

Контейнеризация на примере Docker

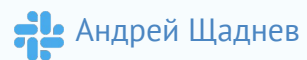


Андрей
Щаднев



Андрей Щаднев


Senior Engineer, Tele2





План занятия

1. [Причины популярности контейнеризации](#)
2. [Основные концепты контейнеризации](#)
3. [Операции с готовыми контейнерами и образами. Модели использования](#)
4. [Итоги](#)
5. [Домашнее задание](#)



Причины популярности контейнеризации



Причины популярности контейнеризации

Основная причина


Повышение *качества* и *уровня автоматизации* релиза программного обеспечения: вместо обновления версии ПО на окружении мы пересоздаем окружение.

Причины популярности контейнеризации

- Ваше приложение может быть в полной мере описано в одном манифесте - Dockerfile.
- Собранные из манифестов образы можно хранить в общедоступном (для команды или сообщества) реестре для совместной работы.
- Неизменяемые образы дают возможность протестировать на 100 % те же изменения на CI / QA окружениях и в результате установить тот же образ на продуктивное окружение.
- При возникновении проблем с новой версией - возможность вернуться к предыдущей с минимизацией рисков прерывания сервиса.

Причины популярности контейнеризации

- Возможна установка двух версий продукта одновременно в рамках одного и того же экземпляра ОС.
- Уверенность в полноценности поставки вашего ПО – не только исходный код, но и все необходимые зависимости в ожидаемых версиях.
- Образы разделены на слои, при внесении изменений в образ мы заменяем только затронутый слой.
- Концепт слоев дает возможность экономить ресурсы ПЗУ и сети.



Основные концепты контейнеризации



Основные концепты контейнеризации

Ключевой момент для понимания
концепты контейнеризации:
понятия **образа** и **контейнера**

Образ, контейнер, docker run



Образ - это файловая система, доступная только для чтения.



Контейнер - это инкапсулированный набор процессов, выполняемых в копии этой файловой системы для чтения и записи. Для оптимизации времени загрузки контейнера вместо обычного копирования используется функция копирования при записи.



Docker Run - запускает контейнер из заданного образа.

Сходство с ООП

Сходство с объектно-ориентированным программированием:

- **образы** концептуально подобны **классам**;
- **слои** концептуально похожи на **наследование**;
- **контейнеры** концептуально похожи на **экземпляры**.

Основной подход концепции контейнеризации

1. образ является неизменяемым - только для чтения;
2. мы создаем новый контейнер из образа;
3. затем мы вносим изменения в контейнер;
4. когда мы удовлетворены изменениями, мы преобразуем их в новый слой;
5. новый образ создается путем добавления нового слоя поверх базового образа.



Что представляют из себя образы?

Образы = **файлы + метаданные.**

Подробнее об образах

- **Файлы** образуют корневую файловую систему нашего контейнера.
- **Метаданные** могут указывать на ряд вещей (например: автор образа; команда для выполнения в контейнере при его запуске; устанавливаемые переменные окружения).
- **Образы состоят из слоев** в определенной логической последовательности; каждый слой может добавлять, изменять и удалять файлы и / или метаданные.
- **Образы могут совместно использовать слои** для оптимизации использования диска, времени передачи и использования памяти.

Примеры слоев образа

- CentOS;
- JRE;
- Wildfly;
- Зависимости
- JAR файл приложения
- Конфигурация

Пример Java web-приложения

- CentOS, как базовый слой;
- пакеты и конфигурации ОС слоя;
- JRE / JDK;
- Wildfly;
- зависимости приложения;
- код приложения и его данные;
- конфигурация приложения.

Возможные пространства имен для образов

- Официальные образы (созданные и поддерживаемые разработчиком продукта)
ubuntu, nginx, node
- Образы организаций и частных пользователей
mrepping/ponysay
- Образы внутреннего использования
docker.mycompany.com/jenkins/jenkins-ant:1.10.5

Для чего и где использовать теги образов?

- **Теги Docker образов схожи с Git-тегами.** Они представляют собой указатель на образ с соответствующим идентификатором.
- **Добавление тега не переименовывает образ,** а исключительно добавляет тег.
- **При публикации образа в реестре, адрес реестра становится тэгом.** Пример: при использовании собственного реестра: `docker.mycompany.com/jenkins/jenkins-ant:1.10.5` где `jenkins-ant` - это изначальное имя образа, а `1.10.5` - версия.

Для чего и где использовать теги образов?

Образы в Docker hub:

- **mpepping/ponysay** - это по сути:
`index.docker.io/mpepping/ponysay`
- **ubuntu** - это по сути:
`library/ubuntu` или `index.docker.io/library/ubuntu`

Для чего и где использовать теги образов?

- Образы могут обладать тегами для определения версий или вариантов образа.
- `Docker pull ubuntu` будет ссылаться на `ubuntu:latest`.
- Тег: `latest` часто является самым последним состоянием (зачастую не стабильным).

Для чего и где использовать теги образов?

Необходимо использовать теги:

- при выкате на продакшн;
- чтобы гарантировать, что одна и та же версия будет использоваться везде (на всех окружениях);
- чтобы получить воспроизводимый результат.

Не стоит использовать теги:

- при проведении экспресс-тестирования и прототипирования;
- при проведении экспериментов;
- когда вам нужна последняя версия.

Как корректно применять тег 'Latest?

- Убедитесь, что вы задали тег в целом и этот тег корректный.

