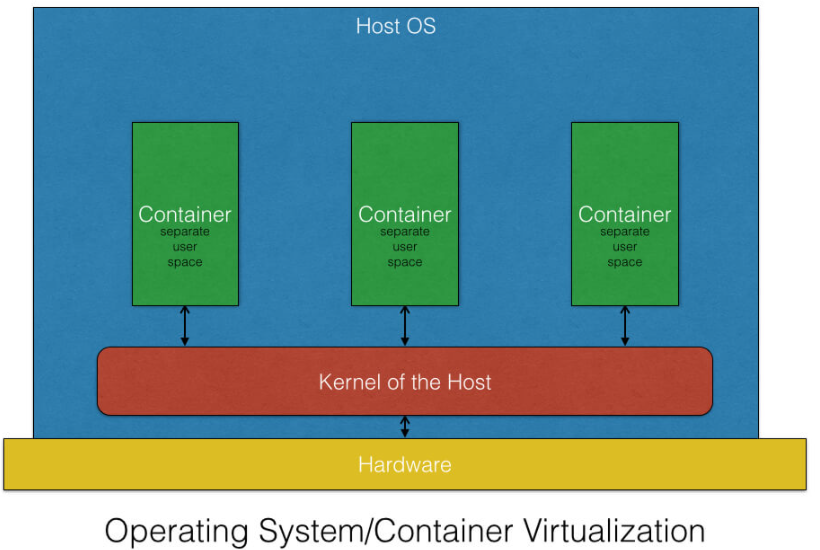
Технология контейнеризации — это ещё одна форма виртуализации ОС, предлагающая изоляцию приложений в пользовательских пространствах (контейнерах). Все контейнеры используют одну и ту же операционную систему. Благодаря технологии контейнеризации можно запускать приложение с нужными библиотеками в типовом контейнере, который соединяется с хостом или другой внешней компонентой при помощи простого интерфейса.

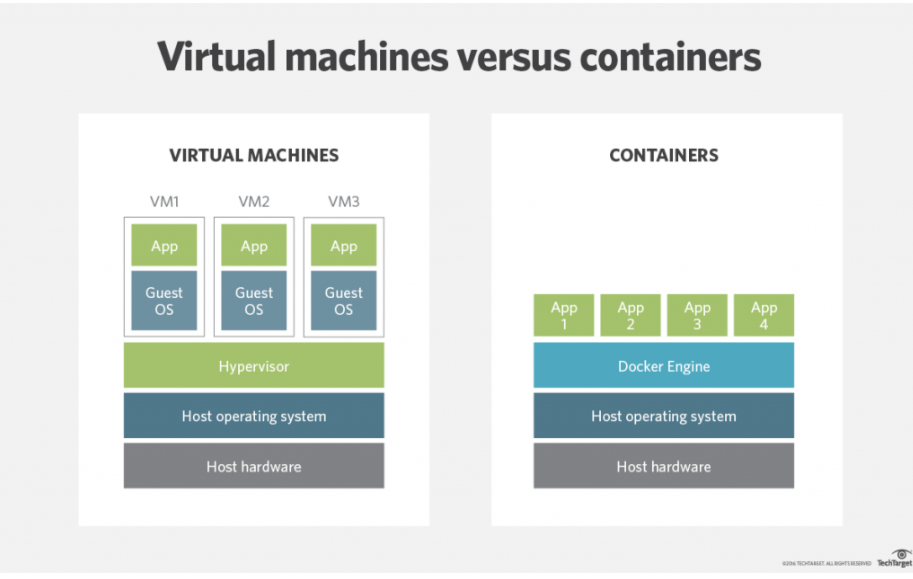
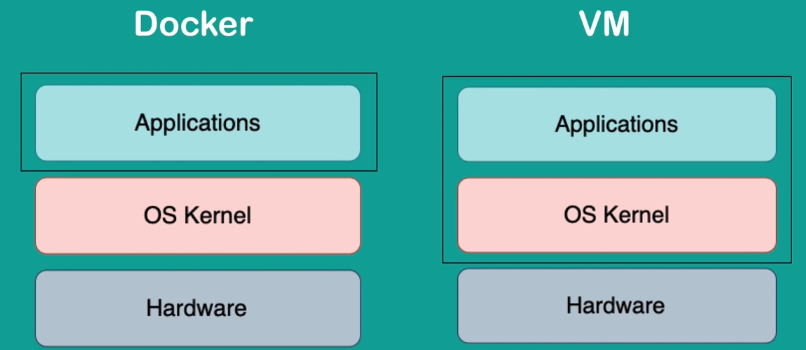
Все компоненты, необходимые для работы приложения (код, среда запуска, системные инструменты, библиотеки и настройки), упаковываются в один образ и могут быть использованы повторно в рамках текущей задачи или для любых других. Контейнер независим от ресурсов и архитектуры хоста. Он создаёт изолированную среду для приложения, не используя CPU, RAM или хранилище хостовой ОС. Все процессы идут внутри.



