# Введение в технологию Docker

Технология Docker предлагает подход, основанный на концепции контейнеризации. Контейнеры Docker – это легкие, изолированные и самодостаточные пакеты, которые содержат все необходимое для запуска приложения, включая код, библиотеки, зависимости и конфигурационные файлы. Это позволяет создавать и запускать приложения практически в любой среде, независимо от операционной системы или инфраструктуры.

Что такое контейнер Docker?

Представьте себе идеальную “коробку”, которая содержит все необходимое для работы конкретного приложения:

Код: файлы кода приложения – самое сердце вашего программного продукта.

Библиотеки: набор инструментов и функций, необходимых для работы кода приложения. Это могут быть языковые библиотеки, фреймворки, инструменты для работы с данными и многое другое.

Зависимости: другие программы или компоненты, от которых зависит ваше приложение. Например, базы данных, серверы, среды выполнения (runtime environments).

Конфигурационные файлы: файлы, которые определяют настройки и параметры работы приложения.

Команды Docker

Для полноценной работы с платформой Docker необходимо знать основные термины, использующиеся при обработке информации.

**Образ контейнера** — пакет со всеми зависимостями и сведениями, необходимыми для создания контейнера. Образ включает в себя все зависимости (например, платформы), а также конфигурацию развертывания и выполнения для среды выполнения контейнера. Как правило, образ создается на основе нескольких базовых образов, наслоенных друг на друга в файловой системе контейнера. После создания образ остается неизменным. Основная цель образа — привести среду (зависимости) к единообразию в различных развертываниях. Это означает, что вы можете отладить образ на одном компьютере, а затем развернуть его на другом компьютере и получить ту же среду.

**Dockerfile**: текстовый файл, содержащий инструкции по созданию образа Docker. Он похож на пакетный сценарий, где первая строка указывает базовый образ, с которого начинается работа, а следующие инструкции устанавливают необходимые программы, копируют файлы и т. п. для создания необходимой рабочей среды.

**Сборка** — действие по созданию образа контейнера на основе сведений и контекста, предоставленных файлом Dockerfile, а также дополнительных файлов в папке, где создается образ. Сборка образов выполняется с помощью следующей команды «docker build».

**Контейнер** — экземпляр образа Docker. Контейнер отвечает за выполнение одного приложения, процесса или службы. Он состоит из содержимого образа Docker, среды выполнения и стандартного набора инструкций. При масштабировании службы вы создаете несколько экземпляров контейнера из одного образа. Или пакетное задание может создать несколько контейнеров из одного образа, передавая разные параметры каждому экземпляру.

**Тома** —­ предложите файловую систему с возможностью записи, которую может использовать этот контейнер. Поскольку образы доступны только для чтения, а большинству программ требуется возможность записи в файловую систему, тома добавляют слой с поддержкой записи поверх образа контейнера, который программы смогут использовать как файловую систему с возможностью записи. Программа не знает, что она обращается к многоуровневой файловой системе. Для нее это обычный вызов файловой системы. Тома размещаются в системе узла под управлением Docker.

**Тег** — метка для образа, которая позволяет различать разные образы или версии одного образа (в зависимости от номера версии или целевой среды).

**Многоэтапная сборка** — этот компонент входит в Docker, начиная с версии 17.05, и позволяет сократить итоговый размер образов. Например, большой базовый образ, содержащий пакет SDK, можно использовать для компиляции и публикации, а затем можно использовать небольшой базовый образ среды выполнения для размещения приложения.

**Репозиторий** — коллекция связанных образов Docker, помеченная тегом, указывающим на версию образа. Некоторые репозитории содержат несколько вариантов определенного образа, например, изображения, содержащего пакеты SDK (более тяжелые), изображение, содержащее только среды выполнения (светлее) и т. д. Эти варианты можно пометить тегами. Один репозиторий может содержать варианты платформ, например, образ Linux и образ Windows.

