SkillFactory

ПРОФЕССИЯ

DevOps-инженер

КОНСПЕКТ

МОДУЛЬ D4. ОСНОВЫ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ В KUBERNETES

D4.1 Основы работы с сетью в Kubernetes

СЕТЕВЫЕ ПЛАГИНЫ

Основная, фундаментальная концепция для всех сетевых провайдеров в ***Kubernetes*** имеет **несколько правил:**

- Все поды могут связываться со всеми подами на всех хостах кластера без использования технологии ***NAT***.
- Агенты на хостах (***kubelet***, ***system daemons***) могут связываться со всеми подами на том хосте, где находятся сами.

Чтобы добиться уникальности ***IP***-адресов, для каждого хоста в кластере ***Kubernetes*** используется своя подсеть ***IP***-адресов, что гарантирует уникальность и в то же время накладывает ограничение на число подов.

SERVICE

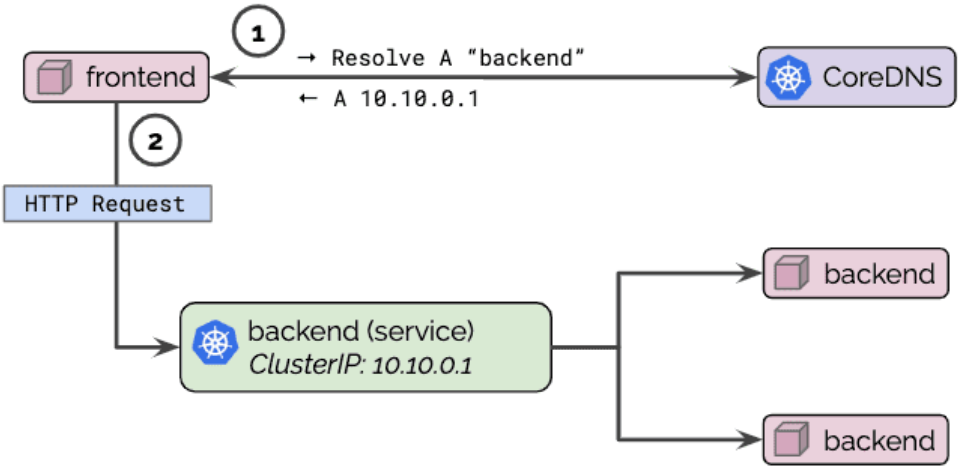
Service создан как логическая связь между подом и политикой доступа к нему. С помощью определённых правил, таких как выбор пода через метки (***Labels***) и балансировка ***DNS Round-Robin***, сервисы обеспечивают доступность динамически создающихся подов через статичные точки входа (***endpoints***).

Другими словами, именно ***Service*** помогает нам не думать про динамически появляющиеся ***IP***-адреса и даёт возможность использовать внутренние ***DNS***-имена.

Сервисы бывают двух типов:

1. **Service**

   Имеет собственный ***IP***-адрес, создаёт ***Endpoint*** и ***A***-запись в ***DNS*** на себя, маршрутизирует запросы по очереди, выбирая доступный под, подходящий под правила обнаружения.



2. **Headless Service**

   Не имеет ***IP***-адреса, поэтому создаёт ***Endpoint*** и ***CNAME***-запись в ***DNS*** для всех ***IP*** подов, подходящих под правила обнаружения.

   При обращении к такому сервису мы будем получать каждый раз список из ***IP***-адресов всех обнаруженных подов.

Сервисы позволяют выбрать один из **четырёх режимов работы:**

1. **ClusterIP (режим работы по умолчанию)**

   Делает сервисы доступными в пределах кластера ***Kubernetes***.

   С помощью правил ***IPtables*** настраиваются слушатели на портах, указанных в ***port***, и перенаправляется трафик в найденные, согласно правилам ***selector***, поды на ***targetPort*** при использовании ***protocol***, указанного в манифесте к порту.

   Имена портов работают как алиасы, чтобы можно было обращаться к ним по имени, например ***targetPort https***.

2. **NodePort**

   Позволяет опубликовать сервис на всех нодах по определённому порту (доступен диапазон портов 30000-32767, порт может быть выбран автоматически, но должен быть уникальным в пределах кластера).

