***29.03.20***

Docker is an open platform for developing, shipping and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

<https://docs.docker.com/engine/docker-overview/>

Docker Engine overview

Docker Engine is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:

* A server with a long-running daemon process dockerd.
* APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
* A command line interface (CLI) client docker.

The CLI uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manage Docker objects, such as images, containers, networks, and volumes.

<https://docs.docker.com/install/>

# Get started

Docker for Windows: If you have a newer system, specifically 64bit Windows 10 Pro, with Enterprise and Education (1607 Anniversary update, Build 14393 or later), consider using [Docker Desktop for Windows](https://docs.docker.com/docker-for-windows) instead. It runs natively on the Windows, so there is no need for a pre-configured Docker QuickStart shell. It also uses Hyper-V for virtualization, so the instructions below for checking virtualization will be out of date for newer Windows systems.

Docker for my home version: Docker Toolbox on Windows

Docker Toolbox includes the following Docker tools:

* Docker CLI client for running Docker Engine to create images and containers.
* Docker Machine so you can run Docker Engine commands from Windows terminals.
* Docker Compose for running the docker-compose command.
* Kitematic, the Docker GUI.
* The Docker QuickStart shell preconfigured for a Docker command-line environment.
* Oracle VM VirtualBox.

Because the Docker Engine daemon uses Linux-specific kernel features, you can’t run Docker Engine natively on Windows. Instead, you must use the Docker Machine command, docker-machine, to create and attach to a small Linux VM on your machine. This VM hosts Docker Engine for you on your Windows system.

**Tip**: One of the advantages of the newer [Docker Desktop for Windows](https://docs.docker.com/docker-for-windows/) solution is that it uses native virtualization and does not require VirtualBox to run Docker.

Установка Докера - <https://docs.docker.com/toolbox/toolbox_install_windows/>

## **The Docker platform**

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don’t need the extra load of a hypervisor, but run directly within the host machine’s kernel.

**Гиперви́зор** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Hypervisor*; от [др.-греч.](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B5%D0%B2%D0%BD%D0%B5%D0%B3%D1%80%D0%B5%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) ὑπέρ «над, выше, сверх» + [лат.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *vīsio* «зрение; видение») или **монито́р виртуа́льных маши́н** (в [компьютерах](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80)) — [программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) или [аппаратная схема](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE), обеспечивающая или позволяющая одновременное, параллельное выполнение нескольких [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) на одном и том же [хост](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82)-компьютере. Гипервизор также обеспечивает изоляцию операционных систем друг от друга, защиту и безопасность, разделение ресурсов между различными запущенными [ОС](https://ru.wikipedia.org/wiki/%D0%9E%D0%A1) и управление ресурсами.

Гипервизор также обязан предоставлять работающим под его управлением на одном хост-компьютере ОС средства связи и взаимодействия между собой (например, через обмен файлами или сетевые соединения) так, как если бы эти ОС выполнялись на разных физических компьютерах.

Гипервизор сам по себе в некотором роде является минимальной операционной системой ([микроядром](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%8F%D0%B4%D1%80%D0%BE) или [наноядром](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%BD%D0%BE%D1%8F%D0%B4%D1%80%D0%BE" \o "Наноядро)). Он предоставляет запущенным под его управлением операционным системам службу [виртуальной машины](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0), виртуализируя или эмулируя реальное (физическое) аппаратное обеспечение конкретной машины. И управляет этими виртуальными машинами, выделением и освобождением ресурсов для них. Гипервизор позволяет независимое «включение», перезагрузку, «выключение» любой из виртуальных машин с той или иной ОС. При этом операционная система, работающая в виртуальной машине под управлением гипервизора, может, но не обязана «знать», что она выполняется в виртуальной машине, а не на реальном аппаратном обеспечении.

<https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D0%B2%D0%B8%D0%B7%D0%BE%D1%80> – также перечислены виды гипервизоров

**Виртуализация** – это технология создания представления нескольких компьютеров или серверов на базе одного физического компьютера, сервера или серверного кластера. Эта физическая машина называется **хостом;**у нее есть определенная конфигурация процессора, оперативной и дисковой памяти и т.д. Физические ресурсы с помощью специализированного ПО распределяются таким образом, чтобы развернуть несколько независимых друг от друга виртуальных машин.

По существу, виртуализация – не более чем иллюзия присутствия нескольких отдельных компьютеров, то есть виртуальных машин, на одном и том же физическом оборудовании. А создается эта иллюзия при помощи гипервизора.

**Гипервизор** – это программа, которая управляет физическими ресурсами вычислительной машины и распределяет эти ресурсы между несколькими различными операционными системами, позволяя запускать их одновременно.

