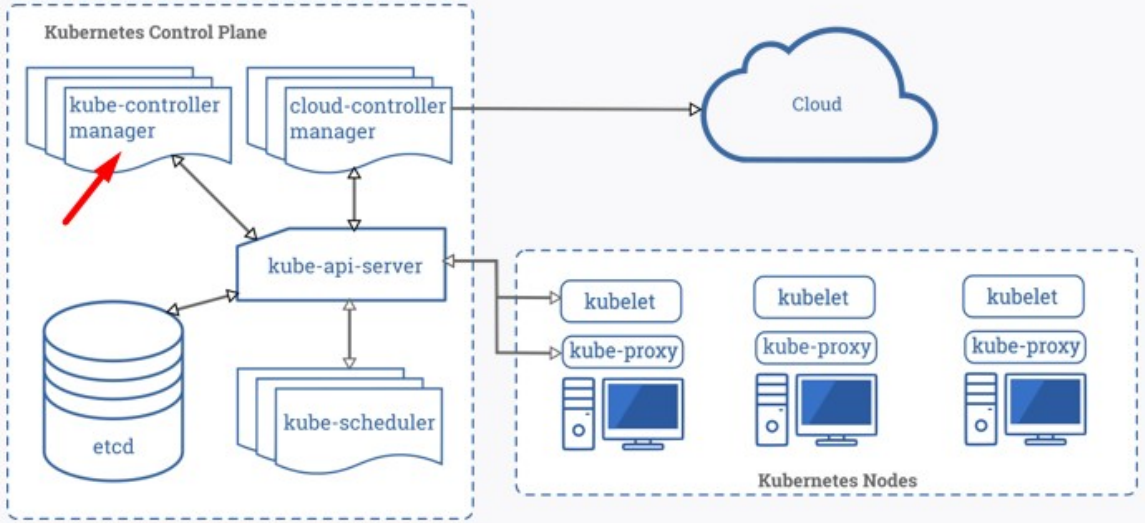
Основные компоненты кластера k8s

controller-manager

controller-manager – запускает процессы набора контроллеров

В состав controller-manager’а входят следующие контроллеры:

- node-controller
- replicaset-controller
- endpoints-controller
- account-controller
- token-controller



Основные компоненты кластера k8s

controller-manager (Подробнее о его наборе контроллеров)

- **node controller** - держит связь с нодами кластера, если нода не отвечает, переносит нагрузки на другие ноды

*kubelet общается с api server, **node controller** через controller manager общается тоже с api server, но (!) не с самим kubelet напрямую*

- **replicaset controller** - смотрит в api-server кластера, видит созданные replicaset’ы, реализует процедуру их создания
- **endpoints controller** - автоматизация создания эндпоинтов для сервисов (связь между подом и его сервисом)
- **account controller** - создание стандартных учетных записей
- **token controller** - создание токенов для доступа API

Основные компоненты кластера k8s

controller-manager. Дополнительные функции

controller-manager – помимо запуска процессов из набора своих контроллеров, также занимается сборкой мусора в кластере

Garbage Collector, **GC** (сборщик мусора) - ищет и удаляет мусор, который в кластере больше не нужен (например, старые реплики, которые превышают установленный лимит)

Основные компоненты кластера k8s

controller-manager. Алгоритм lease

Обычно, на каждую мастер-ноду устанавливают по одному экземпляру controller-manager.
Но, одновременно может работает только один лидирующий controller-manager, и выбор лидера происходит по алгоритму **lease**

Суть алгоритма **lease** очень проста – тот, кто первым успеет записать себя лидером в api-server тот и лидер

Если текущий лидер не будет отвечать нужное время, борьба за лидерство возобновится по тому же принципу