**Виртуальная машина и контейнер: в чём разница**

Виртуальные машины (ВМ) и контейнеры отличаются друг от друга. Это два разных подхода к созданию независимых изолированных вычислительных сред на физическом сервере. В чём особенность каждого варианта?

* **Виртуальная машина**. Требуется гипервизор, для каждой ВМ используется собственная гостевая ОС. Позволяет создавать неоднородные вычислительные среды на одном компьютере. Из-за собственной ОС ВМ может занимать несколько ГБ, а запуск ОС и всех приложений занимает какое-то время.
* **Контейнер**. Даже несколько контейнеров используют ядро одной хостовой ОС. Позволяет создавать на одном компьютере только однородные вычислительные среды. Намного легче ВМ, размер измеряется в Мб. Способен запускаться почти мгновенно. По сути docker виртуализирует только слой Application, в то время как VirtualBox, … виртуализирует слой ядра + application

Из за того, что docker не виртуализирует ядро, а использует ядро хостовой ОС, то он не заработает на Windows, так как docker контейнеры основаны на Linux. Что бы было возможным использовать docker на винде, нужно скачать WSL2 (windows subsystem for linux)

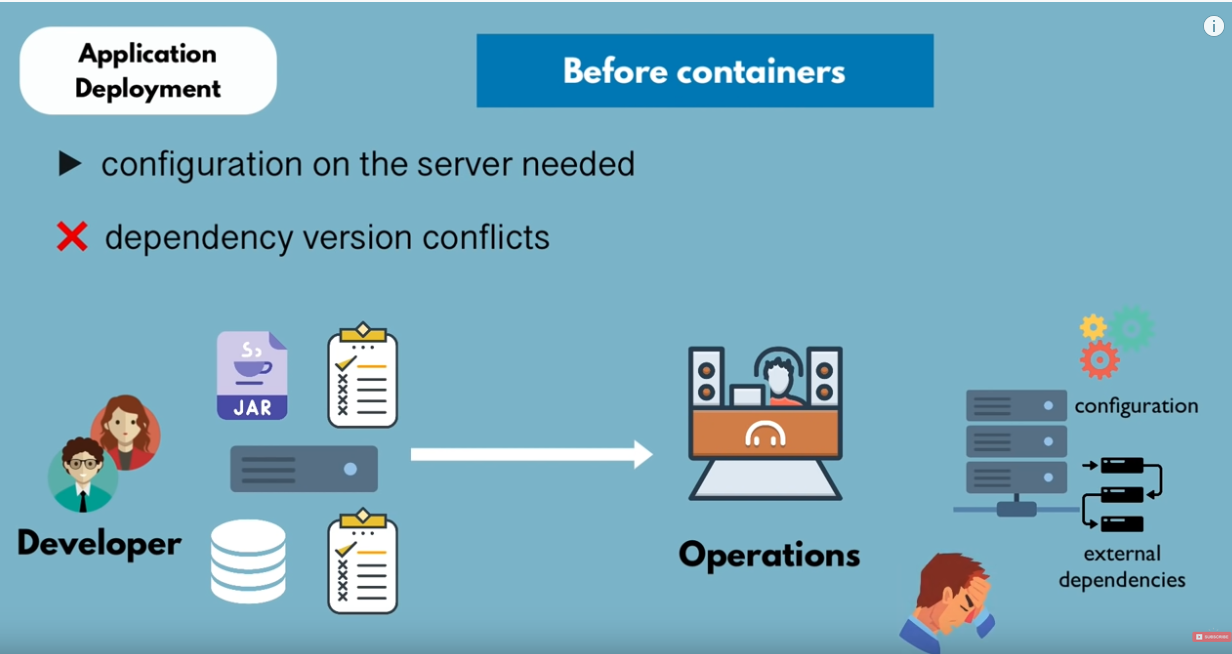
## Для чего нужны контейнеры

Это удобное решение для тестирования и разработки. Когда разработчик запускает свой или чей-то код в тестовой среде, могут возникать ошибки из-за изменения среды приложения. Топология сети, политики безопасности, вычислительные ресурсы также могут влиять на работу приложения. Контейнер самодостаточен и легко пересоздаётся, если нужно откатить изменения или провести ещё одно тестирование.