**Ядро́** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *kernel*) — центральная часть [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) (ОС), обеспечивающая [приложениям](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) координированный доступ к ресурсам [компьютера](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), таким как [процессорное время](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D1%8F), [память](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BF%D0%BE%D0%BC%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B5%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE_%D1%81_%D0%BF%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%BC_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%D0%BE%D0%BC), внешнее [аппаратное обеспечение](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D0%BA%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%82%D1%83%D1%8E%D1%89%D0%B8%D0%B5), внешнее устройство ввода и вывода информации. Также обычно ядро предоставляет сервисы [файловой системы](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) и [сетевых протоколов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!

Docker provides tooling and a platform to manage the lifecycle of your containers:

* Develop your application and its supporting components using containers.
* The container becomes the unit for distributing and testing your application.
* When you’re ready, deploy your application into your production environment, as s container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

## **Docker Engine**

Docker Engine is a client-server application with these major components:

* A server which is a type of long-running program called a daemon process (the dockerd command).
* A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
* A command line interface (CLI) client (the docker command).

**Де́мон** (daemon, dæmon, [др.-греч.](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B5%D0%B2%D0%BD%D0%B5%D0%B3%D1%80%D0%B5%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) δαίμων [*демон*](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BC%D0%BE%D0%BD)) — [компьютерная программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) в системах класса [UNIX](https://ru.wikipedia.org/wiki/UNIX), запускаемая самой системой и работающая в [фоновом режиме](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D0%BD%D0%BE%D0%B2%D1%8B%D0%B9_%D1%80%D0%B5%D0%B6%D0%B8%D0%BC) без прямого взаимодействия с пользователем.

Демоны обычно запускаются во время загрузки системы. Типичные задачи демонов: [серверы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)) [сетевых протоколов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) ([HTTP](https://ru.wikipedia.org/wiki/HTTP), [FTP](https://ru.wikipedia.org/wiki/File_Transfer_Protocol), электронная почта и др.), управление оборудованием, поддержка очередей печати, управление выполнением заданий по расписанию и т. д. В техническом смысле демоном считается [процесс](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), который не имеет управляющего терминала. Чаще всего (но не обязательно) предком демона является [init](https://ru.wikipedia.org/wiki/Init" \o "Init) — корневой процесс UNIX. Традиционно названия демон-процессов заканчиваются на букву *d*, чтобы показать, что этот процесс является демоном, и для различия нормальной компьютерной программы и демона.

В операционных системах [Solaris 10](https://ru.wikipedia.org/wiki/Solaris_10" \o "Solaris 10) и [OpenSolaris](https://ru.wikipedia.org/wiki/OpenSolaris" \o "OpenSolaris) для управления демонами используется специальный механизм — [Service Management Facility](https://ru.wikipedia.org/wiki/Service_Management_Facility" \o "Service Management Facility).

В системах [Windows](https://ru.wikipedia.org/wiki/Windows" \o "Windows) аналогичный класс программ называется [службой](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B1%D1%8B_Windows) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Services*).

Термин был придуман программистами [проекта MAC](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82_MAC&action=edit&redlink=1) [Массачусетского технологического института](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B0%D1%87%D1%83%D1%81%D0%B5%D1%82%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82), он отсылает к персонажу мысленного эксперимента, [демону Максвелла](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BC%D0%BE%D0%BD_%D0%9C%D0%B0%D0%BA%D1%81%D0%B2%D0%B5%D0%BB%D0%BB%D0%B0), занимающегося сортировкой молекул в фоновом режиме.[[1]](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BC%D0%BE%D0%BD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)#cite_note-1) Системы [UNIX](https://ru.wikipedia.org/wiki/UNIX) унаследовали данную терминологию.

[Демон](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BC%D0%BE%D0%BD) также является персонажем [греческой мифологии](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B5%D0%B2%D0%BD%D0%B5%D0%B3%D1%80%D0%B5%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%B8%D1%84%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F), выполняющим задачи, за которые не хотят браться боги. Как утверждается в «Справочнике системного администратора UNIX», в Древней Греции понятие «персональный даймон» было, отчасти, сопоставимо с современным понятием «[ангел-хранитель](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%B5%D0%BB-%D1%85%D1%80%D0%B0%D0%BD%D0%B8%D1%82%D0%B5%D0%BB%D1%8C)».[[2]](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BC%D0%BE%D0%BD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)#cite_note-2)

Иногда слово **daemon** интерпретируют как [акроним](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D1%80%D0%BE%D0%BD%D0%B8%D0%BC) [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) ***D****isk****a****nd****e****xecution****mon****itor*. Операционные системы семейства [BSD](https://ru.wikipedia.org/wiki/BSD) используют [изображение демона в качестве логотипа](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BC%D0%BE%D0%BD_BSD), иллюстрируя дальнейшее развитие понятия демона уже в христианской традиции.

Docker Engine Components Flow



The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.

