Контейнеризация, Docker и Docker compose

Виртуализация и контейнеризация

Перед тем как работать с Docker необходимо рассмотреть такие понятия, как виртуализация и контейнеризация.

Виртуализация — предоставление набора вычислительных ресурсов или их логического объединения, абстрагированное от аппаратной реализации, и обеспечивающее при этом логическую изоляцию друг от друга вычислительных процессов, выполняемых на одном физическом ресурсе.

Контейнеризация — это виртуализация на уровне операционной системы (то есть НЕ аппаратная), при которой ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного.

Пространство пользователя — это адресное пространство виртуальной памяти операционной системы, отводимое для пользовательских программ.

Экземпляры пространства пользователя (обычно называемые контейнерами) с точки зрения пользователя полностью идентичны отдельному экземпляру операционной системы. Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга.

Получается, что контейнеризация — это программная виртуализация, то есть виртуализация на уровне операционной системы, за которую отвечает ядро операционной системы. Одной из характерных черт такого подхода является использование всеми контейнерами общего ядра, того же, что и у хостовой операционной системы (то есть той, в которой размещены контейнеры). Это позволяет избавиться от накладных расходов на эмуляцию виртуального оборудования и запуска полноценного экземпляра операционной системы. Можно сказать, что это "легковесная" виртуализация.

Ядро (kernel) — центральная часть ОС, обеспечивающая приложениям координированный доступ к ресурсам компьютера, таким как процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации. Также обычно ядро предоставляет сервисы файловой системы и сетевых протоколов. В общем, это сердце всей системы.

Оно обеспечивает изолированность экземпляров пространств пользователей. В этом контексте может встретиться такой термин как "cgroups". Так называется как раз тот самый механизм ядра Linux, который позволяет этого добиться. Поэтому, можно сказать, что путь контейнеризации начался с Linux систем. Однако, начиная с Windows 10 в windows тоже появилась поддержка контейнеризации.

Для чего нужен Docker?

Docker — это открытая платформа для разработки, доставки и запуска приложений. Docker позволяет отделить ваши приложения от инфраструктуры, чтобы вы могли быстро доставлять программное обеспечение. С помощью Docker вы можете управлять своей инфраструктурой так же, как вы управляете своими приложениями. Воспользовавшись преимуществами методологий Docker для доставки, тестирования и развертывания кода, вы можете значительно сократить задержку между написанием кода и его запуском в рабочей среде.

Docker платформа

Docker предоставляет возможность упаковать и запустить приложение в слабо изолированной среде, называемой контейнером. Изоляция и безопасность позволяют одновременно запускать множество контейнеров на одном хосте. Контейнеры легкие и содержат все необходимое для запуска приложения, поэтому вам не нужно полагаться на то, что установлено на хосте. Вы можете совместно использовать контейнеры во время работы и быть уверены, что все, с кем вы делитесь, получают один и тот же контейнер, который работает одинаково.

Docker предоставляет инструменты и платформу для управления жизненным циклом ваших контейнеров:

- Разрабатывайте свое приложение и поддерживающие его компоненты с помощью контейнеров.
- Контейнер становится единицей распространения и тестирования вашего приложения.
- Когда вы будете готовы, разверните свое приложение в производственной среде в виде контейнера или службы оркестратора. Это будет работать одинаково, независимо от того, является ли ваша производственная среда локальным центром обработки данных, поставщиком облачных услуг или гибридом.

Для чего может быть использован Docker?

Быстрая и последовательная доставка ваших приложений

Docker оптимизирует жизненный цикл разработки, позволяя разработчикам работать в стандартизированных средах, используя локальные контейнеры, которые предоставляют ваши приложения и сервисы. Контейнеры отлично подходят для рабочих процессов непрерывной интеграции и непрерывной доставки (CI/CD).

Адаптивное развертывание и масштабирование

Контейнерная платформа Docker обеспечивает высокую переносимость рабочих нагрузок. Контейнеры Docker могут работать на локальном ноутбуке разработчика, на физических или виртуальных машинах в центре обработки данных, на серверах облачных провайдеров или в различных средах.

