1 Архитектура Kubernetes

Kubernetes — это одновременно крупный открытый проект и экосистема с большим количеством кода и богатой функциональностью. Автор Kubernetes — компания Google, но со временем этот проект присоединился к организации *Cloud Native Computing Foundation (CNCF)* и стал явным лидером в области контейнерных приложений. Если говорить коротко, это платформа для оркестрации развертывания и масштабирования контейнерных приложений и управления ими. Вы, вероятно, уже читали об этой системе и, может быть, даже применяли ее в своих проектах — любительских или профессиональных. Но, чтобы понять ее суть, научиться эффективно использовать и освоить передовой опыт, этого явно недостаточно.

В данной главе будет заложен фундамент знаний, необходимых для полноценного применения Kubernetes. Сначала я попытаюсь объяснить, чем эта система является, а чем — нет и что собой представляет оркестрация контейнеров. Затем будут рассмотрены важные концепции, которые встречаются на протяжении всей книги. После этого мы углубимся в архитектуру Kubernetes и посмотрим, каким образом пользователи могут задействовать все ее возможности. Дальше обсудим различные среды выполнения и механизмы контейнеризации, которые поддерживаются в Kubernetes (Docker — лишь один из вариантов). В конце поговорим о роли Kubernetes в замкнутой непрерывной интеграции и цикле развертывания.

По прочтении этой главы вы получите четкое понимание того, что такое оркестрация контейнеров. Будете знать, какие проблемы решает Kubernetes, чем продиктованы особенности структуры и архитектуры данной системы и какие среды выполнения она поддерживает. Вы также ознакомитесь со структурой открытого репозитория и сможете самостоятельно находить ответы на любые вопросы.

Что такое Kubernetes

Kubernetes — это платформа, которая вобрала в себя огромное и непрерывно растущее количество сервисов и возможностей. Ее основная функция заключается в планировании рабочей нагрузки на контейнеры в рамках вашей инфраструктуры, и это далеко не все. Далее перечислены некоторые дополнительные возможности Kubernetes:

- мониторинг систем хранения данных;
- распространение секретной информации;
- проверка работоспособности приложений;
- репликация узлов с приложениями;

- применение горизонтального автомасштабирования подов;
- именование и обнаружение;
- балансирование нагрузки;
- «обкатка» обновлений;
- мониторинг ресурсов;
- доступ к журнальным файлам и их обработка;
- отладка приложений;
- предоставление аутентификации и авторизации.

Чем Kubernetes не является

Kubernetes — это не *PaaS* (*platform as a service* — «платформа как услуга»). Она позволяет вам самостоятельно выбирать большинство важных функций будущей системы или предоставить этот выбор другим системам, построенным поверх Kubernetes, таким как Deis, OpenShift и Eldarion. В частности, Kubernetes:

- не диктует выбор определенного типа приложений или фреймворка;
- не требует выбирать конкретный язык программирования;
- не предоставляет базы данных или очереди сообщений;
- не делает различия между приложениями и сервисами;
- не предоставляет магазин сервисов с возможностью развертывания одним щелчком кнопкой мыши;
- дает возможность пользователям выбирать собственные системы журналирования, мониторинга и оповещения.

Оркестрация контейнеров

Основная функция Kubernetes заключается в оркестрации контейнеров, то есть планировании работы контейнеров разной степени загруженности на физических и виртуальных устройствах. Контейнеры должны быть эффективно упакованы, им необходимо соблюдать ограничения, налагаемые средой развертывания и конфигурацией кластера. Кроме того, платформа Kubernetes должна следить за всеми запущенными контейнерами и заменять те из них, которые вышли из строя, перестали отвечать или испытывают какие-то другие сложности. Kubernetes предоставляет множество других возможностей, о которых вы узнаете из следующих глав. В этом разделе мы сосредоточимся на контейнерах и оркестрации.

Контейнеры на физических и виртуальных устройствах

Все в компьютерном мире сводится к аппаратному обеспечению. Для выполнения работы вам нужно выделить реальные устройства, включая физические компьютеры с определенной вычислительной мощью (центральными процессорами или

ядрами), памятью и каким-то локальным постоянным хранилищем данных (жесткими дисками или SSD). Потребуются также общее хранилище данных и сетевое соединение, чтобы объединить все эти компьютеры и позволить им общаться друг с другом. В наши дни, помимо использования «голого железа», вы также можете запустить несколько виртуальных машин (ВМ) на одном и том же компьютере. Кластеры, на которых можно развертывать Kubernetes, могут быть как физическими, основанными на реальных устройствах, так и виртуальными, состоящими из виртуальных машин. Теоретически физические и виртуальные устройства комбинируются, однако такой подход не очень распространен.