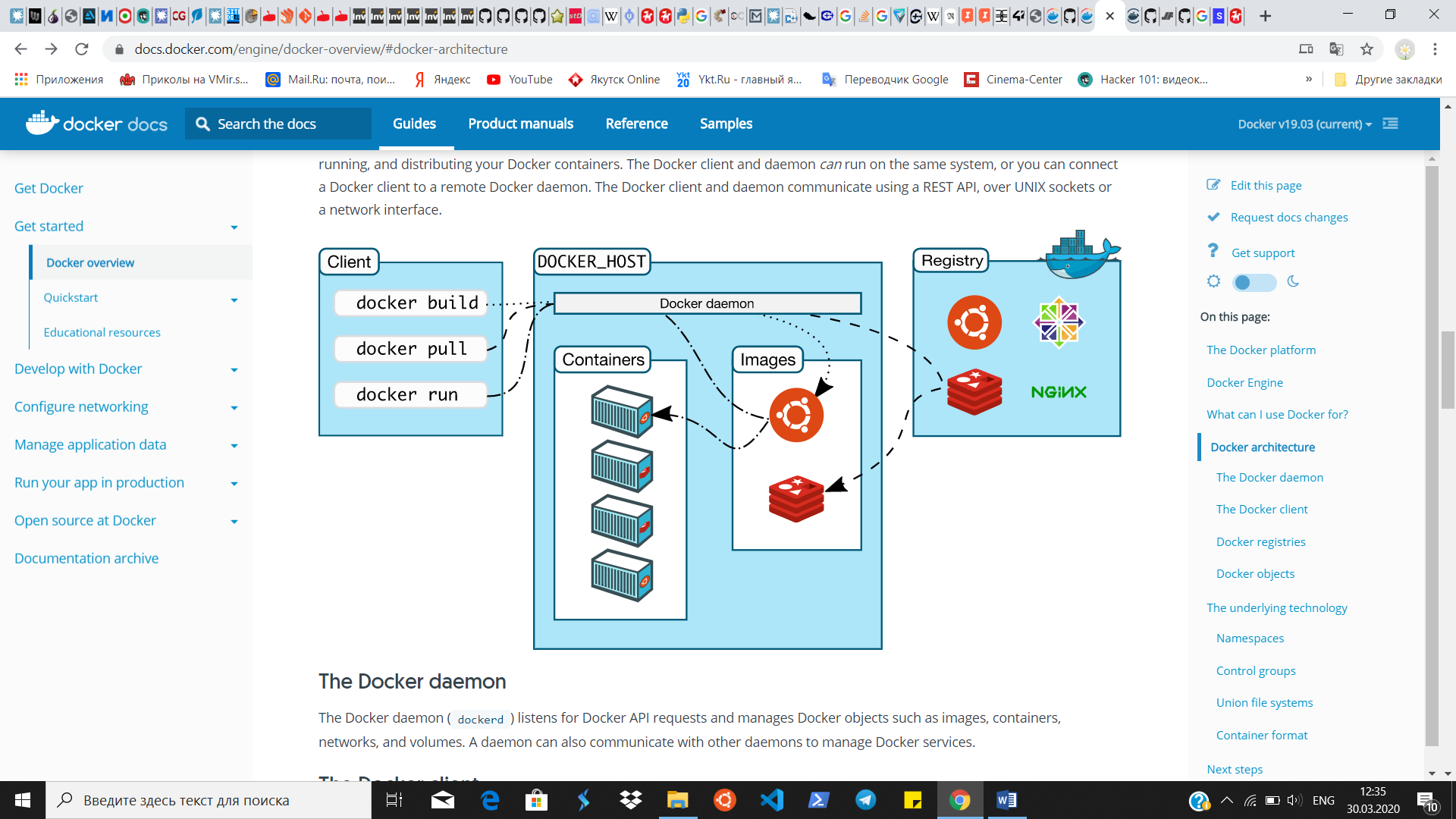
The daemon creates and manages Docker objects, such as images, containers, networks, and volumes.

Note: Docker is licensed under the open source Apache 2.0 license.

<https://docs.docker.com/engine/docker-overview/#docker-architecture> – архитектура докера

## **Docker architecture**[**🔗**](https://docs.docker.com/engine/docker-overview/#docker-architecture)

Docker uses a client-server architecture. The Docker client talks to the Docker daemon which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using s REST API, over UNIX sockets or a network interface.



Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks and volumes. A daemon can also communicate with other daemons to manage Docker services.

Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries – stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

### **Docker objects**

When you use Docker, you are creating and using images, containers, networks, volumes, plugins and other objects.

Images – is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the Ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers – is a runnable instance of an image. You can create, start, stop, move or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container’s network, storage or other underlying subsystems are from other containers of from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

## **Docker concepts**

Docker is a platform for developers and sysadmins to build, run and share applications with containers. The use of containers to deploy applications is called containerization. Containers ae not new, but their use for easily deploying applications is.