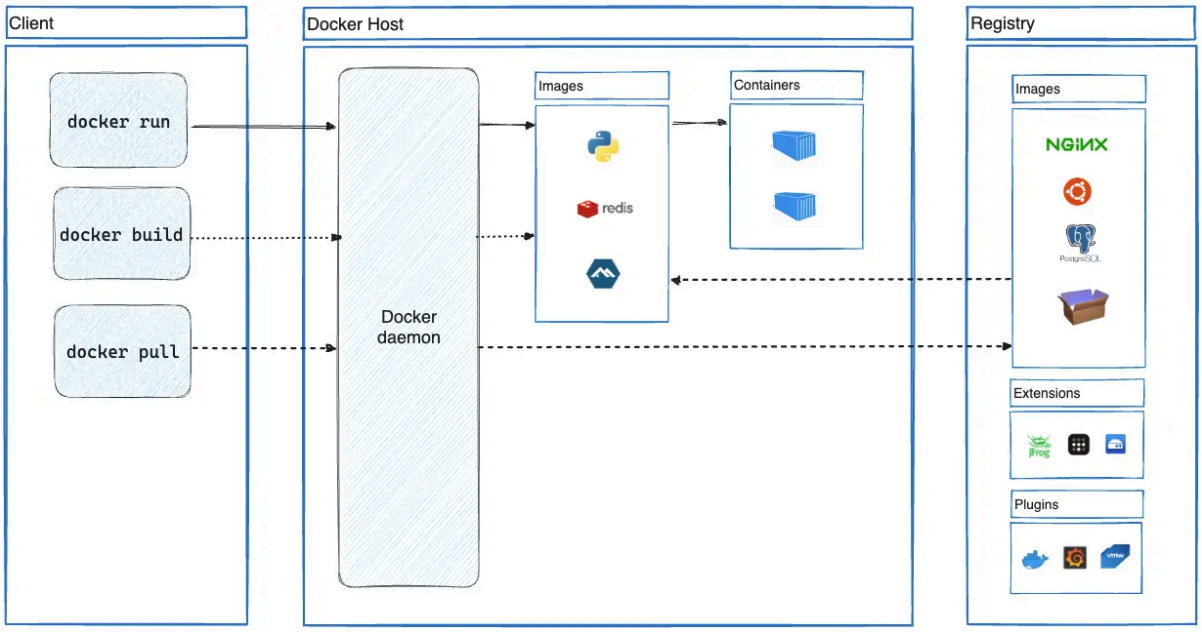
Портативность и легкость Docker также упрощают динамическое управление рабочими нагрузками, масштабирование или удаление приложений и сервисов в соответствии с потребностями бизнеса практически в реальном времени.

Выполнение большего количества рабочих нагрузок на одном и том же оборудовании

Docker легкий и быстрый. Он обеспечивает жизнеспособную и экономичную альтернативу виртуальным машинам на базе гипервизора, поэтому вы можете использовать больше мощности вашего сервера для достижения своих бизнес-целей. Docker идеально подходит для сред с высокой плотностью размещения, а также для малых и средних развертываний, где вам нужно делать больше с меньшими ресурсами.

Архитектура Docker

Docker использует архитектуру клиент-сервер. Клиент Docker взаимодействует с демоном Docker, который выполняет тяжелую работу по созданию, запуску и распространению контейнеров Docker. Клиент Docker и демон могут работать в одной системе, или вы можете подключить клиент Docker к удаленному демону Docker. Клиент и демон Docker взаимодействуют с помощью REST API, через сокеты UNIX или сетевой интерфейс. Еще один клиент Docker — Docker Compose, который позволяет работать с приложениями, состоящими из набора контейнеров.



Docker daemon

Демон Docker (dockerd) прослушивает запросы Docker API и управляет объектами Docker, такими как образы, контейнеры, сети и тома. Демон также может взаимодействовать с другими демонами для управления службами Docker.