**Реестр** — служба, предоставляющая доступ к репозиториям. Реестр по умолчанию для большинства общедоступных образов — «Центр Docker» (принадлежащий Docker как организации). Реестр обычно содержит репозитории нескольких команд. Компании часто используют частные реестры для хранения своих образов и управления ими. Еще один пример — реестр контейнеров Azure.

**Многоархивное изображение**: для мультиплатформенного — это функция Docker, которая упрощает выбор соответствующего образа в соответствии с платформой, в которой работает Docker. Например, когда Dockerfile запрашивает базовый образ из реестра, он фактически получает **8.0-nanoserver-ltsc2022, 8.0-nanoserver-1809**или**8.0-bullseye-slim** в зависимости от операционной системы и версии, в которой работает Docker.

**Центр Docker** — общедоступный реестр для загрузки образов и работы с ними. Центр Docker обеспечивает размещение образов Docker и интеграцию с GitHub и Bitbucket, предоставляет общедоступные или частные реестры, триггеры сборки и веб-перехватчики.

**Реестр контейнеров Azure** — общедоступный ресурс для работы с образами Docker и их компонентами в Azure. Он предоставляет реестр, близкий к вашим развертываниям в Azure, так что вы можете контролировать доступ и использовать группы и разрешения в Azure Active Directory.

**Доверенный реестр Docker (DTR)** — служба реестра Docker (из Docker), которую можно установить локально, чтобы она находилась в центре данных и сети организации. Его удобно использовать для частных образов, которыми необходимо управлять внутри предприятия. Доверенный реестр Docker входит в Центр данных Docker.

**Docker Desktop**: инструменты разработки в среде Windows и macOS для локальной сборки, выполнения и тестирования контейнеров. Docker Desktop для Windows предоставляет среды разработки для контейнеров Linux и Windows. Узел Linux Docker на Windows базируется на виртуальной машине Hyper-V. Узел для контейнеров Windows базируется непосредственно на Windows. Docker Desktop для Mac основан на платформе Apple Hypervisor и гипервизоре xhyve, который предоставляет виртуальную машину узла Docker для Linux в macOS. Docker Desktop для Windows и Mac заменяет решение Docker Toolbox, которое было основано на Oracle VirtualBox.

**Compose** — программа командной строки и формат файлов YAML с метаданными для определения и выполнения многоконтейнерных приложений. Вы определяете одно приложение на основе нескольких образов с помощью одного или нескольких файлов .yml, которые могут изменять значения в зависимости от среды. Создав определения, вы можете развернуть все многоконтейнерное приложение с помощью одной команды (docker-compose up), которая создает один контейнер на образ на узле Docker.

**Кластер** — коллекция узлов Docker, представленная в виде единого виртуального узла Docker, чтобы можно было масштабировать приложение в нескольких экземплярах служб, распределенных по нескольким узлам кластера. Кластеры Docker можно создавать с помощью Kubernetes, Azure Service Fabric, Docker Swarm и (или) Mesosphere DC/OS.