Containerization is increasingly popular because containers are:

* **Flexible**: Even the most complex applications can be containerized.
* **Lightweight**: Containers leverage and share the host kernel, making them much more efficient in terms of system resources then virtual machines.
* **Portable**: You can build locally, deploy to the cloud, and run anywhere.
* **Loosely coupled**: Containers are highly self sufficient and encapsulated, allowing you to replace or upgrade one without disrupting other.
* **Scalable**: You can increase and automatically distribute container replicas across a datacenter.
* **Secure**: Containers apply aggressive constraints and isolations to processes without any configuration required on the part of the user.

### **Images and containers**

Fundamentally, a container is nothing but a running process, with some added encapsulation features applied to it in order to keep it isolated from the host and from other containers. One of the most important aspects of container isolation is that each container interacts with its own private filesystem; this filesystem is provided by a Docker image. An image includes everything needed to run an application – the code or binary, runtimes, dependencies and any other filesystem objects required.

### **Containers and virtual machines**

A container runs natively on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.

By contract, a **virtual machine** (VM) runs a full-blown *“guest”* operating system with **virtual** access to host resources through a hypervisor. In general, VMs incur a lot of overhead beyond what is being consumed by your application logic.

***Practice***

**V.2 Exercises**

<https://docs.docker.com/engine/reference/commandline/run/> - команды Docker

For each exercise, we will ask you to give the shell command(s) to:

1. Create a virtual machine with docker-machine using the virtualbox driver, and named Char.

* docker-machine create --driver virtualbox Char

1. Get the IP address of the Char virtual machine.

* docker-machine ip Char

***30.03.20***

1. Define the variables needed by your virtual machine Char in the general env of your terminal, so that you can run the docker ps command without errors. You have to fix all four environment variables with one command, and you are not allowed to use your shell’s builtin to set these variables by hand.

Shell builtin

In computing, a shell builtin is a [command](https://en.academic.ru/dic.nsf/enwiki/1443187/138871) or a [function](https://en.academic.ru/dic.nsf/enwiki/1443187/25900), called from a [shell](https://en.academic.ru/dic.nsf/enwiki/1443187/1086317), that is executed directly in the shell itself, instead of an external executable [program](https://en.academic.ru/dic.nsf/enwiki/1443187/3158) which the shell would load and execute.

With Docker Machine, your Docker engine is running in a VM, which is effectively a remote machine, so your local CLI needs to be configured to connect to it. Docker Machine knows the connection details for the engines it manages, so running docker-machine env default prints out the details for the default machine. The output is something like this:

$ docker-machine env default

export DOCKER\_TLS\_VERIFY="1"

export DOCKER\_HOST="tcp://172.16.62.130:2376"

export DOCKER\_CERT\_PATH="/Users/elton/.docker/machine/machines/default"

export DOCKER\_MACHINE\_NAME="default"

Using eval executes each of those export commands, instead of just writing them to the console, so it's a quick way of setting up your environment variables.

You can undo it and reset the local environment with docker-machine env --unset, which gives you the output for unsetting the environment (so the CLI will try to connect to the local Docker Engine).

This is indeed the expected way to use Docker on a machine that does not natively support Docker, e.g. on Windows or Mac OS X. The Docker documentation includes this step in its description for using Docker Machine here: https://docs.docker.com/machine/get-started/

What this step does (I suggest you also try this yourself):

Run docker-machine env default.

Take the output of that command and execute it in the current shell session.

If you run docker-machine env default yourself, you will see that it simply suggests to set some environment variables, which allow the Docker commands to find the VM running the Docker daemon. Without these variables set, Docker simply does not know how to communicate with the Docker daemon.

In a server environment (Linux), you will not need Docker Machine, since the Linux kernel natively supports running containers. You only need Docker Machine (a small VM running a Linux kernel) on operating systems that don't natively support running containers.

* eval $(docker-machine env Char) - Connect your shell to the new machine

Get the environment commands for your new VM. As noted in the output of the docker-machine create command, you need to tell Docker to talk to the new machine. You can do this with the docker-machine env command.

* eval $("D:\Docker Toolbox\docker-machine.exe" env Char) – я на Windows 10
* Функция eval позволяет выполнить код, переданный ей в виде строки.
* Тебе нужно сообщить клиенту, где находится хост докера eval $(docker-machine env Char)
* Если коротко, то docker-machine env как раз выгружает переменные окружения, в которых в том числе содержится информация для подключения, а ты просто делаешь eval этого (выполняет инструкции export, которые получаешь от этой команды)

1. Get the hello-world container from the Docker Hub, where it’s available.

* docker pull hello-world

1. Launch the hello-world container, and make sure that it prints its welcome message, then leaves it.

* Docker run hello-world

1. Launch an **nginx** container, available on Docker Hub, as a background task. It should be named **overlord**, be able to restart on its own, and have its 80 port attached to the 5000 port of **Char**. You can check that your container functions properly by visiting http://<ip-de-char>:5000 on your web browser.

