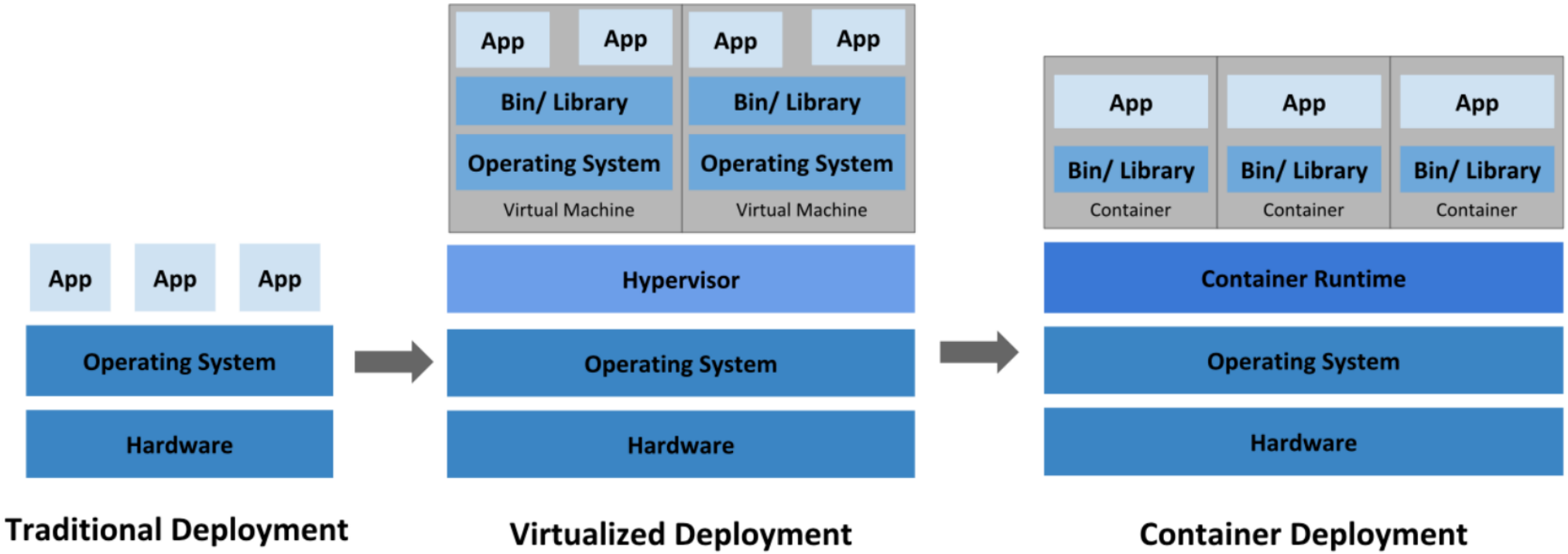
Docker и Kubernetes

Борисенко Глеб

ФТиАД2021

Немного истории



Преимущества Docker

- Компактность, простота и эффективность создания образа контейнера по сравнению с использованием образа виртуальной машины;
- Идентичная окружающая среда при разработке, тестировании и релизе: на ноутбуке работает так же, как и в облаке;
- Разделение задач между Dev и Ops: создавайте образы контейнеров приложений во время сборки/релиза, а не во время развертывания;
- Незаменим для микросервисной архитектуры: независимые приложения упаковываются в контейнеры, которые можно динамически развертывать и управлять;
- Безопасность – контейнеры запускаются в изоляции

Docker is an open platform for developing, shipping, and running applications.

Образ

- Образ контейнера (container image или docker image) - это стандартная единица программного обеспечения, в которую упаковано приложение со всеми необходимыми для его работы зависимостями — кодом приложения, средой запуска, системными инструментами, библиотеками и настройками;
- Образ контейнера становится контейнером в тот момент, когда мы его запускаем (например, выполняем docker run).