Преимущества контейнеров

Контейнеры олицетворяют собой настоящий прорыв в парадигме разработки и выполнения крупных, сложных программных систем. По сравнению с традиционными моделями они имеют такие преимущества, как:

- ❑ быстрое создание и развертывание приложений;
- ❑ непрерывные разработка, интеграция и развертывание;
- ❑ разграничение ответственности разработчиков и администраторов;
- ❑ однородность сред разработки, тестирования и промышленного использования;
- ❑ переносимость между разными облачными провайдерами и ОС;
- ❑ сосредоточение управления непосредственно на приложении;
- ❑ слабо связанные, распределенные, эластичные, независимые микросервисы;
- ❑ изоляция ресурсов;
- ❑ утилизация ресурсов.

Контейнеры в облаке

Контейнеры обеспечивают изоляцию микросервисов, при этом они сильно упрощены, что делает их идеальным средством упаковывания. К тому же, в отличие от виртуальных машин, они не требуют значительных накладных расходов. Таким образом, контейнеры как нельзя лучше подходят для облачного развертывания, в условиях которого выделять каждому микросервису по виртуальной машине было бы слишком расточительно.

Все основные облачные провайдеры, такие как Amazon AWS, Google's GCE, Microsoft's Azure и даже Alibaba Cloud, предоставляют услуги по размещению контейнеров. Система GKE от Google изначально основывается на Kubernetes. AWS ECS имеет собственный механизм оркестрации. Microsoft Azure поддерживает контейнеры за счет Apache Mesos. Kubernetes можно развертывать на любых облачных платформах, но лишь недавно интеграция этой системы с другими сервисами стала достаточно глубокой. В конце 2017 года все облачные провайдеры объявили о непосредственной поддержке Kubernetes. Компания Microsoft запустила AKS, в AWS появилась услуга EKS, а в Alibaba Cloud начались работы по прозрачной интеграции Kubernetes с помощью Kubernetes controller manager.

От мелких домашних животных к крупному рогатому скоту

В прошлом, когда системы были небольшими, у каждого сервера было имя. Разработчики в точности знали, какое программное обеспечение на каждом устройстве используется. Помнится, во многих компаниях, в которых я работал, шли многодневные споры о том, в честь чего должны называться наши серверы. Среди популярных вариантов были композиторы и герои греческих мифов. Отношение к серверам было очень нежным, мы считали их чем-то вроде любимых домашних животных. Выход сервера из строя был целым происшествием. Все собирались вместе и обсуждали: где бы взять новый сервер, что было запущено на неисправном устройстве и как перенести это на новый компьютер. Если там хранились важные данные, оставалось лишь надеяться, что у нас есть резервная копия, которая, может быть, даже подлежит восстановлению.

Очевидно, что такой подход не масштабируется. Когда количество серверов переваливает за несколько десятков и тем более сотен, вам приходится обращаться с ними как с крупным рогатым скотом. Вы должны думать обо всем стаде, а не об отдельных животных. У вас по-прежнему могут быть домашние любимцы, но веб-серверы не должны входить в их число.

Kubernetes идеально вписывается в данную концепцию, принимая на себя всю ответственность за распределение контейнеров по конкретным компьютерам. В большинстве случаев вам не нужно взаимодействовать с отдельными устройствами (узлами). Лучше всего это подходит для приложений, которые не сохраняют свое состояние. В остальных случаях все немного сложнее, однако в Kubernetes для этого предусмотрено решение под названием StatefulSet, которое мы вскоре обсудим.

В данном разделе мы рассмотрели идею оркестрации контейнеров, поговорили о том, как они соотносятся с серверами — физическими или виртуальными, и взвесили преимущества от выполнения контейнеров в облаке. А в конце сделали абстрактное сравнение домашних питомцев со скотом. В следующем разделе мы познакомимся с миром Kubernetes — его концепциями и терминологией.