* docker run --name overlord --restart always -p 5000:80 -d nginx
* -d – в фоновом режиме; $ docker ps –a – запущенные процессы - if container already exists in the system;
* If the container exists, remove it using: $ docker rm <name> Or forcefully using: $ docker rm -f <name>

1. Get the internal IP address of the **overlord** container without starting its shell and in one command.

* $ docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' overlord
* $ docker inspect --format '{{ .NetworkSettings.IPAddress}}' overlord
* inspect - provides detailed information on constructs controlled by Docker.
* $ docker inspect overlord | grep -w "IPAddress" | sed -n 2p | awk -F '"' '{print $4}' – костыльный – grep -w - совпадение только с полным словом; sed –n 2p – убрать вторую строку

<https://sonikelf.ru/find-i-grep-v-linux-i-kak-instrument-dlya-administrirovaniya/> - записи сисадмина много команд по grep и find, а также regex

1. Launch a shell from an **alpine** container, and make sure that you can interact directly with the container via your terminal, and that the container deletes itself once the shell’s execution is done.

* **Alpine Linux** — дистрибутив Linux, ориентированный на безопасность, легковесность и нетребовательность к ресурсам. В основном используется во встраиваемых системах, также хорошо подходит для создания Docker-контейнеров.
* The **-t** option goes to how Unix/Linux handles terminal access. In the past, a terminal was a hardline connection, later a modem based connection. These had physical device drivers (they were real pieces of equipment). Once generalized networks came into use, a pseudo-terminal driver was developed. This is because it creates a separation between understanding what terminal capabilities can be used without the need to write it into your program directly (read man pages on **stty**, **curses**).

So, with that as background, run a container with no options and by default you have a stdout stream (so docker run | <cmd> works); run with **-i**, and you get stdin stream added (so <cmd> | docker run -i works); use **-t**, usually in the combination **-it** and you have a terminal driver added, which if you are interacting with the process is likely what you want. It basically makes the container start look like a terminal connection session.

* docker run/exec -i will connect the STDIN of the command inside the container to the STDIN of the **docker run/exec** itself.

So docker run **-i** alpine cat gives you an empty line waiting for input. Type "hello" you get an echo "hello". The container will not exit until you send CTRL+D because the main process cat is waiting for input from the infinite stream that is the terminal input of the docker run.

On the other hand echo "hello" | docker run -i alpine cat (поменять местами при выполнении) will print "hello" and exit immediately because cat notices that the input stream has ended and terminates itself.

If you try docker ps after you exit either of the above, you will not find any running containers. In both cases, cat itself has terminated, thus docker has terminated the container.

Now for **"-t"**, this tells the main process inside docker that its input is a terminal device. So

docker run -t alpine cat will give you an empty line, but if you try to type "hello", you will not get any echo. This is because while cat is connected to a terminal input, this input is not connected to your input. The "hello" that you typed did not reach the input of cat. cat is waiting for input that never arrives.

echo "hello" | docker run -t alpine cat will also give you an empty line and will not exit the container on CTRL-D but you will not get an echo "hello" because you didn't pass **-i**

If you send CTRL+C, you get your shell back, but if you try docker ps now, you see the cat container still running. This is because cat is still waiting on an input stream that was never closed. **I have not found any useful use for the -t alone without being combined with -i**.

Now, for **-it** together. This tells cat that its input is a terminal and in the same time connect this terminal to the input of docker run which is a terminal. docker run/exec will make sure that its own input is in fact a **tty** before passing it to cat. This is why you will get a input device is not a **TTY** if you try echo "hello" | docker run -it alpine cat because in this case, the input of docker run itself is the pipe from the previous echo and not the terminal where docker run is executed

Finally, why would you need to pass **-t** if **-i** will do the trick of connecting your input to cat's input? This is because commands treat the input differently if it's a terminal. This is also best illustrated by example

* docker run -e MYSQL\_ROOT\_PASSWORD=123 -i mariadb mysql -u root -p will give you a password prompt. If you type the password, the characters are printed visibly.
* docker run -i alpine sh will give you an empty line. If you type a command like ls you get an output, but you will not get a prompt or colored output.
* In the last two cases, you get this behavior because **mysql** as well as **shell** were not treating the input as a **tty** and thus did not use **tty** specific behavior like masking the input or coloring the output.

<https://stackoverflow.com/questions/30137135/confused-about-docker-t-option-to-allocate-a-pseudo-tty>

* **tty** — Unix-утилита, выводящая имя терминала, соединённого со стандартным вводом. В случае, если стандартный ввод не является терминалом, выводит сообщение «not a tty».
* docker run -it --rm alpine /bin/sh
* docker pull alpine && docker run --rm -it alpine (-it флаг для интерактивных процессов, таких как оболочка; --rm удалит контейнер после выхода; docker pull - pull an image or a repository from a registry)

1. From the shell of a debian container, install via the container’s package manager everything you need to compile C source code and push it onto a git repo (of course, make sure before that the package manager and the packages already in the container are updated). For this exercise, you should only specify the commands to be run directly in the container.

