**Конова Е. В.**

**Dependencies management**

***git branch name: dependencies***

**[0.5] What is Docker, and how it differs from dependencies management systems? From virtual machines?**

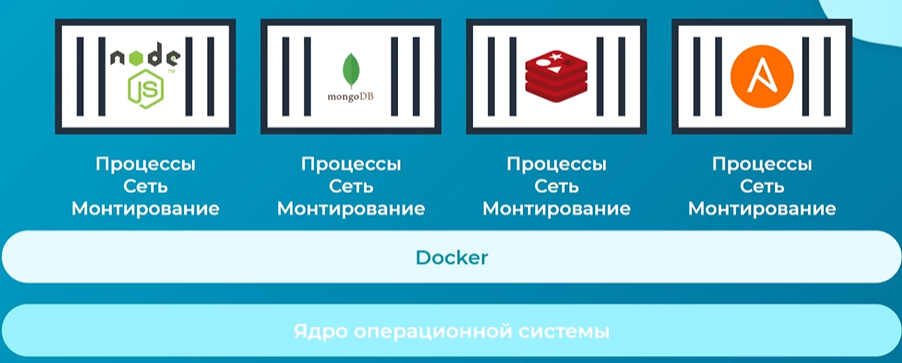
Docker не является системой управления зависимости или менеджером пакетов, потому что docker - это платформа контейнеризации с открытым исходным кодом, с помощью которой можно автоматизировать создание приложений, их доставку и управление. Платформа позволяет быстрее тестировать и выкладывать приложения, запускать на одной машине требуемое количество контейнеров. Он не только работает с зависимостями, но и является полноценной платформой для развертывания необходимого софта на его основе.



Docker позволяет избежать проблем с совместимостью с ОС или совместимостью с зависимостями, например, если софт в проекте нужно обновить или если в разработке используются разные ОС или разработчики предпочитают разные ОС, таким образом, решаются проблемы «матрицы совместимости».

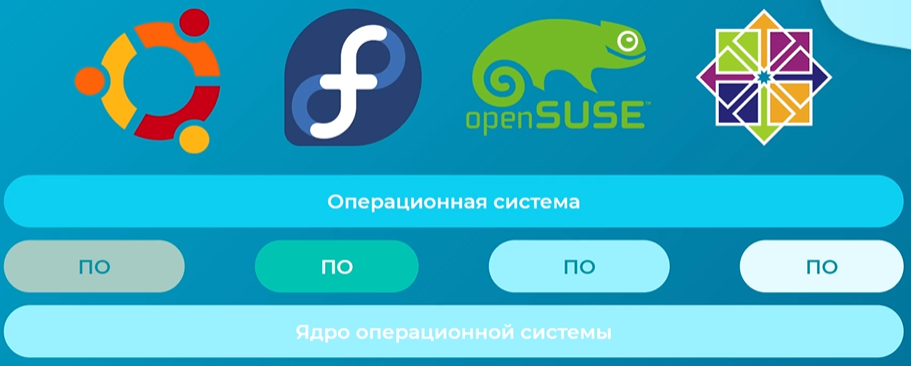
Виртуализация напоминает отдельный компьютер со своим оборудованием и ОС, внутри которого можно запустить еще одну ОС. А контейнеризация предполагает, что виртуальная среда запускается из ядра ОС, не предусматривает виртуализации оборудования и снижает потребление ресурсов.

Контейнеры — это способ стандартизации развертки приложения и отделения его от общей инфраструктуры. Экземпляр приложения запускается в изолированной среде, не влияющей на основную операционную систему.



Контейнеры похожи на виртуальные машины, за исключением того, что все они используют одно ядро операционной системы.

Для операционной системы схема взаимодействия слоев выглядит примерно так:



Для Docker ситуация выглядит вот так:



Идея в том, что Docker может запускать поверх себя любую версию ОС, если все они основаны на одном ядре (например, на основе Линукс). Но Docker может запускать контейнер на основе другого дистрибутива, т.к. используется его базовое ядро хоста, которое работает с другими системами.

В отличии от гипервизора, докер не предназначен для запуска различных ОС и ядер и виртуализации на одном оборудовании. Различия в одной картинке:



У каждой ВМ внутри своя система, толстый слой зависимостей, а затем – приложения. Это приводит к увеличению потребления ресурсов, поскольку нужно крутить несколько ОС с их ядрами. Еще ВМ тяжелые и потребляют много дискового пространства. Виртуальные машины дольше загружаются, потому что для них требуется загрузка всей операционной системы.