Пример типичной папки

- Dockerfile – как создать docker image
- Poetry.lock – зависимости
- App.py - приложение

Dockerfile

- Dockerfile – это то, по чему собирается образ. Просто содержит ряд инструкций, которые последовательно выполняются.
- Инструкции весьма простые, базовые, вроде копирования файлов в контейнер, выполнения команды баша и т.п.

Пример простого Dockerfile-а

```
1  FROM ubuntu:18.04
2  COPY . /app
3  RUN make /app
4  CMD python /app/app.py
```

Самые частые инструкции

- FROM – какой образ взять за базовый (на примере образ берется из Docker Registry, об этом позже)
- RUN – выполнить команду BASH-а во время сборки
- CMD – выполнить команду BASH-а при запуске контейнера (эта инструкция воспринимается только единственная, финальная из всех таких в файле) *Есть еще ENTRYPOINT, но тут лучше самим загуглить*
- COPY – скопировать файлы в образ
- ENV – задать переменную окружения
- Есть и другие, не менее важные, эти просто самые частые

Docker Registry и Dockerhub

- Docker Registry – просто хранилище образов. Только образов.
- Dockerhub – реестр докер образов, который содержит как образы выпущенные сторонними разработчиками, так и образы, выпущенные разработчиками docker.
- ubuntu – как раз оттуда
- 18.04 – это тег конкретного образа. Версия, иначе говоря.
- Если не указываете сами, то при сборке вашему образу присваивается тег **latest**

Основные команды docker image

Command	Description
docker image build	Build an image from a Dockerfile
docker image pull	Pull an image or a repository from a registry
docker image push	Push an image or a repository to a registry
docker image rm	Remove one or more images

Основные команды docker container

Command	Description
docker container stop	Stop one or more running containers
docker container rm	Remove one or more containers
docker container run	Run a command in a new container
docker container start	Start one or more stopped containers
docker container exec	Run a command in a running container

Порты

Порты задаем аргументом в команде docker run

```
docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Name, shorthand	Default	Description
`--publish , -p`		Publish a container's port(s) to the host

```
1 | docker run -p 3000:3000 image-name
```

Волюмес

- Container Volumes – это инструмент, который предоставляет возможность соединять какую-то папку внутри контейнера с папкой на хосте
- Есть два ~~стула~~ типа volumes:

	Named Volumes	Bind Mounts
Host Location	Docker chooses	You control
Mount Example (using `-v`)	my-volume:/usr/local/data	/path/to/data:/usr/local/data
Populates new volume with container contents	Yes	No
Supports Volume Drivers	Yes	No

Пример создания и использования

создаем новый Named Volume

```
1 | docker volume create new-volume
```

Передаем его в качестве аргумента команде docker run

```
1 | docker run -v new-volume:/etc/data image-name
```

Переменные окружения

- Переменные окружения можно задавать и при запуске контейнера

```
docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Name, shorthand	Default	Description
`--env , -e`		Set environment variables

```
1  docker run MySQL_example
2      -e MYSQL_ROOT_PASSWORD=secret `
3      -e MYSQL_DATABASE=example
```

Слои

- Команды RUN, COPY, ADD создают слои, которые кэшируются.
- При пересборке, если вы не трогали их, будет браться последний нетронутый слой

Stages

- Можно билдить образ за несколько стадий – несколько FROM-ов
- В одном сделать, например, одну сборку, в другом использовать ее (чтобы не засорять ненужными зависимостями образ)
- Собираться образ будет только по последней
- Если честно, в проме я ни разу не видел еще использования такого, все артефакты билдятся отдельными образами, просто запускаются они автоматически через оркестратор

Best practices

- Create ephemeral containers. Останавливать, уничтожать, ребилдить и замещать контейнер с абсолютно минимальным конфигурированием
- Understand build context. Build context – директория, где запускается сборка по Dockerfile-у. Не включайте лишние файлы в ваши образы.
- Exclude with .dockerignore. То же самое, что и .gitignore.
- Don’t install unnecessary packages
- Decouple applications. Не пытайтесь всунуть кучу задач в один контейнер, делите логически ваши приложения.
- Minimize the number of layers

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

А если контейнеров много?