Если тег не задан, то по умолчанию будет использован тег “latest”

- Проблема с тегом “latest” - никто не знает, на что он указывает:
 - последнее изменение в репозитории?
 - последний коммит в какой-то ветке? и какой именно?
 - последний созданный git тег в данном репозитории ?
 - какая-то произвольная версия?
- Если вы каждый раз перезаписываете “latest”, то теряете возможность отката на предыдущую версию с помощью Docker.
- Теги образов должны иметь осмысленные имена, то есть соответствовать ветвям кода, тегам или хэшам.



Операции с готовыми контейнерами и образами. Модели использования

Основные команды Docker

- **Docker run** - запускает выбранный образ в docker daemon;
- **Docker exec** - запуск команды в запущенном контейнере;
- **Docker stop** - остановка запущенного контейнера;
- **Docker rm** - удалить остановленный контейнер;
- **Docker build** - сборка образа из Dockerfile
- **Docker tag** - создание тэга
- **Docker push** - загрузка образа во публичный или приватный репозиторий.

Остановка запущенного Docker-контейнера

Два способа, которыми можно завершить работу контейнера:

- **удалить его с помощью команды `docker kill`** - мы немедленно останавливает контейнер;
- **остановить с помощью команды `docker stop`** - более изящный, этот способ посылает сигнал TERM, и если через 10 секунд контейнер не остановился, то посылает KILL.

Внимание: сигнал KILL не может быть перехвачен и принудительно завершает работу контейнера.

Список запущенных контейнеров

Список запущенных контейнеров:

- **docker ps** Показывает список запущенных контейнеров

Список включающий в себя остановленные контейнеры можно получить с опцией -a (--all) для **docker ps** .

Вывод диагностической информации контейнера:

- Используйте `docker attach`, если необходимо перенаправить поток данных в контейнер.
- Для получение диагностической информации из запущенного контейнера используйте `docker logs`.

```
docker logs --tail 1 --follow containerID
```



Список запущенных контейнеров

docker ps - показывает список запущенных контейнеров.

Список, включающий в себя остановленные контейнеры, можно получить с опцией -a (--all) для **docker ps**.

Вывод диагностической информации контейнера

- **Docker attach** - если необходимо перенаправить поток данных в контейнер;
- **Docker logs** - если хотите получить диагностическую информацию из запущенного контейнера.

`docker logs --tail 1 --follow containerID`

Подключение к запущенному Docker-контейнеру

Вы можете подключиться к контейнеру с помощью **docker attach containerID**. Обратите внимание:

- контейнер должен быть запущен;
- к одному контейнеру может быть подключено несколько клиентов;
- если вы не укажете `--detach-keys` при присоединении, то по умолчанию установиться `^P^Q`.

Отправка контекста сборки в Docker

- Контекст сборки - это рабочая директория, содержащая Dockerfile и дополнительные файлы (части сборки). Зачастую это текущая директория “.”, передается в команде при сборке docker образа.
- Содержание контекста сборки отправляется (в виде архива) клиентом Docker демону Docker.
- Это позволяет использовать удаленную машину для сборки с использованием локальных файлов.
- Чем больше дискового пространства занимает директория контекста сборки тем потенциально дольше займет сборка образа.



Переименование запущенных Docker-контейнеров

Вы можете переименовать контейнеры с помощью **docker rename**.

Это позволяет "освободить" имя, не удаляя текущий контейнер.



Docker exec vs Docker run

Docker exec - yourContainerName bash;

Docker run - yourImageName bash.

Docker exec vs Docker run

Пример на предыдущем слайде показывает ключевое различие:

- **Docker exec** - предназначен для выполнения бинарного файла, отличного от указанного в ENTRYPOINT (если он существует в манифесте образа), в работающем контейнере. Основной вариант использования – поиск неисправностей и анализ
- **Docker run** - если образ доступен локально, предназначен для запуска. В других случаях - предварительно скачивает образ и запускает контейнер из образа. Без дополнительных параметров запускается исполняемый бинарный файл, указанный в качестве точки входа для выполнения. Дополнительные параметры могут переопределить ENTRYPOINT для текущего контейнера и выполнить другой бинарный файл или тот же, но с другими параметрами.

Дополнительные параметры Docker run

- **docker run -it** для запуска интерактивном режиме.
Чаще всего используется для поиска неисправностей
- **docker run -d** для запуска в фоновом режиме

Docker run -v для сохранения и общего использования данных

```
docker run -v /local-data:/data-in-container --name container_name -d image-name
```

С помощью подключение локальной директории в контейнер можно реализовать:

- постоянное хранение данных, полученных в результате запущенного в контейнере приложения;
- совместное использование данных в двух и более запущенных контейнерах.

Создание и использование сети для запущенных контейнеров

```
docker network create network-name
```

```
docker run -d --name=container-name --net=network-name image-name
```

```
docker network connect network-name container-name
```

Для реализации сетевого соединения между контейнерами:

- создайте новую виртуальную сеть или используйте существующую;
- запустите новый контейнер из образа или подключите сеть к уже запущенному контейнеру.

Возможно подключение контейнера(ов) к одной или более сетей.



Итоги

Итоги

- Познакомились с основными системами управления виртуализацией, их классификацией, преимуществами и недостатками каждой из них, а также с понятием гипервизора и его типами.
- Узнали о комплексном подходе для виртуализации - об инфраструктуре как сервисе и его реализациях в виде публичных и частных облаков.



Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Андрей Щаднев