   Настраивает правила маршрутизации, чтобы сервис был доступен по *<NodeIP>:<NodePort>*. Но если обратиться на этот же порт к другому хосту кластера, сервис ***kubeproxy*** маршрутизирует запрос на нужный хост, где опубликован данный сервис. Внутри кластера по-прежнему можно использовать ***port*** и имя сервиса.

3. **LoadBalancer**

   Позволяет опубликовать сервис через балансировщики сетевой нагрузки, такие как ***Haproxy*** или ***Application Load Balancer (AWS)***. Динамически открывает порты, как и в случае с ***NodePort***, но не настраивает правила внешней маршрутизации, а ожидает создания балансировщика и делегирует ему управление трафиком.

4. **ExternalName**

   Создаёт запись в ***DNS*** вида ***CNAME***, не настраивает правила внешней маршрутизации. Может быть использован как заглушка или для более сложной маршрутизации к внешним источникам через балансировщики или ***Ingress***.

INGRESS

Ingress предоставляет маршрутизацию седьмого уровня ***HTTP*** и ***HTTPS*** (а более продвинутые — и ***L4 TCP*** и ***UPD***) извне кластера к сервисам кластера.

- В ***Rules*** описываются правила маршрутизации к сервисам: на какой сервис, при каком условии пойдёт трафик, как отвечать на тот или иной тип запроса.

- ***Host*** содержит доменное имя, на которое будет отвечать данная конфигурация. Для одного ***Ingress***-контроллера все имена должны быть уникальными и может существовать лишь один ***wildcard*** "*" . В противном случае либо контроллер будет просто игнорировать дубликат, либо откажется работать и упадёт с ошибкой.

- ***Paths*** описывает пути, которые идут за ***host***, и правила маршрутизации (в какой сервис будет перенаправляться трафик).

- ***PathType*** описывает, как обрабатывать путь.

Есть три вида ***PathType***:

- ***Exact*** требует точного совпадения имени) : ***/example*** совпадает, ***/example/*** — уже нет.
- ***Prefix*** требует точного совпадения в начале строки: ***/example/aa*** будет совпадать с ***/example/aa*** и ***/example/aa/bbb***, но не будет совпадать с ***/example/aaa*** или ***/example/bbb***.
- ***ImplementationSpecific*** может принимать оба вида (как просто совпадение имени, так и начало строки).

SERVICE MESH

Service Mesh — это выделенный слой инфраструктуры для обеспечения взаимодействия между сервисами. Он отвечает за надёжную доставку запросов через сложную топологию сервисов, составляющих современное приложение, созданное для работы в облаке.

D4.2 Основы Deployments в Kubernetes

DEPLOYMENTS & REPLICASET

Основные сценарии использования *Deployments*:

1. Создание необходимого количества ***ReplicaSet***, которые развернут поды согласно шаблону и будут следить за их статусом и количеством.
2. Переопределение состояния приложений через обновление ***PodTemplateSpec Deployments***. При этом создаётся новая версия ***ReplicaSet***, и ***Deployments*** управляет перемещением приложений из старого ***ReplicaSet*** в новый, согласно описанной стратегии деплоя.
3. Откат к предыдущей версии ***ReplicaSet***, если новая версия нестабильна.
4. Управление количеством ***ReplicaSet*** для масштабирования приложений.
5. Отправка состояния ***Deployments*** и сигнализация о сбое.
6. Очистка старых версий ***ReplicaSet*** после успешных развёртываний.

Какие бывают пробы?

ReadinessProbe

Данная проба говорит, что сервис внутри контейнера запустился и готов принимать трафик. Именно когда эта проба отрабатывает успешно, статус пода переходив в ***Ready***.

```
readinessProbe:
  httpGet:
    path: /healthz
    port: http-port
  failureThreshold: 1
  periodSeconds: 10
```

LivenessProbe

Тоже вполне говорящее название. Проверяется контейнер (с ним всё в порядке, или его требуется перезапустить).

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

Два перечисленных вида проб запускаются одновременно с созданием пода и начинают работать сразу.

Чтобы запускать ***LivenessProbe*** чуть позже (иногда приложению требуется время для старта), был добавлен параметр ***InitialsDelaySeconds***.