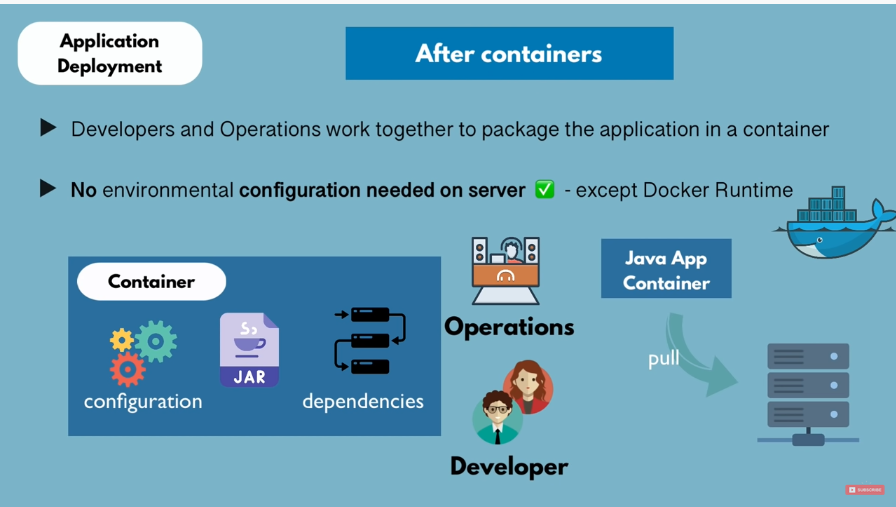
До контейнеров, разработчику нужно было локально развернуть такое же окружение, как было на бэкенде (базы данных, веб сервер, …), что вело к длительному развертыванию, из за того что для каждой ОС свои шаги инсталяции, человек может допустить ошибку во время инсталяции и все могло начаться с начала. С приходом контейнеризации, можно просто упаковать все сервисы в 1 контейнер и запустить его. И запуск будет одинаковым на всех ОС. Так же из за того что контейнеры изалированы, то можно иметь несколько версий одной и той же прилаги

Что представляет собой контейнеризация с точки зрения используемых технологий? Для создания контейнеров используются е технологии, как Linux XC, OpenVZ, Linux VServer, BSD Jails и Solaris. Первая популярная технология контейнеризации в Linux — это OpenVZ, превратившаяся позднее в более совершенный коммерческий продукт Virtuozzo.

**Развертывание созданного приложения до контейнеров**. Разработчики должны были создавать инструкции что и как сделать, прикрепляя конфигурационные файлы. После разрабы отдавали эти артефакты тем ребятам, которые разворачивали созданное приложение. Это приводило к ошибкам. Например, инструкции были не точны, могло быть расхождение в версиях и тд.



**Развертывание созданного приложения контейнерами**. Так как все зависимости упаковываются в 1 контейнер, то развертывание стало проще