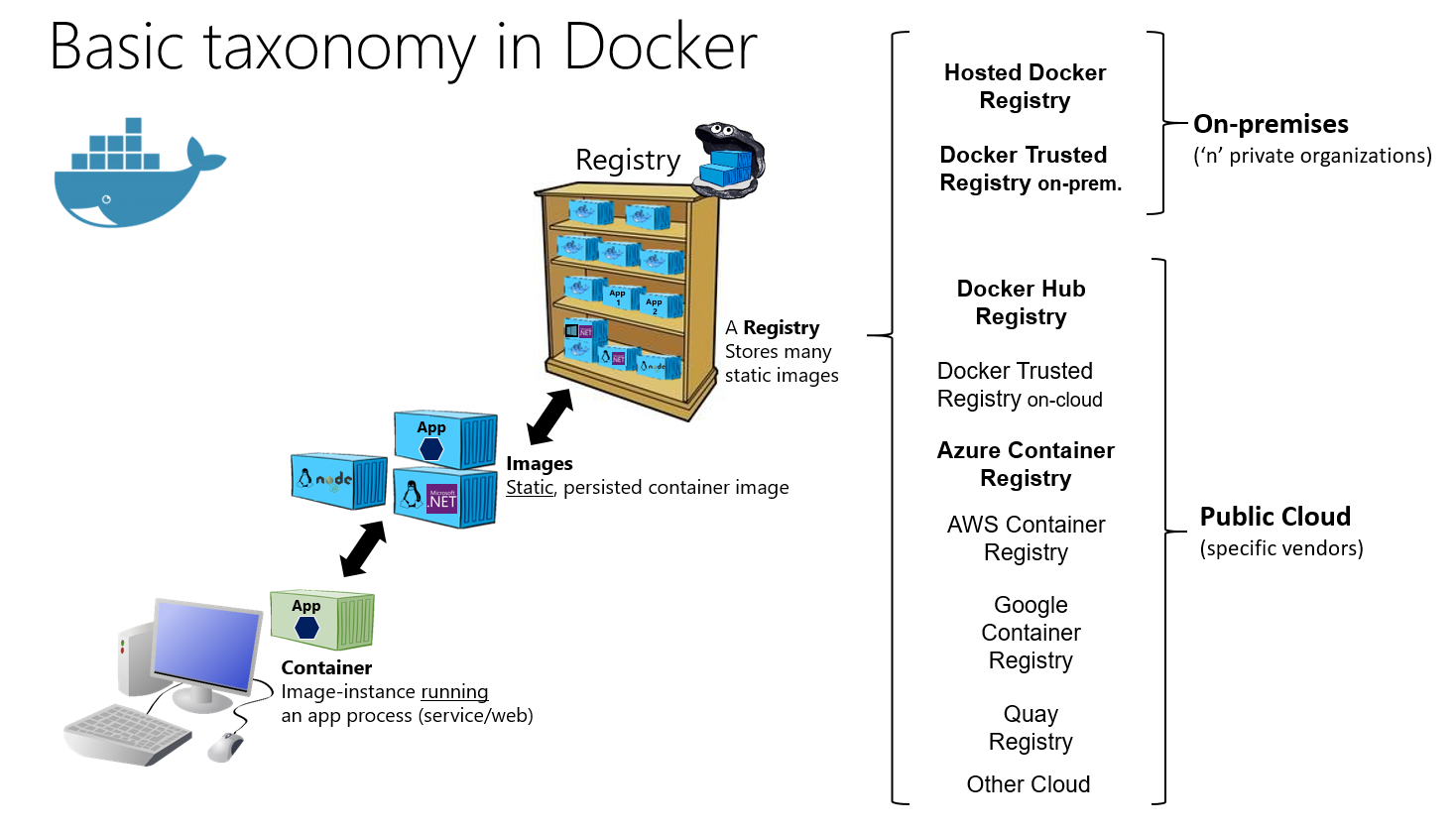
**Orchestrator**: инструмент, упрощающий управление кластерами и узлами Docker. С помощью оркестраторов можно управлять образами, контейнерами и узлами через интерфейс командной строки (CLI) или графический интерфейс. Вы можете управлять соединениями контейнеров, конфигурациями, балансировкой нагрузки, обнаружением служб, высоким уровнем доступности, конфигурацией узлов Docker и многим другим. Оркестратор используется для выполнения, распределения, масштабирования и восстановления рабочих нагрузок в коллекции узлов. В качестве оркестраторов обычно используются те же продукты, которые обеспечивают кластерную инфраструктуру, например: Kubernetes и Azure Service Fabric (на рынке доступны и другие предложения).

При использовании Docker разработчик создает приложение или службу и упаковывает приложение или службу и их зависимости в образ контейнера. Образ — это статическое представление приложения или службы, а также их конфигурации и зависимостей.

Для запуска приложения или службы создается экземпляр образа приложения, чтобы создать контейнер, который будет запущен на узле Docker. Контейнеры изначально проверяются в среде разработки или на ПК.