**[0.5] What are the advantages and disadvantages of using containers over other approaches?**

Плюсы вытекают из предыдущего абзаца:

* 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.

Минусы довольно субъективны, могу выделить следующее:

* Высокая сложность. Рост количества контейнеров, работающих с приложением, влияет на сложность управления ими. В производственной среде для работы с множеством контейнеров стоит использовать оркестраторы.
* 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.

**[0.5] Explain how Docker works: what are Dockerfiles, how are containers created, and how are they run and destroyed?**

Контейнер состоит из операционной системы, пользовательских файлов и метаданных. Как мы знаем, каждый контейнер создается из образа. Этот образ говорит docker-у, что находится в контейнере, какой процесс запустить, когда запускается контейнер и другие конфигурационные данные. Docker образ доступен только для чтения. Когда docker запускает контейнер, он создает уровень для чтения/записи сверху образа (используя union file system, как было указано раньше), в котором может быть запущено приложение.

Или с помощью программы docker, или с помощью RESTful API, docker клиент говорит docker демону запустить контейнер. Docker, по порядку, делает следующее:

* **скачивает образ ubuntu:** docker проверяет наличие образа ubuntu на локальной машине, и если его нет — то скачивает его с [Docker Hub](https://hub.docker.com/). Если же образ есть, то использует его для создания контейнера;
* **создает контейнер:** когда образ получен, docker использует его для создания контейнера;
* **инициализирует файловую систему и монтирует read-only уровень:** контейнер создан в файловой системе и read-only уровень добавлен образ;
* **инициализирует сеть/мост:** создает сетевой интерфейс, который позволяет docker-у общаться хост машиной;
* **Установка IP адреса:** находит и задает адрес;
* **Запускает указанный процесс:** запускает ваше приложение;
* **Обрабатывает и выдает вывод вашего приложения:** подключается и логирует стандартный вход, вывод и поток ошибок вашего приложения, что бы вы могли отслеживать как работает ваше приложение.

Для удаления контейнеров используется команда docker container rm или просто docker rm.

Для удаления запущенных контейнеров необходимо предварительно остановить их. Конечно, можно воспользоваться --force, но это может привести к повреждениям данных, с которыми работает приложение. Всегда лучше сначала остановить контейнеры с помощью команды docker stop.

За удаление образов Docker отвечает две команды: docker rmi и docker image rm. Они идентичны и работают примерно также, как и docker rm. Еще можно удалить тома. За удаление томов в командной строке отвечает docker volume rm.

**[0.25] Name and describe at least one Docker competitor (i.e., a tool based on the same containerization technology).**

Docker стал стандартом де-факто в мире контейнеров. Часто, когда говорят про контейнеры, подразумевают именно Docker-контейнеры. Но это не единственная реализация технологии контейнеризации, есть и другие:

* **PodMan**

Имеет интерфейс командной строки, который очень похож на команды Docker. Основное отличие от докера заключается в том, что у Podman нет отдельного демона, это самостоятельная утилита. Podman используется как инструмент управления контейнерами по умолчанию в дистрибутиве Fedora Linux.

* **LXC**

Система виртуализации на уровне ОС, а не самостоятельная платформа, как Docker. Она создает отдельное виртуальное окружение с собственным пространством процессов и сетевым стеком, в котором все контейнеры используют один экземпляр ядра ОС. LXC часто рассматривается как среднее между chroot и полноценной виртуальной машиной.

**[0.25] What is conda? How it differs from apt, yarn, and others?**

Цитируем определение Conda с официального блога: Conda — это менеджер пакетов с открытым кодом и система управления средой, которая работает на Windows, macOS и Linux. Conda проста в установке, выполнении и обновлении пакетов и зависимостей. Conda легко создает, сохраняет, загружает и переключается между средами на локальном компьютере. Она задумывалась для программ на Python, но может создавать пакеты и дистрибутивы программного обеспечения на любом языке.

Conda & apt

What you might want to do if you need a specific version of Python for a particular project is making a 'virtual environment'. Basically, that means that pip packages are installed within the project folder rather than in your bin folder somewhere on your computer.

Virtual environment can also link to a version of python using something like virtualenv --python=/usr/bin/python2.6.

apt install python=3.6 will install in the standard bin folder of your distro.

conda install python=3.6 will check in which environment you currently are and install it there. It of course requires Anaconda installed and setup on your computer.

There are a lot of virtual environment management packages out there and I am not going to give an opinion on which is the best.

Note that if you install it using apt install, the version used in command line for python3 or python may be ambiguous, to be sure, you can specify the full path or make an alias for that path if there isn't one.