* docker run -dit --name test debian
* docker exec -it test /bin/sh
* apt-get update && apt-get upgrade –y

apt-get install -y build-essential git

* ПРОВЕРКА ВНУТРИ КОНТЕЙНЕРА: dpkg -l | grep gcc и dpkg -l | grep git
* docker exec -it test dpkg -l git gcc
* apt-get update && apt-get upgrade –y

apt-get install –y gcc git

* пакет build-essentials является справочным для всех пакетов, необходимых для компиляции пакета Debian. Обычно она включает компиляторы и библиотеки GCC/g++ и некоторые другие утилиты.
* build-essential - это пакет, который содержит ссылки на множество пакетов, необходимых для создания программного обеспечения в целом.
* "build-essential" содержит инструменты (такие как компилятор gcc, make tool и т. д.) для компиляции/сборки программного обеспечения из исходного кода. Таким образом, вы начинаете с (обычно C) исходных файлов и создаете из них исполняемые файлы.
* команда sudo apt-get build-dep packagename означает "как root, установить все зависимости для 'packagename' , так что я могу построить его". Таким образом, build-dep-это команда apt-get, такая же, как install, remove, update и т. д.

1. Create a volume named hatchery.

* Существуют два способа, позволяющих сделать срок жизни данных большим срока жизни контейнера. Один из способов заключается в использовании технологии bind mount. При таком подходе к контейнеру можно примонтировать, например, реально существующую папку. Работать с данными, хранящимися в такой папке, смогут и процессы, находящиеся за пределами Docker. Минусы использования технологии bind mount заключаются в том, что её использование усложняет резервное копирование данных, миграцию данных, совместное использование данных несколькими контейнерами. Гораздо лучше для постоянного хранения данных использовать тома Docker.
* <https://habr.com/ru/company/ruvds/blog/441574/> - хранение данных в Докере; обучение работы с Докером – часть 6;
* Volume - Том — это файловая система, которая расположена на хост-машине за пределами контейнеров. Созданием и управлением томами занимается Docker. Вот основные свойства томов Docker:

- Они представляют собой средства для постоянного хранения информации.

- Они самостоятельны и отделены от контейнеров.

- Ими могут совместно пользоваться разные контейнеры.

- Они позволяют организовать эффективное чтение и запись данных.

- Тома можно размещать на ресурсах удалённого облачного провайдера.

- Их можно шифровать.

- Им можно давать имена.

- Контейнер может организовать заблаговременное наполнение тома данными.

- Они удобны для тестирования.

* docker volume create --name hatchery

1. List all the Docker volumes created on the machine. Remember. VOLUMES.

* docker volume ls

1. Launch a **mysql** container as a background task. It should be able to restart on its own in case of error, and the root password of the database should be **Kerrigan**. You will also make sure that the database is stored in the **hatchery volume**, that the container directly creates a database named **zerglings**, and that the container itself is named **spawning-pool**.

* docker run –d --name spawning-pool --restart=on-failure:7 -e MYSQL\_ROOT\_PASSWORD=Kerrigan -e MYSQL\_DATABASE=zerglings -v hatchery:/var/lib/mysql mysql ~~--default-authentication-plugin=mysql\_native\_password~~(?????)
* <https://docs.docker.com/config/containers/start-containers-automatically/> - restart=on-failure
* <https://docs.docker.com/engine/reference/commandline/run/> - docker run
* -e - set environment variables
* -v - Bind mount a volume
* To fix error with database connection with mysql 8 and newer version of Wordpress. You need to modify the authentification plugin in mysql. If you are upgrading your server to MySQL 8.0 and observe that your application is experiencing error related caching\_sha2\_password plugin, it is likely because your clients/connectors does not (yet) support caching\_sha2\_password plugin. To resolve this issue, you may consider using mysql\_native\_password as default authentication for MySQL 8.0 server. Add following entry in MySQL configuration file.

[mysqld]

default-authentication-plugin=mysql\_native\_password

This way, once server is restarted, your existing applications should be able to connect to the server without any issues.

* # To fix error with database connection with mysql 8 and newer version of Wordpress

# You need to modify the authentification plugin in mysql

# Run the followings commands

# $ docker exec -it spawning-pool bash

# $ bash: mysql --user=root --password=Kerrigan

# $ bash: mysql> ALTER USER 'root' IDENTIFIED WITH mysql\_native\_password BY 'Kerrigan';

docker run --name spawning-pool --restart on-failure -e MYSQL\_ROOT\_PASSWORD=Kerrigan -e MYSQL\_DATABASE=zerglings -v hatchery:/var/lib/mysql -d mysql

* **ПРОВЕРКА:** docker exec -it spawning-pool mysql -u root -p
* mysql -u root -p - для работы с mysql через командную строку <https://jeka.by/post/1003/rabotaem-s-mysql-cherez-komandnuyu-stroku/>

1. Print the environment variables of the **spawning-pool** container in one command, to be sure that you have configured your container properly.