Разработчикам следует хранить образы в реестре, который выступает в качестве библиотеки образов и необходим при развертывании на оркестраторы в рабочей среде. Docker поддерживает общедоступный реестр с помощью «Docker hub». Другие поставщики предлагают реестры для различных коллекций образов, включая Реестр контейнеров Azure. Кроме того, организации могут развернуть локальные частные реестры для своих образов Docker.

На рисунке 1 показано, как образы и реестры в Docker связаны с другими компонентами. На нем также показано несколько вариантов реестра от поставщиков.

  
Рисунок 1 — Классификация основных понятий Docker

Реестр напоминает книжную полку. В нем хранятся образы, которые можно извлекать для создания контейнеров, необходимых для запуска служб или веб-приложений. Частные реестры Docker могут храниться в локальной среде или в общедоступном облаке. Docker Hub — это общедоступный реестр, поддерживаемый Docker. Помимо решения Docker Trusted Registry корпоративного уровня можно использовать Реестр контейнеров Azure, предлагаемый Azure. AWS, Google и другие компании также предлагают свои реестры контейнеров.

Размещение образов в реестре позволяет хранить статические и неизменяемые фрагменты приложений, включая всех их зависимости на уровне платформы. Эти образы можно добавить в систему управления версиями и развернуть в нескольких средах, таким образом, каждый образ представляет согласованную единицу развертывания.

Частные реестры образов, размещенные локально или в облаке, рекомендуется использовать, если:

— ваши образы не могут быть открыты для общего доступа по соображениям конфиденциальности;

— вам необходимо обеспечить минимальную сетевую задержку между образами и выбранной средой развертывания. Например, если рабочая среда представляет собой облако Azure, вы, вероятно, захотите разместить образы в Реестре контейнеров Azure, чтобы сетевая задержка была минимальной. Точно так же, если рабочая среда является локальной, вы можете развернуть локальный доверенный реестр Docker в той же локальной сети.

Среда разработки приложений Docker

**Выбор средства разработки: IDE или редактор**

Предпочитаете ли вы использовать полнофункциональную среду IDE или упрощенный редактор, корпорация Майкрософт предлагает средства, с помощью которых можно разрабатывать приложения Docker.

**Visual Studio (для Windows).** Для разработки приложений .NET 6 на основе Docker в Visual Studio требуется Visual Studio 2022 версии 17.0 или более поздней. Среда Visual Studio 2022 содержит встроенные средства для использования Docker. Средства для Docker позволяют разрабатывать, запускать и проверять приложения непосредственно в целевой среде Docker. Нажмите клавишу F5 для запуска и отладки приложения (на основе одного контейнера или нескольких) непосредственно в узле Docker или нажмите сочетание клавиш CTRL + F5 для редактирования и обновления приложения без повторной сборки контейнера. Интегрированная среда разработки — самый эффективный инструмент для разработки приложений на основе Docker.

**Visual Studio для Mac.** Это интегрированная среда разработки (дальнейшее развитие Xamarin Studio), которая работает в macOS. Для разработки на платформе .NET 6 требуется версия 8.4 или более поздняя. Этот инструмент должен быть предпочтительным вариантом для разработчиков, которые работают на компьютерах с macOS и хотят использовать мощную интегрированную среду разработки.

**Visual Studio Code и CLI Docker**. Если вам нужен упрощенный кроссплатформенный редактор, поддерживающий любой язык программирования, то вы можете использовать Visual Studio Code и CLI Docker. Интегрированная среда разработки реализует кроссплатформенный подход к разработке приложений для macOS, Linux и Windows. Кроме того, Visual Studio Code поддерживает расширения для Docker, такие как IntelliSense для Dockerfile, и ярлыки для выполнения команд Docker из редактора.

Установив [Docker Desktop](https://hub.docker.com/search/?type=edition&offering=community), вы можете использовать единый интерфейс CLI Docker, чтобы создавать приложения как для Windows, так и для Linux.

**Рабочий процесс разработки приложений Docker на основе контейнера**

В этом разделе описывается рабочий процесс внутреннего цикла разработки приложений на основе контейнера Docker. Рабочий процесс внутреннего цикла означает, что речь идет только о разработке, которая выполняется на компьютере разработчика, не касаясь более широкого рабочего процесса DevOps. Начальные этапы настройки среды здесь не рассматриваются, так как они выполняются только один раз.