- Если мы хотим поднимать и запускать сразу группу контейнеров, нам поможет **Docker Compose**:

1. Определяем Dockerfile-ы для приложений
2. Определяем необходимые сервисы в docker-compose.yml
3. Запускаем docker-compose

Пример docker-compose.yml

```yaml
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

А что делать с сетью?

Networking

Remember that containers, by default, run in isolation and don't know anything about other processes or containers on the same machine. So, how do we allow one container to talk to another? The answer is **networking**. Now, you don't have to be a network engineer (hooray!). Simply remember this rule...

> If two containers are on the same network, they can talk to each other. If they aren't, they can't.

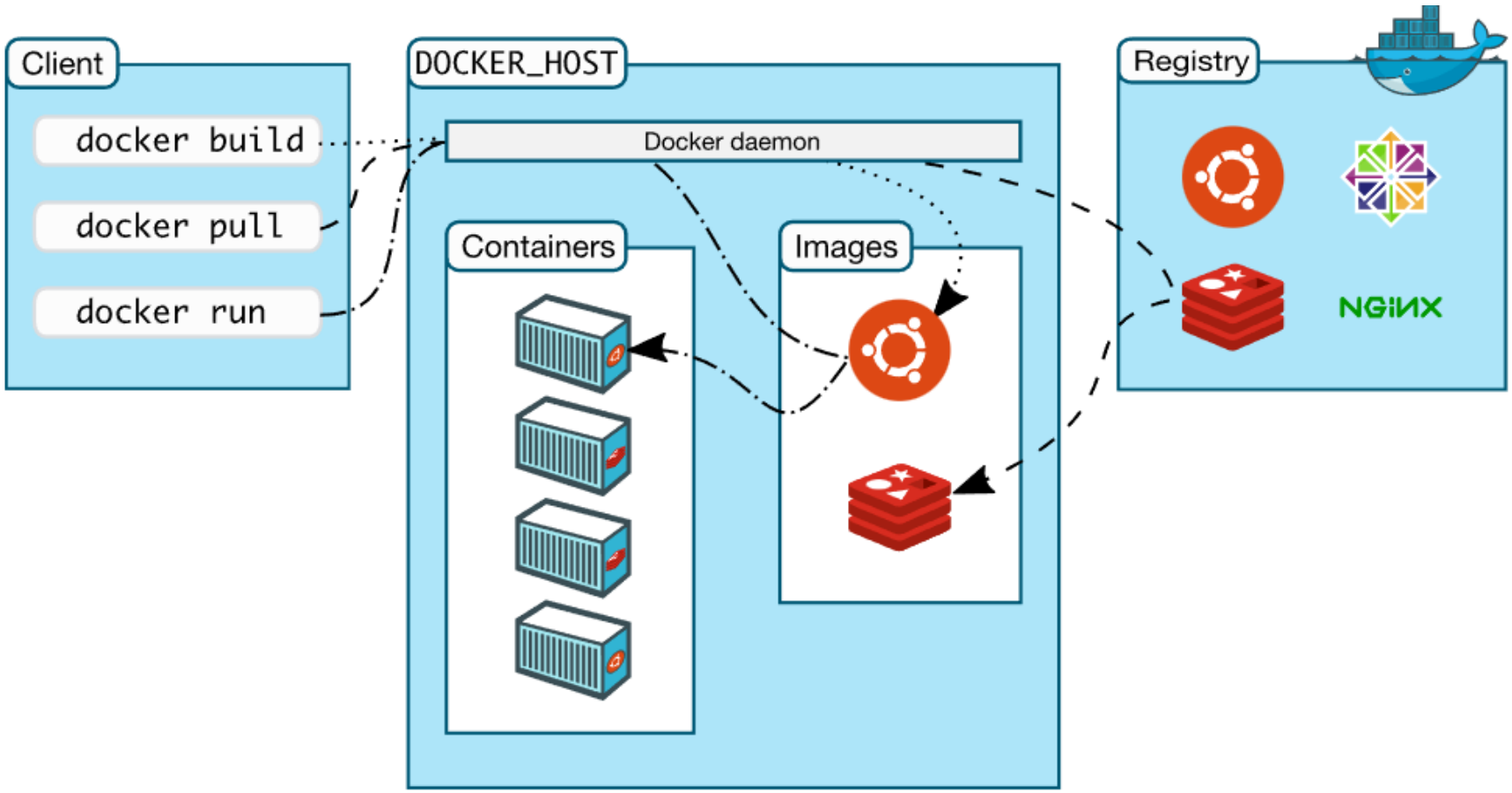
Networking

- По дефолту, Compose создает отдельную сеть для вашего приложения. Каждый контейнер для сервиса находится в ней и как доступен другим контейнерам, так и они доступны ему.
- Сетевой доступ к конкретному контейнеру обеспечивается очень просто – по домену, которым является название контейнера
- Сети можно настраивать, у них есть разные типы, все это можно найти в доке. *Но вам скорее всего будет хватать дефолтной.*