Основные компоненты кластера k8s

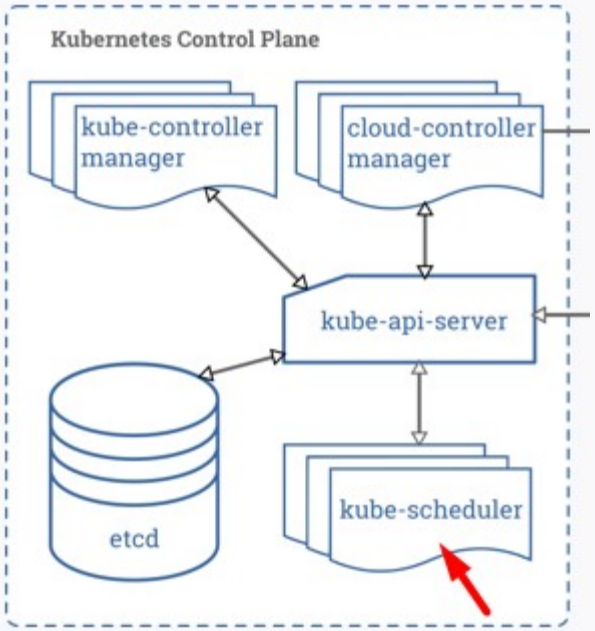
scheduler

scheduler назначает поды на ноды с учетом множества факторов

controller-manager генерирует манифесты подов, записывает данные в api-server, а **scheduler** назначает их на ноды, но учитывает важные параметры:

- QoS (quality of service)
- Affinity и Anti-affinity
- Requests и Limits
- Priority Class (preemption)

scheduler обычно запускают по одному на каждой мастер-ноде (**lease**)



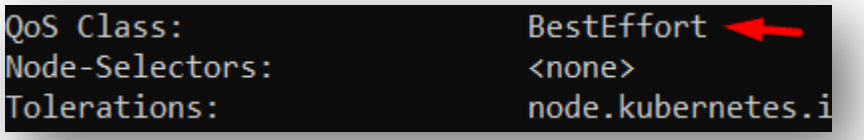
Основные компоненты кластера k8s

scheduler. Подробнее о политике выбора нод для подов

Поды могут попадать под один из трёх Quality of service (**QoS**):

- **Guaranteed**: поды, для которых указаны реквесты и лимиты для всех контейнеров, и для всех они одинаковы
- **Burstable**: не гарантированные поды, для которых задан реквест как минимум на CPU или память для одного из контейнеров
- **BestEffort**: поды без реквестов и лимитов вообще

Проверить QoS класс пода можно через kubectl describe pod



Основные компоненты кластера k8s

scheduler. Подробнее о политике выбора нод для подов

Affinity/anti-affinity (node-selector)

Раздел спецификации «**nodeSelector**» предоставляет очень простой способ привязать модули к узлам с определенными метками.

Функция **affinity/anti-affinity** значительно расширяет типы ограничений, которые вы можете выразить.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node5
```

Основные компоненты кластера k8s

scheduler. Подробнее о политике выбора нод для подов

requests/limits

В некоторых сценариях использования k8s вам может понадобиться явное указание реквестов и лимитов ресурсов для ваших сервисов.

Сделать это можно с помощью соответствующих разделов в манифесте – **requests/limits**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: cpu-demo
  namespace: cpu-example
spec:
  containers:
  - name: cpu-demo-ctr
    image: vish/stress
    resources:
      limits:
        cpu: "1"
      requests:
        cpu: "0.5"
    args:
    - -cpus
    - "2"
```

Основные компоненты кластера k8s

scheduler. Подробнее о политике выбора нод для подов

priority class / preemption (вытеснение)

Поды могут иметь приоритет. Приоритет отображает важность пода относительно других подов в кластере

Если под не может быть запущен на подходящем узле из-за нехватки ресурсов, то **scheduler** пытается “вытеснить” поды с более низким приоритетом и переместить их на другие узлы кластера, чтобы освободить ресурсы и запустить ожидающий под

В PriorityClass добавлено поле **PreemptionPolicy**,
что позволяет подам данного класса приоритета
“вытеснять” поды с более низким классом приоритета

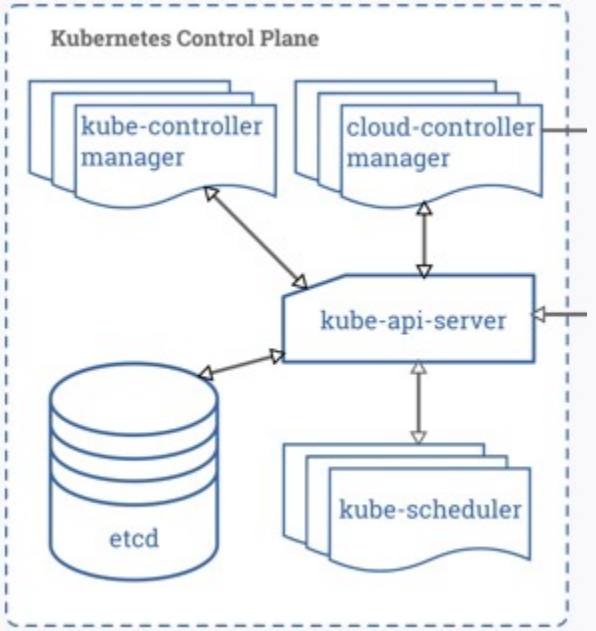
```
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: high-priority
```

Основные компоненты кластера k8s

Итог - из каких компонентов состоит кластер k8s?