Приложение состоит из ваших собственных служб и дополнительных библиотек (зависимостей). Ниже приведены основные шаги, которые обычно выполняются при сборке приложения Docker, как показано на рисунке 2.

Процесс разработки для приложений Docker: 1 — код приложения, 2 — запись Dockerfile/s, 3 — создание образов, определенных в Dockerfile/s, 4 — (необязательно) Создание служб в файле docker-compose.yml, 5 — запуск контейнера или приложения docker-compose, 6 — тестирование приложения или микрослужб, 7 — отправка в репозиторий и повтор.

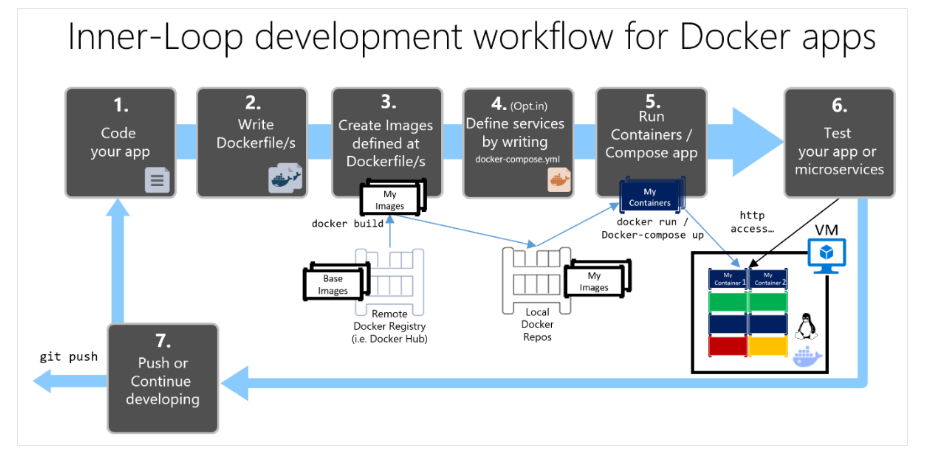


Рисунок 2— Пошаговый рабочий процесс разработки приложения на основе контейнера Docker

В этом разделе подробно описывается весь процесс, и каждый важный шаг объясняется с акцентом на среду Visual Studio.

Если разработка выполняется с помощью редактора и CLI (например, Visual Studio Code и Docker CLI на macOS или Windows), то необходимо знать каждый шаг и обычно более детально, чем при использовании Visual Studio. Дополнительные сведения о работе в среде CLI см. в электронной книге Жизненный цикл приложений в контейнерах Docker с платформами и средствами Майкрософт.

При использовании Visual Studio 2022 многие из этих шагов выполняются автоматически, что значительно повышает производительность. Это особенно справедливо в тех случаях, когда используется Visual Studio 2022 для работы с многоконтейнерными приложениями. Например, всего лишь одним щелчком мыши Visual Studio добавляет Dockerfile и файл docker-compose.yml в проекты с конфигурацией для вашего приложения. При запуске приложения в Visual Studio он создает образ Docker и запускает многоконтейнерное приложение непосредственно в Docker; вы даже можете отлаживать несколько контейнеров одновременно. Эти возможности значительно повышают скорость разработки.

Однако то, что Visual Studio автоматизирует эти действия, не означает, что вам не нужно знать, что происходит внутри Docker. Таким образом, каждый шаг подробно описывается в следующих рекомендациях.

**Устройство и принцип работы Docker**

Виртуализация в Docker реализуется на уровне ОС. Виртуальная среда запускается прямо из ядра основной операционной системы и использует её ресурсы.

В поставку Docker входят следующие компоненты (рисунки 3 и 4):

**Docker host** — это операционная система, на которую устанавливают Docker и на которой он работает.

**Docker daemon** — служба, которая управляет Docker-объектами: сетями, хранилищами, образами и контейнерами.

**Docker client** — консольный клиент, при помощи которого пользователи взаимодействуют с Docker daemon и отправляют ему команды, создают контейнеры и управляют ими.