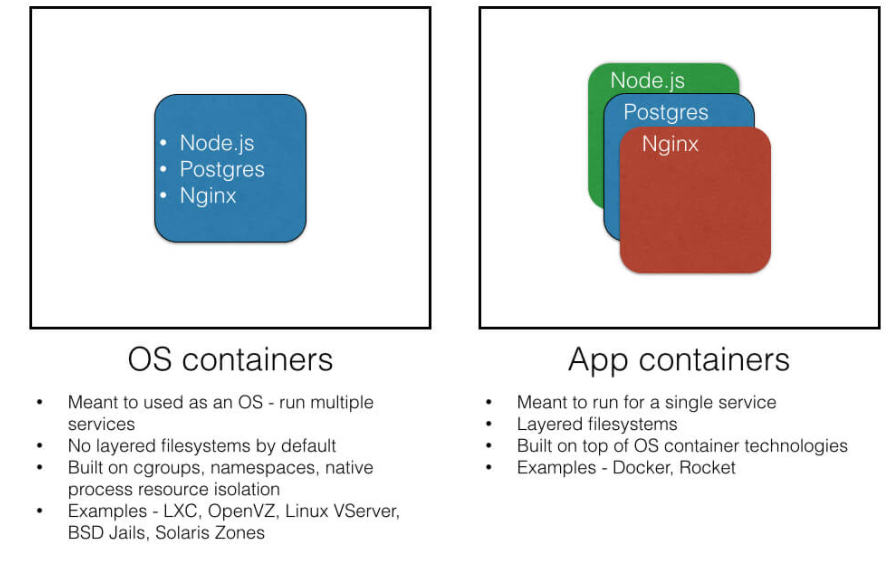
## Плюсы контейнеризации

* **Скорость создания**. Контейнер можно создать быстрее, чем ВМ. При этом среда контейнеризации для некоторых задач даёт больше возможностей.
* **Экономичность**. Контейнер занимает меньше места в хранилище, что уменьшает накладные расходы.
* **Высокая производительность**. Отсутствие межсетевых зависимостей и конфликтов повышает производительность разработки. Каждый контейнер фактически представляет собой микросервис, который можно независимо обновлять, не задаваясь вопросом синхронизации.
* **Управление версиями**. Можно мониторить версионность контейнеров, следить за различиями между ними.
* **Возможность миграции среды вычислений**. Все зависимости приложений и ОС, необходимые для работы приложения, инкапсулируются. Это позволяет без труда переносить образ контейнера из одной среды в другую. Так, один образ можно запускать в среде Windows и Linux или dev/test/stage.
* **Стандартизация**. Как правило, контейнеры создаются на основе открытых стандартов. Поэтому с ними можно работать в большинстве дистрибутивов Linux, Microsoft, MacOS.
* **Безопасность**. Контейнеры изолированы друг от друга и базовой инфраструктуры. Изменение/обновление/удаление одного контейнера не влияет на другой.

## Недостатки технологии

* **Высокая сложность**. Рост количества контейнеров, работающих с приложением, влияет на сложность управления ими. В производственной среде для работы с множеством контейнеров стоит использовать оркестраторы. Например, [Kubernetes](https://www.cloud4y.ru/cloud-services/kubernetes-as-a-service/) и Mesos.
* **Разрастание**. Нередко в контейнеры упаковывается гораздо больше ресурсов, чем реально требуется. Из-за этого образ разрастается, занимая больше места на диске.
* **Поддержка Native Linux**. Docker и многие другие контейнерные технологии основаны на Linux-контейнерах (LXC). Из-за этого запуск контейнеров в Windows-среде не всегда удобен, а ежедневное использование сложнее, чем при работе в Linux.
* **Недостаточная зрелость**. Технологии контейнеризации приложений появились на рынке сравнительно недавно. Не всегда удаётся сразу решить возникшую проблемы. Иногда требуется время на поиск решения.

Несмотря на то, что контейнеров может быть много, они, как правило, короткоживущие. Docker-контейнеры, например, часто называют одноразовыми. Можно использовать его, получить результат, а затем удалить и запустить точно такой же.



## Контейнеры как инструмент реализации архитектуры микросервисов

Один из способов получить максимальную отдачу от микросервисов – применять лучшие в отрасли технологии для управления. Все больше разработчиков используют [инструмент контейнерной оркестровки Kubernetes](https://www.cloud4y.ru/blog/kubernetes-business/). Он повышает эффективность работы микросервисов и помогает контейнерам и приложениям проще взаимодействовать. С его помощью удобно управлять приложениями, перемещать их в различные облачные или локальные внутренние среды. Платформа Kubernetes также предлагает лучшую в своем классе безопасность, обеспечивая согласованный подход к доступу и управлению даже в гибридной среде.

Максимально эффективное использование контейнеров и микросервисов достигается за счет гибкого подхода. Это включает в себя использование DevOps и внедрение автоматизации в вашей среде. Для достижения этих целей Kubernetes является обязательным условием. Однако, будучи технологией с открытым исходным кодом, которая постоянно совершенствуется, настройка и управление контейнерной платформой – это отдельный навык. Развернуть кластер Kubernetes можно в облаке или локально, на своей инфраструктуре. Ключевым моментом здесь является поиск [надежного партнера](https://www.cloud4y.ru/), который будет сопровождать вас в этом вопросе.

Microservices architectures for application development evolved out of this container boom. With containers, applications could be broken down into their smallest component parts or “services” that serve a single purpose, and those services could be developed and deployed independently of each other instead of in one monolithic unit. For example, let’s say you have an app that allows customers to buy anything in the world. You might have a search bar, a shopping cart, a buy button, etc. Each of those “services” can exist in their own container, so that if, say, the search bar fails due to high load, it doesn’t bring the whole thing down. And that’s how you get your Prime Day deals today.

**Uses for VMs vs Containers**

Both containers and VMs have benefits and drawbacks, and the ultimate decision will depend on your specific needs.

VMs are a better choice for running apps that require all of the operating system’s resources and functionality when you need to run multiple applications on servers, or have a wide variety of operating systems to manage. If you have an existing monolithic application that you don’t plan to or need to refactor into microservices, VMs will continue to serve your use case well.

Containers are a better choice when your biggest priority is maximizing the number of applications or services running on a minimal number of servers and when you need maximum portability. If you are developing a new app and you want to use a microservices architecture for scalability and portability, containers are the way to go. Containers shine when it comes to cloud-native application development based on a microservices architecture.

You can also run containers on a virtual machine, making the question less of an either/or and more of an exercise in understanding which technology makes the most sense for your workloads.

In a nutshell:

* VMs help companies make the most of their infrastructure resources by expanding the number of machines you can squeeze out of a finite amount of hardware and software.
* Containers help companies make the most of the development resources by enabling microservices and DevOps practices.