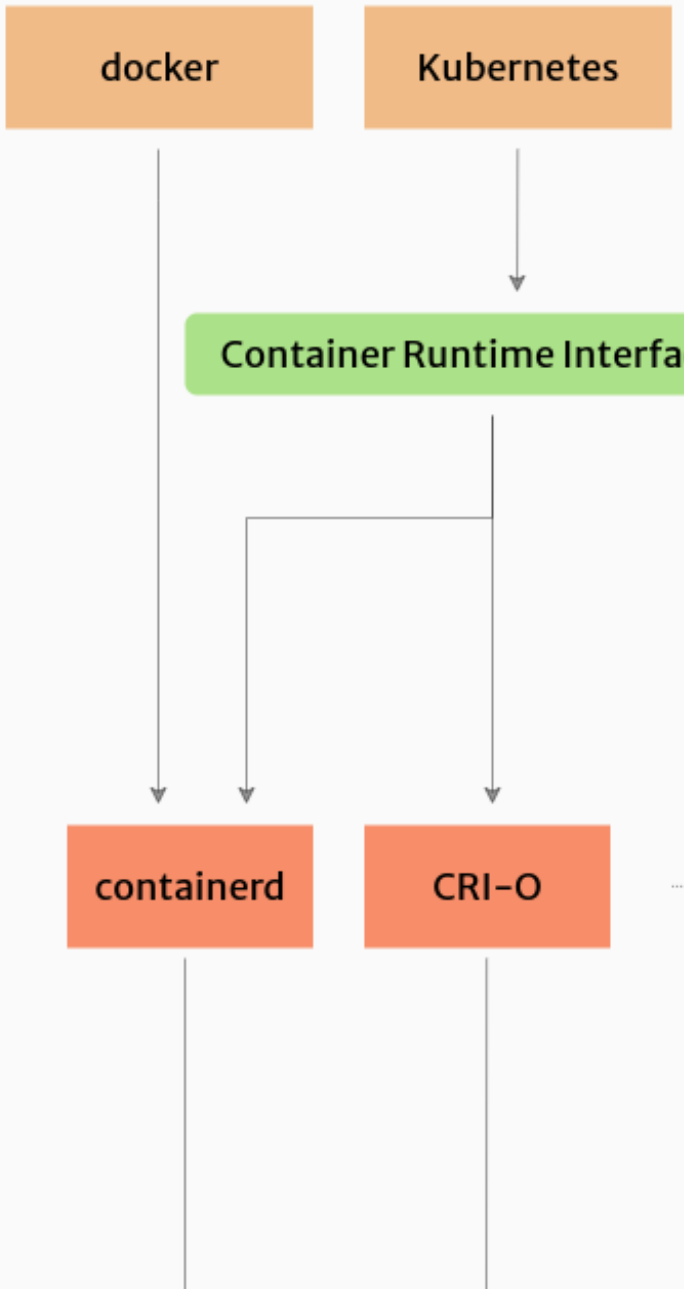
Как выглядит деплой тогда?

- Клонируем гит с вашем проектом либо пушим образы нужные в docker registry/dockerhub
- Запускаем Docker Compose
- Готово

А как вообще работает докер по сути



- Существует разница между компанией Docker, контейнерами Docker, образами Docker и инструментами Docker. Важно понимать, что Docker — лишь один из инструментов для работы с контейнерами, существуют и другие инструменты
- Основные стандарты:
  - **Container Runtime Interface (CRI)** определяет API между Kubernetes и Container Runtime (средой выполнения контейнеров).
  - **Open Container Initiative (OCI)** определяет стандарт образов и контейнеров.



These are the tools that you use to run containers, as a developer, or in production.

CRI is a Kubernetes API.

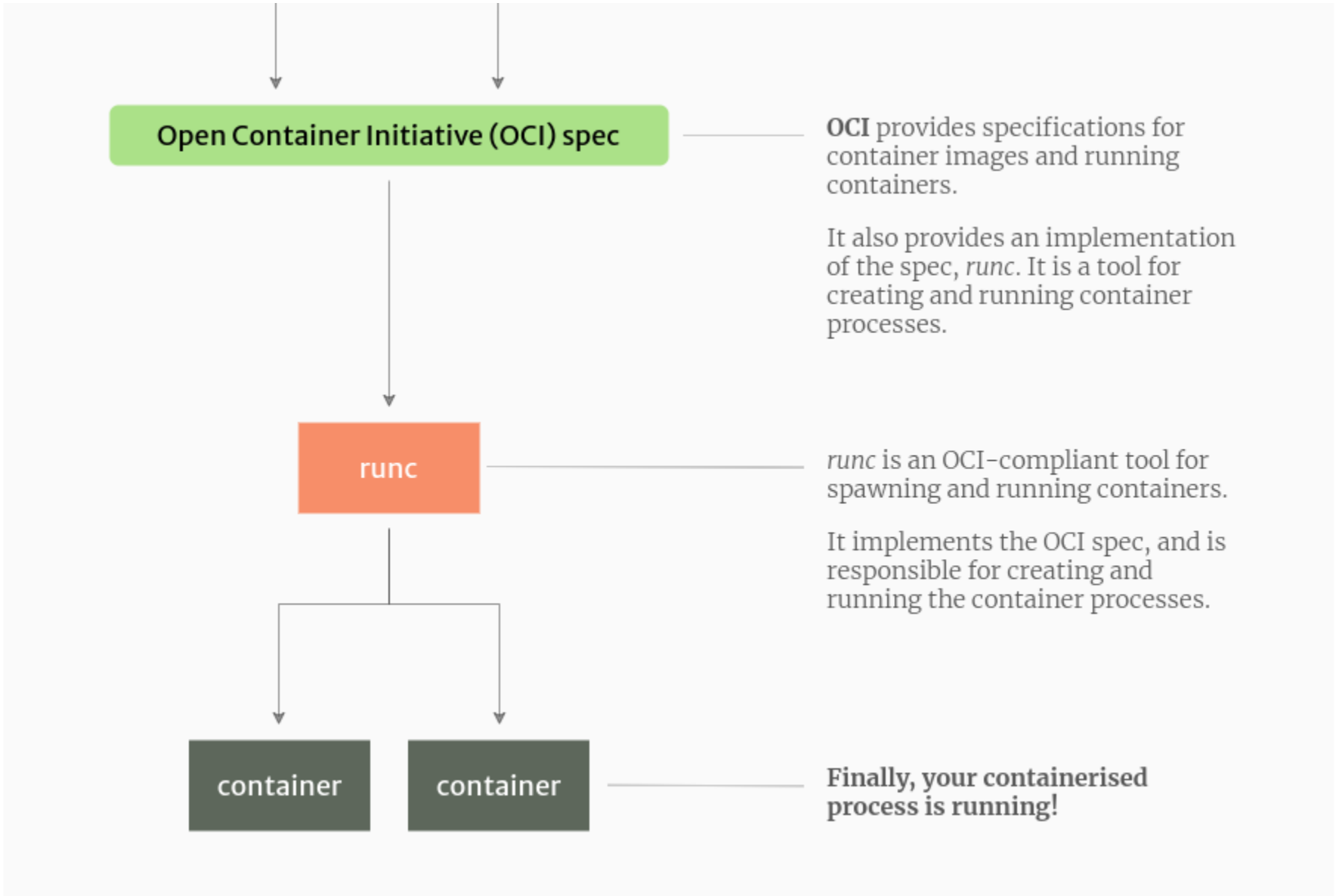
It defines the way that Kubernetes interacts with different container runtimes.

Because it's standardised in a spec, so you can choose which CRI implementation you want to use. Or write your own.

You can choose from a couple of runtimes that implement the CRI spec.