**Docker image** — это неизменяемый образ, из которого разворачивается контейнер.

**Docker container** — развёрнутое и запущенное приложение.

**Docker Registry** — репозиторий, в котором хранятся образы.

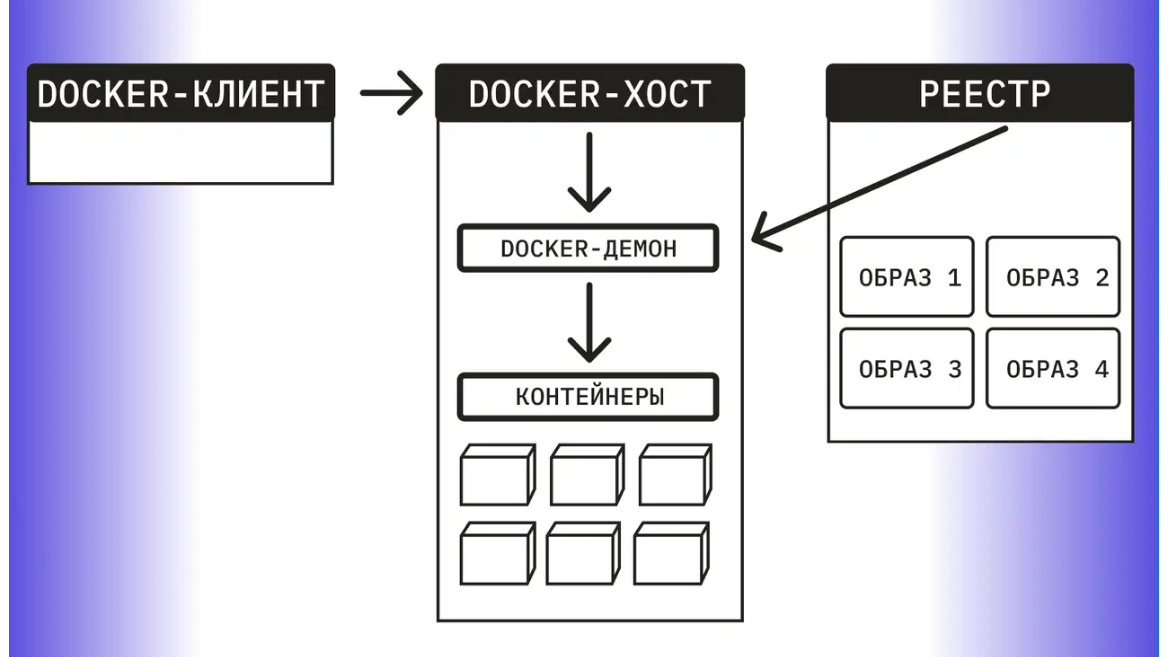
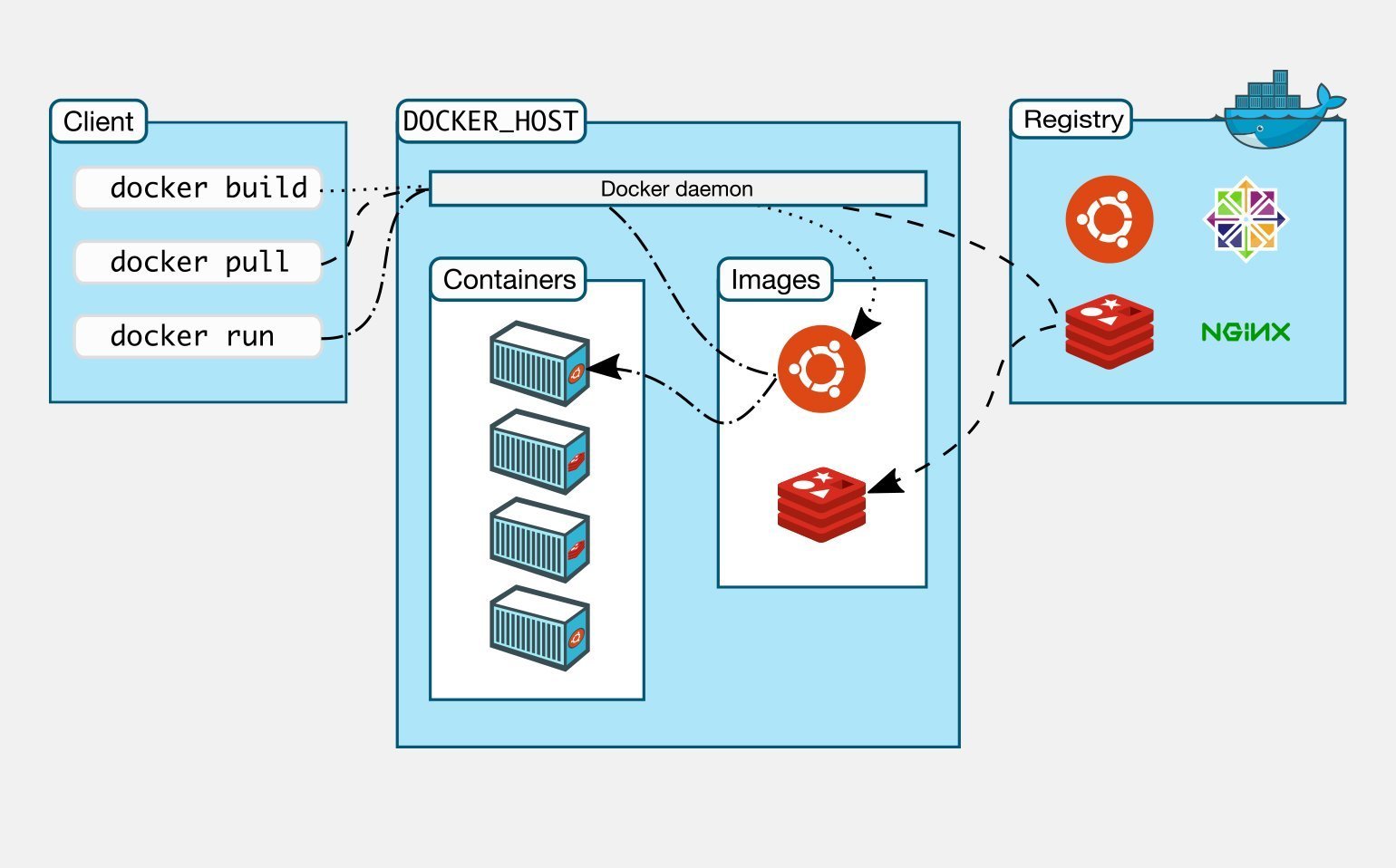