Основные компоненты k8s-кластера:

etcd – хранилище конфигурации
api-server – основной компонент API
controller-manager – запуск набора контроллеров и сборка мусора
scheduler – назначение подов на нод с учетом множества факторов

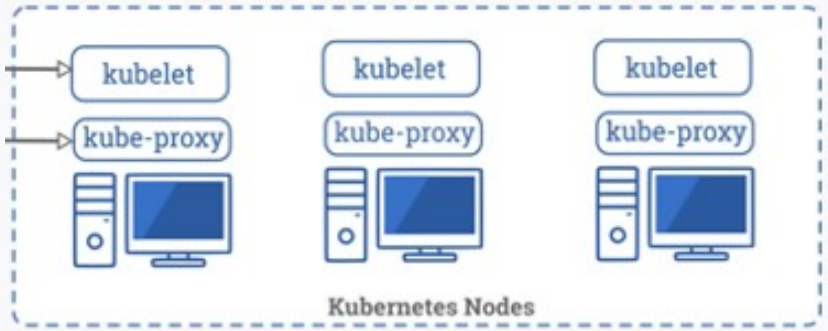


Дополнительные компоненты кластера k8s

Дополнительные компоненты кластера k8s

Помимо основных компонентов, установленных на мастер-нодах, для работы кластера необходимы дополнительные компоненты, которые управляются на всех нодах (мастеры и воркеры):

- kubelet
- kube-proxy



Дополнительные компоненты кластера k8s

kubelet

kubelet – агент, работающий на узле кластера

- Работает на каждой ноде (и мастеры и воркеры)
- Не запускается в докере, работает как процесс на хосте (systemctl status kubelet)
- Отдает команды docker daemon через docker api (docker run, напр.)
- фактически реализует запуск подов на узле
- Обеспечивает проверки liveness probe, readiness probe, startup probe

Дополнительные компоненты кластера k8s

kubelet. Подробнее о probe

Liveness probe – используется для определения, когда контейнер необходимо перезапустить

Readiness probe – используется для проверки, доступен ли модуль в течение всего жизненного цикла. В отличие от liveness probe, в случае сбоя проверки останавливается только трафик к модулю , но перезапуска не происходит

Startup probe – используется для проверки, что приложение в контейнере было запущено. Если проба настроена, то liveness и readiness проверки блокируются, до того как проба пройдет успешно

```yaml
livenessProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 1
  periodSeconds: 10
```

```yaml
readinessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