Концепции, лежащие в основе Kubernetes

В данном разделе я вкратце познакомлю вас со многими важными концепциями, лежащими в основе Kubernetes, и попытаюсь объяснить, зачем они нужны и как связаны между собой. Это делается для того, чтобы вы смогли ориентироваться в этих концепциях и терминах. Позже вы увидите, как все сплетается воедино, помогая сформировать группы API и категории ресурсов, чтобы в итоге мы получили отличные результаты. Многие из данных концепций можно рассматривать в качестве отдельных строительных блоков. Некоторые из них, например узлы и ведущие элементы, реализованы в виде набора компонентов. Эти компоненты находятся на другом уровне абстракции, и им посвящен раздел «Ведущие компоненты Kubernetes». Знаменитая схема архитектуры Kubernetes приведена на рис. 1.1.

Интернет
Брандмауэр
Прокси
kubelet
Под
cAdvisor
Контейнер
Узел
kubectl
(пользовательские команды)
Аутентификация,
авторизация
REST
(поды, сервисы,
контроллеры
репликации)
Диспетчер контроллеров
(контроллер
репликации и т. д.)
Распределенное
отслеживаемое
хранилище
Контроллер
планирования
Планировщик
API

Рис. 1.1

Кластер

Кластер — это набор компьютеров, хранилищ данных и сетевых ресурсов, с помощью которых Kubernetes выполняет различные задачи в вашей системе. Стоит отметить, что система может состоять из нескольких кластеров. Позже мы подробно рассмотрим нетривиальный сценарий с многокластерным режимом.

Узел

Узел — это отдельный компьютер (физический или виртуальный). Его задача состоит в запуске подов, о которых мы поговорим чуть позже. Каждый узел в Kubernetes содержит несколько компонентов, таких как kubelet и прокси kube. Все они находятся под управлением ведущего узла. Узлы — это нечто наподобие рабочих пчел, которые занимаются выполнением всех основных задач. Раньше они назывались *миньонами (minions)*, поэтому не удивляйтесь, если встретите этот термин в старой документации. Миньоны — это те же узлы.

Ведущий узел

Ведущий узел — это панель управления Kubernetes. Он состоит из нескольких компонентов, таких как API-сервер, планировщик и диспетчер контроллеров, и отвечает за глобальное (уровня кластера) планирование работы подов и обработку событий. Обычно все ведущие компоненты размещаются на едином узле. Но если сценарий предполагает высокую доступность или кластеры очень велики, следует подумать о том, чтобы сделать ведущий узел избыточным. Мы подробно обсудим высокодоступные кластеры в главе 4.

Под

Под (pod) — это единица работы в Kubernetes. Каждый под содержит один или несколько контейнеров. Поды всегда работают совместно, то есть на одном компьютере. Все контейнеры внутри пода имеют одни и те же IP-адрес и пространство портов, они могут общаться между собой через локальный сервер или посредством межпроцессного взаимодействия. Кроме того, все контейнеры имеют доступ к общему локальному хранилищу данных узла, на котором находится под. Такое хранилище может быть подключено к каждому контейнеру. Поды — важный элемент Kubernetes. В одном и том же Docker-контейнере можно запускать сразу несколько приложений, используя сервис supervisord для управления разными процессами, однако такой подход обычно проигрывает методу с применением подов. Рассмотрим причины этого.

- **Прозрачность.** Когда контейнеры внутри пода видны извне, это позволяет предоставлять им различные инфраструктурные услуги, такие как управление процессами и мониторинг ресурсов. Таким образом пользователи получают множество удобных возможностей.

- **Разделение программных зависимостей.** Мы можем управлять версиями отдельных контейнеров, заново их перестраивать и развертывать. Возможно, в Kubernetes когда-то появится поддержка горячих обновлений отдельных контейнеров.
- **Простота использования.** Пользователям не нужно запускать собственные диспетчеры процессов, беспокоиться о передаче сигналов и кодов завершения и т. д.
- **Эффективность.** Поскольку инфраструктура берет на себя большую ответственность, контейнеры могут быть упрощенными.

Поды обеспечивают отличное решение для управления группами тесно связанных между собой контейнеров, которые для выполнения своей задачи должны взаимодействовать на одном и том же узле. Важно помнить, что поды считаются фиктивными расходными сущностями, которые при желании можно удалить или заменить. Вместе с подом уничтожается любое хранилище данных, которое в нем находилось. Каждый экземпляр пода получает *уникальный идентификатор* (*unique ID*, или *UID*), чтобы при необходимости их можно было различить.