В новых версиях ***Kubernetes*** (начиная с 1.18) появился новый вид проб, призванный впоследствии заменить ***ReadinessProbe***.

StartupProbe

Данная проба всегда запускается первой и только после успешного срабатывания запускает две другие.

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

Все пробы имеют **три метода** с очень говорящими названиями:

- ***httpGet*** — ***kubelet*** выполняет ***http***-запрос на нужный порт и путь, при ответе 200 OK обнуляет счётчики неудач и возвращает статус OK. В случае неудачного запроса повышает счётчик ***failure*** на 1 и выполняет проверку повторно через заданный интервал времени. Если счётчик достигнет значения ***failureThreshold***, ***pod*** будет помечен нерабочим.

- ***tcpSocket*** — ***kubelet*** выполняет попытку установить ***tcp***-соединение на указанный порт с хоста (путь не используется, тут уже уровень ***L4***). В остальном ведёт себя так же, как предыдущий метод.
- ***exec*** — выполняет запуск команды или скрипта внутри контейнера через вызов ***kubelet***'ом команды *kubectl exec podname -c <command>*. Результатом работы может быть ***exit0***, что означает успех, либо ***exit1***, что означает провал. Есть ещё промежуточное состояние ***exit2***, на которое можно завязать свою логику (если речь идёт об операторах).

DAEMONSET

DaemonSet гарантирует, что все (или некоторые) хосты (***node***) запускают копию пода. По мере добавления хостов в кластер к ним добавляются поды. Когда хосты удаляются из кластера, эти поды собираются сборщиком мусора. Удаление ***DaemonSet*** очистит созданные им поды.

STATEFULSET

StatefulSet управляет развёртыванием и масштабированием набора подов и предоставляет гарантии порядка и уникальности этих подов.

D4.3 Стратегии деплоя

Название стратегии	Краткий принцип работы	Назначение стратегии
Recreate	Удаляет старую версию, разворачивает новую версию приложения.	Используется, если приложение нельзя запускать параллельно со старой версией, например из-за миграций в СУБД или использования файловой системы в режиме ***ReadWriteOnce***.
RollingUpdate	Запускает новую версию приложения параллельно со старой. Если новая версия запущена успешно, сворачивает старую версию, оставляя возможность отката изменений.	Используется для минимизации простоя приложения и быстрого отката изменений.

Blue/Green (Red/Black)	Параллельно запускает новую версию приложения, даёт возможность перенаправить весь трафик на новую версию и вывести из эксплуатации старую.	Используется для быстрой возможности отката изменений и минимизации простоя приложения. Также существуют дополнительные сценарии прогрева кэшей или предварительного тестирования новой версии.
Canary	Параллельно запускает новую версию приложения, но позволяет направлять в новую версию только часть трафика.	Используется для тестирования новой версии приложения на группе пользователей, поэтапной миграции пользователей или прогрева кэшей приложения.

D4.4 Операторы в Kubernetes

Цель оператора — предоставить возможность расширить стандартный ***API*** без модификации самого ***Kubernetes***. Оператор позволяет управлять множеством сущностей кластера ***Kubernetes***, не задумываясь о том, что у него под капотом (какие данные и что с ними делать, какие команды необходимо ещё выполнить для поддержания кластера).

Фактически оператор призван максимально упростить работу с приложением в рамках кластера, автоматизируя выполнение эксплуатационных задач, которые раньше приходилось решать вручную.

D4.5 Автоматическое масштабирование

ПРОФИЛЬ НАГРУЗКИ

Профилем нагрузки называют набор операций с заданными интенсивностями, полученный на основе сбора статистических данных либо определённый путём анализа требований к тестируемой системе.

ГОРИЗОНТАЛЬНОЕ МАСШТАБИРОВАНИЕ

Горизонтальное масштабирование — это увеличение числа инстансов для возможности параллельной обработки нагрузки.

ВЕРТИКАЛЬНОЕ МАСШТАБИРОВАНИЕ

Вертикальным масштабированием называют увеличение или уменьшение мощности оборудования. Например, мы можем заказать другой тип инстансов для наших хостов кластера или другой тип/тарифный план для СУБД.