* docker inspect --format ‘{{ .Config.Env }}’ spawning-pool
* docker exec spawning-pool env
* docker exec - применяется к запущенному контейнеру, запускает новый процесс внутри пространства процессов контейнера. <https://habr.com/ru/post/272145/>
* docker run - находит образ, создает контейнер поверх него и запускает контейнер. Это сделано для удобства и скрывает детали двух команд. Run a command in a new container.

***31.03.20***

Проблемы с запуском контейнеров и продолжением работы с заданиями.

* Запуск уже существующей вирт. Машины Char - **docker-machine start Char / docker-machine stop Char**
* <https://habr.com/ru/post/486200/> - Docker Tips: Очистите свою машину от хлама
* **docker system df** - сколько же места реально занято на вашей машине Docker’ом
* для того, чтобы сделать вирт.машину активной нужно снова установить переменные среды - eval $(docker-machine env Char) - все контейнеры и тома в ней сохранились.
* <https://linux-notes.org/ustanovka-docker-machine-v-unix-linux/> - через ssh можно подключиться к внутренностям машины
* docker start <name\_of\_container> - запускает контейнер
* docker exec -it <name\_of\_container> env - выполняет в запущенном контейнере вывод переменных его среды (<https://docs.docker.com/network/links/>)

Возвращаюсь к работе с заданиями:

1. Launch a **wordpress** container as a background task, just for fun. The container should be named **lair**, its 80 port should be bound to the 8080 port of the virtual machine, and it should be able to use the **spawning-pool container** as a database service. You can try to access **lair** on your machine via a web browser, with the IP address of the virtual machine as a URL. Congratulations, you just deployed a functional Wordpress website in two commands!

* **Wordpress** - система управления содержимым сайта с открытым исходным кодом; написана на PHP; сервер базы данных — MySQL; выпущена под лицензией GNU GPL версии 2. Сфера применения — от блогов до достаточно сложных новостных ресурсов.
* docker run -d --name lair -p 8080:80 --link spawning-pool:mysql wordpress
* --link name:alias - name — имя контейнера, alias — имя, под которым этот контейнер будет известен запускаемому.
* docker run -d --rm --name lair -p 8080:80 --link spawning-pool:mysql wordpress

1. Launch a **phpmyadmin** container as a background task. It should be named **roach-warden**, its 80 port should be bound to the 8081 port of the virtual machine and it should be able to explore the database stored in the **spawning-pool** container.

* docker run --name roach-warden -d --link spawning-pool:db -p 8081:80 phpmyadmin/phpmyadmin
* **ЗАМЕТКА**: **--link** - устарела, теперь нужно использовать **--network**, для этого сначала нужно создать мост. --link spawning-pool:database - не работает из-за него myphpadmin
* Сервер: db via TCP/IP - по дефолту сервер mysql называется db, поэтому другие названия приводят к ошибке «Невозможно подключиться к серверу mysql»(?)
* <https://hub.docker.com/r/phpmyadmin/phpmyadmin/> - myphpadmin образ на хабе
* <https://docs.docker.com/network/links/> - офиц.документация с предупреждением, что --link устарел про порты и другие связи - в основном link
* docker run --detach --name roach-warden --publish 8081:80 --network my-docker-bridge -e PMA\_HOST=spawning-pool phpmyadmin/phpmyadmin
* [https://docs.docker.com/network/bridge/##differences-between-user-defined-bridges-and-the-default-bridge](https://docs.docker.com/network/bridge/) - Use bridge networks
* <https://docs.docker.com/engine/reference/commandline/network_create/#specify-advanced-options> - docker network create
* Чтобы связать между собой контейнеры
* По умолчанию для контейнеров используется **bridge**. При первом запуске контейнера Docker создает дефолтную bridge-сеть с одноименным названием - используется всеми контейнерами, для лучшей изоляции надо создать пользовательский мост. Эту сеть можно увидеть в общем списке по команде docker network ls. Чтобы проинспектировать ее свойства, запустим команду docker network inspect bridge. <https://habr.com/ru/post/333874/>
* Link - делится переменными окружения между контейнерами в отличие от network. Однако в link отсутствует двухсторонней коммуникации между связанными контейнерами - сложности при более 2х связанных контейнеров. To share environment variables: 1) Multiple containers can mount a file or directory containing the shared information, using a Docker volume. 2) Multiple containers can be started together using docker-compose and the compose file can define the shared variables. 3) You can use swarm services instead of standalone containers, and take advantage of shared secrets and configs.
* Containers connected to the same user-defined bridge network effectively expose all ports to each other. For a port to be accessible to containers or non-Docker hosts on different networks, that port must be published using the -p or --publish flag.
* docker network create my-bridge-network
* docker network connect my-bridge-network spawning-pool
* docker run --detach --name roach-warden --publish 8081:80 --network my-docker-bridge -e PMA\_HOST=spawning-pool phpmyadmin/phpmyadmin
* docker network disconnect my-bridge-network roach-warden
* docker network rm my-bridge-network

***01.04.20***

Еще раз углубилась и сделала 14 и 15 задания.