Метка

Метка — это пара «ключ — значение», с помощью нее группируются наборы объектов (зачастую подов). Этот механизм играет важную роль для нескольких других концепций, таких как контроллеры репликации, наборы реплик и сервисы, которые работают с динамическими группами объектов и должны как-то идентифицировать членов данных групп. Между объектами и метками существует связь вида $N \times N$. Каждый объект может иметь несколько меток, и каждая метка применима к разным объектам. Метки связаны определенными ограничениями, и это сделано намеренно. Например, каждая метка объекта должна иметь уникальный ключ, соответствующий строгому синтаксису. Он должен состоять из двух частей — префикса и имени. Префикс необязателен и отделяется от имени косой чертой (`/`). Он должен представлять собой корректный поддомен DNS и не может содержать свыше 253 символов. Имя обязано быть, его длина ограничена 63 символами. Имена должны начинаться и заканчиваться алфавитно-цифровыми символами (a — z, A — Z, 0–9) и содержать только буквы, цифры, точки, тире и подчеркивания. Значения подчиняются тем же правилам, что и имена. Следует отметить, что метки предназначены лишь для идентификации объектов, а для внедрения произвольных метаданных используются аннотации.

Аннотации

Связывать произвольные метаданные с объектами позволяют аннотации. Kubernetes всего лишь хранит аннотации и делает доступными их метаданные. В отличие от меток они не имеют строгих ограничений относительно допустимых символов и длины.

Мой опыт свидетельствует, что такие метаданные всегда необходимы в сложных системах. И замечательно, что в Kubernetes это понимают и предоставляют их в стандартной поставке, чтобы не приходилось проектировать собственное хранилище метаданных и связывать его с объектами.

Мы охватили большинство концепций Kubernetes, осталось еще несколько, которые я уже вскользь упоминал. В следующем разделе продолжим знакомство с архитектурой Kubernetes. Мы рассмотрим мотивацию, которая стоит за теми или иными структурными решениями, изучим содержимое и реализацию данной системы и даже заглянем в ее исходный код.

Селекторы меток

Селекторы меток используются для выбора объектов на основе их меток. Тождественные селекторы указывают имя ключа и значение. Для обозначения равенства или неравенства значений предусмотрены два оператора, = (или ==) и !=, например:

```
role = webserver
```

Это позволит получить все объекты, у которых метка содержит соответствующие ключ и значение.

Селектор метки может иметь несколько условий, разделенных запятой, например:

```
role = webserver, application != foo
```

Селекторы могут основываться на наборе из нескольких значений:

```
role in (webserver, backend)
```

Контроллеры репликации и наборы реплик

Контроллеры репликации и *наборы реплик* предназначены для управления группами подов, выбранных с помощью селекторов меток. Они гарантируют, что определенное количество экземпляров подов всегда находится в рабочем состоянии. Главное различие между ними состоит в том, что контроллеры репликации проверяют принадлежность к группе по одному имени, а наборы реплик позволяют указать несколько значений. Наборы реплик считаются предпочтительным инструментом, так как контроллеры реплик — это их подмножество и, как мне кажется, в какой-то момент они будут признаны устаревшими.

Kubernetes гарантирует, что количество работающих подов всегда соответствует значению, которое вы указали в контроллере репликации или наборе реплик. Каждый раз, когда это количество уменьшается в результате проблем с узлом или самим подом, Kubernetes запускает новые экземпляры. Имейте в виду, что, если вы сами превысите указанный лимит при ручном запуске подов, контроллер репликации удалит лишние.

Раньше контроллеры репликации играли главную роль во многих сценариях, таких как выкатывание обновлений и запуск одноразовых задач. Но с разви-

тием Kubernetes многие из этих сценариев получили поддержку в виде отдельных объектов, например *Deployment*, *Job* и *DaemonSet*. Все они будут рассмотрены позже.