Рисунок 3 —Простая структура Docker

  
Рисунок 4 — Схемы архитектуры Docker

**Dockerfile** — файл-инструкция для сборки образа.

**Docker Compose** — инструмент для управления несколькими контейнерами. Он позволяет создавать контейнеры и задавать их конфигурацию.

**Docker Desktop** — GUI-клиент, который распространяется по [GPL](https://skillbox.ru/media/code/gayd-po-svobodnym-litsenziyam-ot-gnu-chto-eto-takoe-i-kakimi-oni-byvayut/?utm_source=media&utm_medium=link&utm_campaign=all_all_media_links_links_articles_all_all_skillbox). Бесплатная версия работает на Windows, macOS, а с недавних пор и на Linux. Это очень удобный клиент, который отображает все сущности Docker и позволяет запустить однонодовый Kubernetes для компьютера (рис. 5).

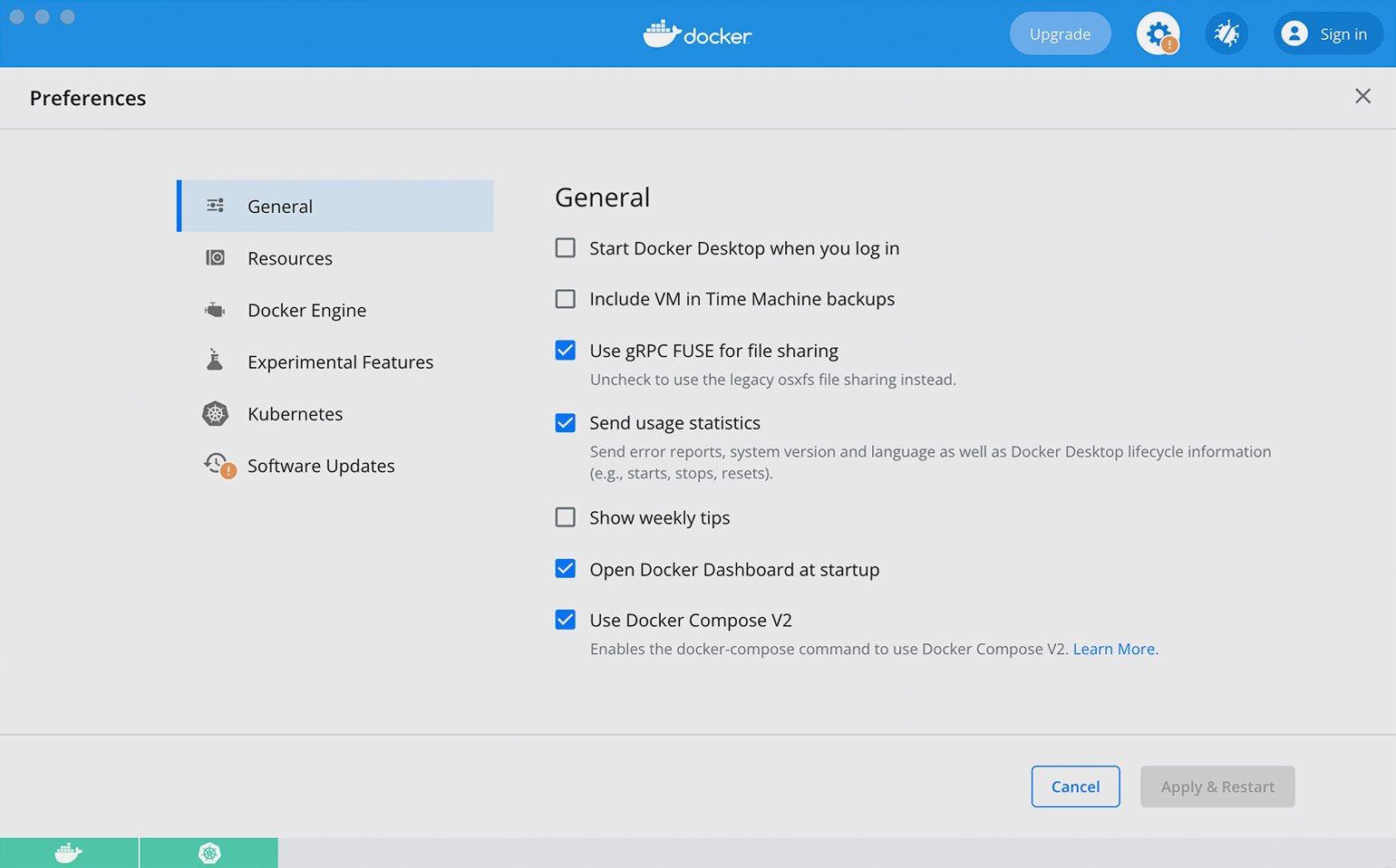


Рисунок 5 — Интерфейс Docker

Как упоминалось ранее, Docker изначально создавался под Linux. Поэтому на Windows и macOS запускают виртуальную машину с Linux, а поверх неё — Docker. В macOS используют VirtualBox, а в Windows — Hyper-V.

Работа поверх виртуалок повышает потребление ресурсов. Поэтому Docker на macOS и Windows работает медленнее и с рядом ограничений. Для разработки это приемлемо, но «в бою» так делать никто не будет. На всех популярных платформах в проде используют Linux.