Docker client

Клиент Docker (docker) — это основной способ взаимодействия многих пользователей Docker с Docker. Когда вы используете такие команды, как docker run, клиент отправляет эти команды в dockerd, который их выполняет. Команда docker использует Docker API. Клиент Docker может взаимодействовать с несколькими демонами.

Докер реестры

В докер реестре хранятся образы Docker. Docker Hub — это общедоступный реестр, который может использовать каждый, и Docker по умолчанию ищет образы в Docker Hub. Вы даже можете запустить свой собственный частный реестр.

Когда вы используете команды docker pull или docker run, Docker извлекает необходимые образы из настроенного вами реестра. Когда вы используете команду docker push, Docker отправляет ваш образ в настроенный реестр.

Докер-объекты

Когда вы используете Docker, вы создаете и используете образы, контейнеры, сети, тома, плагины и другие объекты. В этом разделе представлен краткий обзор некоторых из этих объектов.

Образы

Образ — это шаблон, доступный только для чтения, с инструкциями по созданию Docker-контейнера. Часто образ основан на другом образе с некоторой дополнительной настройкой. Например, вы можете создать образ, базирующийся на ubuntu, но устанавливающий веб-сервер Apache и ваше приложение, а также детали конфигурации, необходимые для запуска вашего приложения.

Вы можете создавать свои собственные образы или использовать только те, которые созданы другими и опубликованы в реестре. Чтобы создать собственный образ, вы создаете Dockerfile с простым синтаксисом для определения шагов, необходимых для создания образа и его запуска. Каждая инструкция в Dockerfile создает слой в образе. Когда вы меняете Dockerfile и перестраиваете образ, перестраиваются только те слои, которые изменились. Это часть того, что делает образы такими легкими, маленькими и быстрыми по сравнению с другими технологиями виртуализации.

Контейнеры

Контейнер — это работоспособный экземпляр образа. Вы можете создавать, запускать, останавливать, перемещать или удалять контейнер с помощью Docker API. Вы можете подключить контейнер к одной или нескольким сетям, подключить к нему хранилище или даже создать новый образ на основе его текущего состояния.

По умолчанию контейнер относительно хорошо изолирован от других контейнеров и хост-компьютера. Вы можете контролировать, насколько изолирована сеть, хранилище или другие базовые подсистемы контейнера от других контейнеров или от хост-компьютера.

Контейнер определяется его образом, а также любыми параметрами конфигурации, которые вы предоставляете ему при его создании или запуске. Когда контейнер удаляется, любые изменения его состояния, которые не сохраняются в постоянном хранилище, исчезают.

Пример команды запуска Docker

Следующая команда запускает ubuntu контейнер, интерактивно подключается к локальному сеансу командной строки и запускает /bin/bash.

```
docker run -i -t ubuntu /bin/bash
```

Когда вы запускаете эту команду, происходит следующее:

1. Если у вас нет ubuntu образа на локальном хосте, то Docker извлекает его из настроенного реестра, как если бы вы запускали его `docker pull ubuntu` вручную.

2. Docker создает новый контейнер, как если бы вы выполнили `docker container create` команду вручную.
3. Docker выделяет для контейнера файловую систему для чтения и записи в качестве его последнего уровня. Это позволяет работающему контейнеру создавать или изменять файлы и каталоги в своей локальной файловой системе.
4. Docker создает сетевой интерфейс для подключения контейнера к сети по умолчанию, поскольку вы не указали никаких сетевых параметров. Это включает в себя назначение IP-адреса контейнеру. По умолчанию контейнеры могут подключаться к внешним сетям, используя сетевое соединение хост-компьютера.
5. Docker запускает контейнер и выполняет /bin/bash. Поскольку контейнер работает в интерактивном режиме и подключен к вашему терминалу (благодаря флагам -i и -t), вы можете вводить ввод с помощью клавиатуры, в то время как Docker записывает выходные данные на ваш терминал.
6. Когда вы запускаете команду exit для завершения /bin/bash команды, контейнер останавливается, но не удаляется. Вы можете запустить его снова или удалить.