Сервисы

Сервисы применяются для предоставления пользователям или другим сервисам определенных функций. Обычно они состоят из группы подов, которые, как несложно догадаться, идентифицируются с помощью меток. Сервисы могут предоставлять доступ к внешним ресурсам или экземплярам подов, которыми вы управляете напрямую на уровне виртуальных IP-адресов. В Kubernetes доступ к стандартным сервисам можно получить с помощью удобных конечных точек. Следует отметить, что сервисы работают на третьем сетевом уровне (TCP/UDP). В Kubernetes 1.2 появился объект `Ingress`, который предоставляет доступ к HTTP-объектам (подробней об этом чуть позже). Существует два механизма для публикации и поиска сервисов: DNS и переменные среды. Нагрузку на сервисы способна регулировать автоматически сама система Kubernetes, но если при этом применяются внешние ресурсы или требуется особое обращение, разработчики могут выполнять балансировку вручную.

Я опустил множество технических деталей, связанных с пространствами портов и реальными/виртуальными IP-адресами. В последующих главах они будут рассмотрены во всех подробностях.

Том

Локальное хранилище в поде — элемент временный и удаляется вместе с подом. Иногда этого достаточно, если вам всего лишь нужно передавать информацию между контейнерами на одном узле. Но иногда данные должны сохраняться и после уничтожения пода или быть доступными для нескольких его экземпляров. На этот случай предусмотрена концепция *томов*. Docker тоже имеет поддержку томов, но она довольно ограничена (хотя ее постоянно развивают). Платформа Kubernetes использует собственные тома и поддерживает дополнительные типы контейнеров, такие как rkt, поэтому принципиально не может полагаться на аналогичный механизм в Docker.

Существует множество разновидностей томов. Многие из них напрямую поддерживаются в Kubernetes, но современный подход предполагает добавление новых типов томов через интерфейс *CSI* (*Container Storage Interface* — интерфейс хранилища контейнеров), который мы подробно обсудим чуть позже. Тома типа `emptyDir` подключаются к каждому контейнеру и основываются на содержимом материнской системы. При желании их можно разместить в памяти. Это хранилище удаляется при уничтожении пода. Существует много разных видов томов для определенных облачных платформ, сетевых файловых систем и даже репозиториев Git. В качестве любопытного примера приведу том `persistentDiskClaim`, который использует стандартное для вашей среды хранилище (обычно в облаке), инкапсулируя кое-какие детали.

StatefulSet

Поды появляются и исчезают, и если вам важны содержащиеся в них данные, можете задействовать постоянное хранилище. Это неплохой вариант, но иногда возникает необходимость в автоматическом распределенном хранилище данных, например MySQL Galera. Подобные кластеризованные системы распределяют данные по разным узлам с уникальными идентификаторами. Вы добьетесь этого с помощью обычных подов и сервисов или если воспользуетесь StatefulSet. Помните, мы сравнивали серверы с домашними животными и крупным рогатым скотом и объясняли, почему скот — правильная метафора? Так вот, StatefulSet находится где-то посередине. По аналогии с наборами реплик данная концепция гарантирует, что в любой момент у нас есть столько-то «домашних питомцев» с уникальными идентификаторами. Каждый такой «питомец» имеет:

- стабильное сетевое имя, доступное через DNS;
- порядковый номер;
- стабильное хранилище, привязанное к порядковому номеру и сетевому имени.

StatefulSet может помочь с обнаружением узлов, а также их добавлением и удалением.

Конфиденциальная информация

Конфиденциальная информация, такая как учетные данные и сведения о токенах, хранится в виде небольших объектов. Они находятся внутри хранилища etcd. Если поду нужен доступ к этим объектам, он может подключить сервер Kubernetes API в виде набора файлов с помощью выделенных секретных томов, ссылающихся на обычные тома с данными. Один и тот же секретный объект реально подключить к нескольким экземплярам пода. Платформа Kubernetes самостоятельно создает секретные объекты для своих компонентов, и вы можете делать то же самое. В качестве альтернативы конфиденциальную информацию можно размещать в переменных среды. Заметьте, что для пущей безопасности подобная информация в подах всегда находится в памяти (в `tmpfs`, если речь идет о подключении секретных объектов).

Имена

Каждый объект в Kubernetes идентифицируется по идентификатору UID и имени. Имена позволяют ссылаться на объекты в API-вызовах, могут состоять из цифр, букв в нижнем регистре, тире (`-`) и точек (`.`), а их длина не должна превышать 253 символов. Если удалить объект, вместо него можно создать новый с тем же именем, но UID должен быть уникальным в рамках жизненного цикла кластера. UID генерируются платформой Kubernetes, так что вы об этом заботиться не должны.