containerd came from Docker. It's made CRI-compliant through its *cri* plugin.

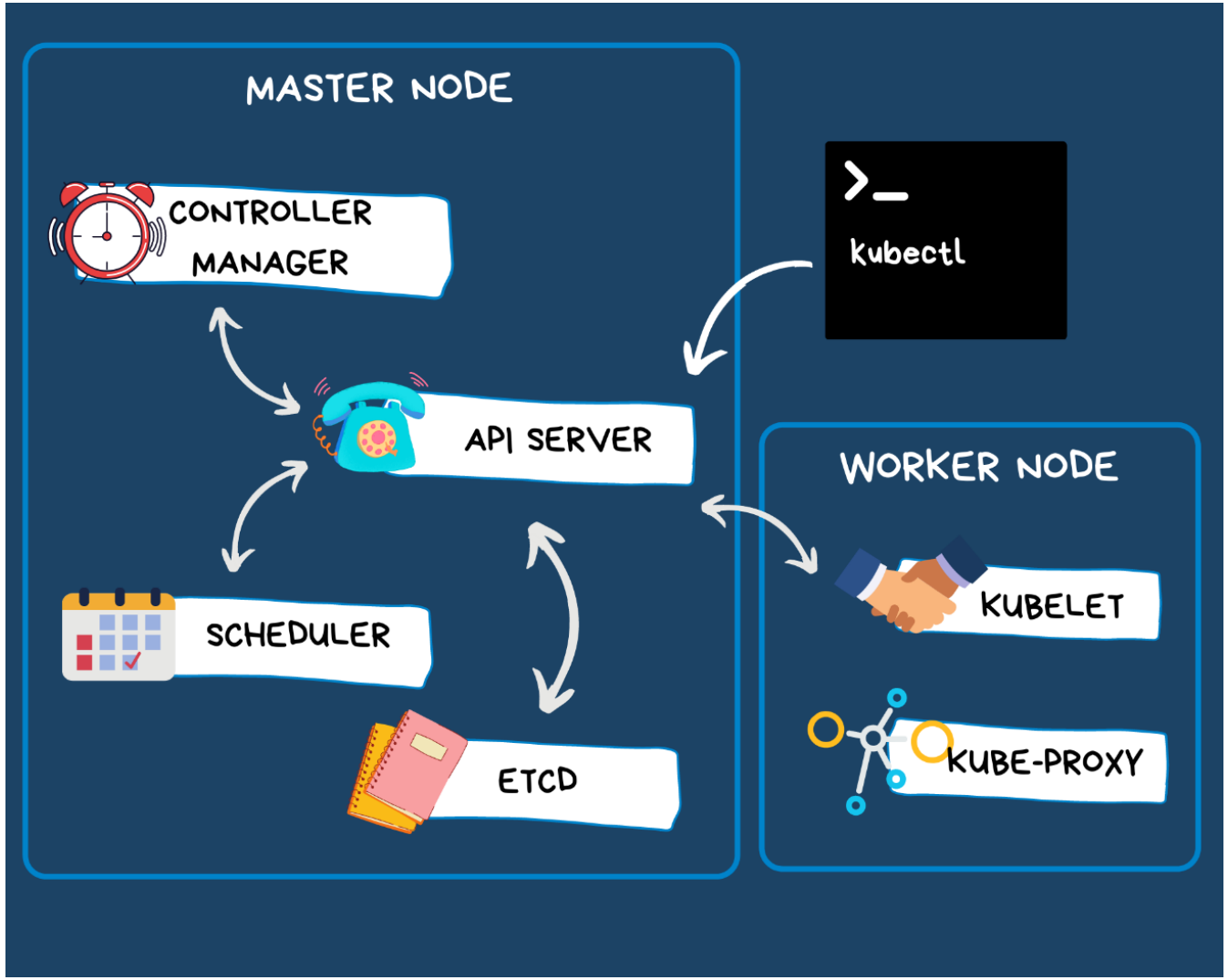
CRI-O was developed as a CRI implementation, from Red Hat/IBM/etc.


Open Container Initiative (OCI) spec
OCI provides specifications for container images and running containers.
It also provides an implementation of the spec, runc. It is a tool for creating and running container processes.
runc
runc is an OCI-compliant tool for spawning and running containers.
It implements the OCI spec, and is responsible for creating and running the container processes.
container
container
Finally, your containerised process is running!

Кубернетес, Kubernetes, или K8s

- Kubernetes – платформа для оркестрации контейнеров и автоматизации некоторых действий с ними
- Kubernetes кластер – кластер из хостов, на которых развернута платформа и на которых и поднимаются контейнеры
- Де-факто стандарт промышленной среды на сегодня
- Kubernetes на одном хосте имеет особое название – minicube
- Архитектура платформы есть на их сайте, выглядит круто
- Есть прикольный сайт про основы: https://gochronicles.com/tag/kubernetes-101/

Верхнеуровневая архитектера