```yaml
startupProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 30
  periodSeconds: 10
```

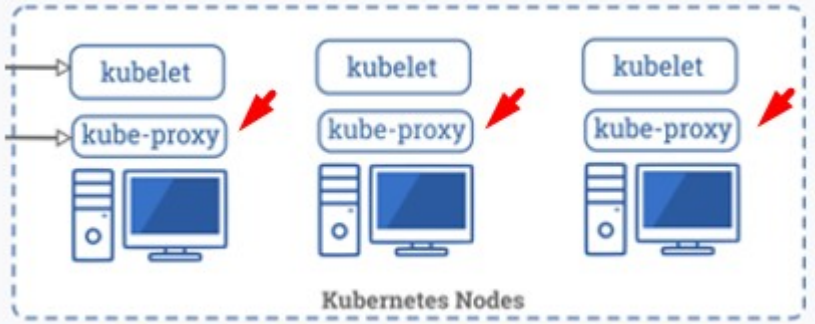
Дополнительные компоненты кластера k8s

kube-proxy

kube-proxy – сетевой прокси, работающий на каждом узле в кластере

- Взаимодействует с api-server
- Устанавливается на всех нодах
- Управляет сетевыми правилами на нодах
- Запускается в контейнере

*Некоторые cni-плагины забирают работу **kube-proxy** себе*



Дополнительные компоненты кластера k8s

Второстепенные компоненты, с которыми вы столкнетесь при работе с кластером k8s:

- **cri** (движок процесса контейнеризации)
- **cni** (сетевые плагины)
- **dns** (k8s-совместимые dns-серверы)
- **ccm** (controller-manager для облачных решений)

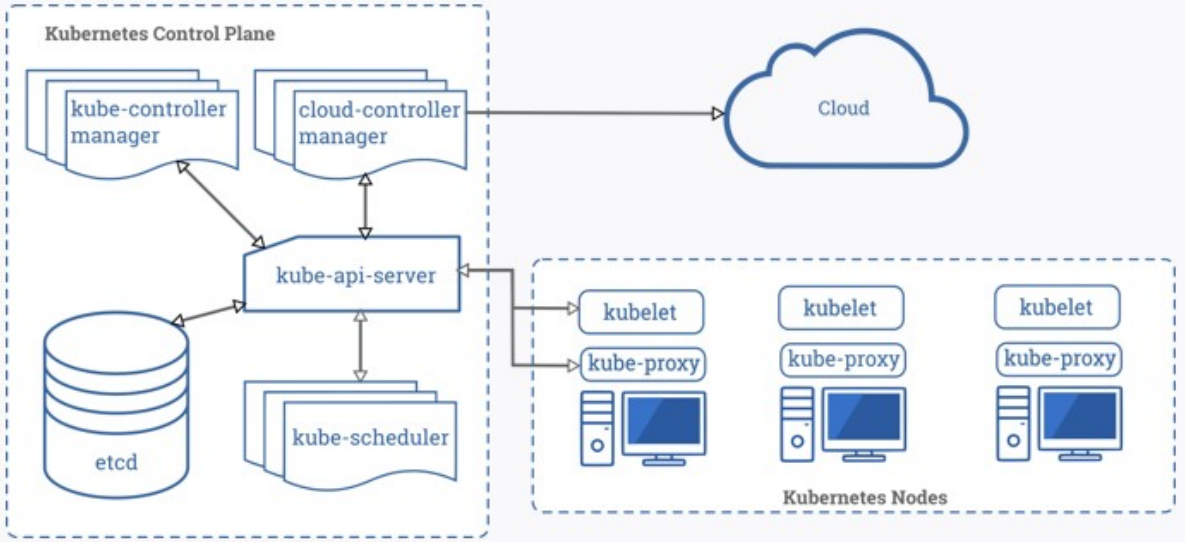
Дополнительные компоненты кластера k8s

Подведем итог

Для работоспособности кластера необходимы следующие компоненты:

На мастер-нодах:
- **etcd**
- **api-server**
- **controller manager**
- **scheduler**

На мастер- и воркер-нодах:
- **kubelet**
- **kube-proxy**



При развертывании кластера вы можете также столкнуться и с другими неосновными компонентами: **cri**, **cni, dns, ccm**

Подробнее о etcd

Основные компоненты кластера k8s

etcd

etcd – key/value база данных для хранения конфигурации кластера

- Работает по алгоритму **raft** (он обеспечивает надежность за счет поддержки кворума)
- Единственная база данных для хранения конфигурации, которую поддерживает k8s
- Единственный stateful-компонент
- На каждую master-ноду устанавливается по ноде **etcd**

Основные компоненты кластера k8s

etcd. Подробнее о raft

raft – алгоритм решения консенсуса в сети надежных вычислений

Каждая нода может находиться в одном из 3-х состояний:
- Лидер (Leader)
- Кандидат (Candidate)
- Ведомый (Follower)

Управление кластером разделено на две фазы:
- Выборы лидера (leader election)
- Репликация протокола (log replication)


LIVE

Основные компоненты кластера k8s

etcd в k8s

Для достижения консенсуса, в кластере k8s обычно развертывают **нечетное количество** (чаще всего 3 или 5) мастер-нод с установленным **etcd** на каждую из них (для обеспечения кворума при работе с фазами **raft-**протокола: выбора лидера и репликация протокола)

И помните, что **etcd** может быть применен не только в кластере k8s. Знать и понимать принципы его работы важно, и может пригодиться в будущем..


LIVE

Спасибо за внимание!

Игнатенко Филипп

Руководитель разработки PaaS-сервисов российской облачной платформы (БАЗИС)

Преподаватель на курсах DevOps, DevSecOps, Docker, Kubernetes, Linux на платформе онлайн-образования «Otus»

Спикер международных конференций

Scan me! Ignatenko Filipp