Основные понятия/концепции

- **Nodes**: Нода это машина в кластере Kubernetes.
- **Pods**: Pod это группа контейнеров с общими разделами, запускаемых как единое целое.
- **Replication Controllers**: replication controller гарантирует, что определенное количество «реплик» pod'ы будут запущены в любой момент времени.
- **Services**: Сервис в Kubernetes это абстракция которая определяет логический объединённый набор pod и политику доступа к ним.
- **Volumes**: Volume(раздел) это директория, возможно, с данными в ней, которая доступна в контейнере.
- **Labels**: Label'ы это пары ключ/значение которые прикрепляются к объектам, например pod'ам. Label'ы могут быть использованы для выбора наборов объектов.
- **Kubectl Command Line Interface**: kubectl интерфейс командной строки для управления Kubernetes.

С чем вообще помогает кубернетес

- Упала пода – поднимает сразу новую (*точнее пытается, и только если это настроено*), настраивает сетевой доступ к ней, подключает волюмы как надо
- Удобное управление конфигами и секретами сразу для кластера
- Прозрачная аллокация ресурсов
- Есть namespaces – грубо говоря, отдельные группы, «отделы»
- Удобное, легкое, прозрачное масштабирование и балансировка

Ingress и Ingress Controller

- Ingress – правила, которые регулируют сетевой поток. Например, направлять пакеты, пришедшие на путь /mypathA, на сервис A.
- Ingress Controller – та штука, которые эти правила и выполняет.
- Ingress Controller – гаишник, а ingress – ПДД.
- Самый распространенный тип Ingress Controller – nginx, но есть и другие. Облака могут предлагать свои.

ReplicaSet и Deployments

- ReplicaSet – улучшенная версия Replication Controller, по сути, дающая больше опций по выбору подов
- Deployment – на первый взгляд, то же самое, что и ReplicaSet, но на самом деле это более верхний уровень абстракий. Основная суть – позволяет легче обновлять поды (автоматизированный кубером rolling update)

Helm

- Приложения развивались, какие-то переиспользовались кучу раз, и поэтому придумали ставить их как пакеты для кубера
- Helm – это диспетчер пакетов для Kubernetes
- У пакета для кубера есть свое название – chart (helm chart)
- Chart – это что-то вроде набора шаблонов и скриптов
- Мы просто находим нужное нам и ставим через helm install
- Примеры пакетов: wordpress, prometheus-operator, dask, etc.