* docker images - показывает размер образа
* docker ps -as - также добавляется размер
* docker attach <id\_or\_name\_of\_container> - attach local standard input, output, and error streams to a running container - входит в работающий и при выходе закрывает контейнер

1. Look up the **spawning-pool** container’s logs in real time without running its shell.

* docker logs --folow spawning-pool

1. Display all the currently active containers on the **Char** virtual machine.

* docker ps - только запущенные
* docker ps -a - все контейнеры

1. Relaunch the **overlord** container.

* docker restart overlord

1. Launch a container name **Abathur**. It will be a **Python** container, 2-slim version, its /root folder will be bound to a HOME folder on your host, and its 3000 port will be bound to the 3000 port of your virtual machine. You will personalize this container so that you can use the **Flask** micro-framework in its latest version. You will make sure that an html page displaying "Hello World" with tags can be served by **Flask**. You will test that your container is properly set up by accessing, via curl or a web browser, the IP address of your virtual machine on the 3000 port. You will also list all the necessary commands in your repository.

* docker run -it ~~--rm~~ --detach --name Abathur -v $HOME:/root --publish 3000:3000 python:2-slim
* **Flask** — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.
* <https://flask.palletsprojects.com/en/1.1.x/quickstart/#a-minimal-application> - Flask Quickstart
* docker exec Abathur pip install Flask
* echo 'from flask import Flask\napp = Flask(\_\_name\_\_)\n@app.route("/")\ndef hello\_world():\n\treturn "<h1>Hello, World!</h1>"' > ~/app.py
* docker exec -e FLASK\_APP=/root/app.py Abathur flask run --host=0.0.0.0 --port 3000
* docker run -dit --name Abathur -v mnt/d/root -p 3000:3000 python:2-slim
* docker exec Abathur pip install --upgrade pip
* <https://www.datacamp.com/community/tutorials/pip-python-package-manager> - Python и package-manager pip
* docker exec Abathur pip install flask
* docker exec -it Abathur /bin/sh
* echo “from flask import Flask\napp = Flask(\_\_name\_\_)\n\n@app.route(‘/’)\ndef hello\_world():\n\treturn "Hello, World!"\n\nif \_\_name\_\_ == ‘\_\_main\_\_’:\n\tapp.run(debug=True, host=’0.0.0.0’, port=3000)” > /mnt/d/root/hello.py
* <https://flask-russian-docs.readthedocs.io/ru/latest/quickstart.html> - русская офиц докум-я flask
* <https://habr.com/ru/post/346306/> - начало работы с flask
* <https://pythonru.com/uroki/3-osnovy-flask> - начало работы с flask - В первой строке класс **Flask** импортируется из пакета **flask**. Во второй строке создается объект **Flask**. Для этого конструктору **Flask** назначается аргумент **\_\_name\_\_.** Конструктор **Flask** должен иметь один обязательный аргумент. Им служит название пакета. В большинстве случаев значение **\_\_name\_\_** подходит. Название пакета приложения используется **фреймворком Flask**, чтобы находить статические файлы, шаблоны и т. д.
* **Маршрут (или путь)** используется во фреймворке **Flask** для привязки URL к функции представления. Эта функция отвечает на запрос. Во **Flask** декоратор **route** используется, чтобы связать URL адрес с функций.
* Этот код назначает функцию **hello\_world()** обработчиком корневого URL в приложении. Другими словами, каждый раз, когда приложение будет получать запрос, где путь — /, вызывается функция **hello\_world(),** и на этом запрос завершается.
* Для запуска сервера разработки нужно использовать метод **run()** объекта Flask. Условие **\_\_name\_\_ == "\_\_main\_\_"** гарантирует, что метод **run()** будет вызван только в том случае, если **main.py** будет запущен, как основная программа. Если попытаться использовать метод **run()** при импорте **main.py** в другой модуль Python, он не вызовется.
* Режим отладки будет перезапускать сервер после любых изменений в коде. Чтобы включить режим, нужно передать аргумент **debug=True** методу **run(). host='0.0.0.0' -** делаем сервер публично доступным.
* docker exec --env FLASK\_APP=/mnt/d/root/hello.py Abathur flask run --host=0.0.0.0 --port=3000
* **Фре́ймворк** (остов, каркас, структура) — программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта. <https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA>
* **Фреймворк** — набор инструментов для ускоренной разработки сайта, нацелен на решение определенных задач.
* Говоря простыми словами, framework – это каркас, который состоит из множества различных библиотек, которые облегчают разработку программного продукта или сайта. То есть фреймворк это набор из нескольких библиотек. Фреймворки различны по своим возможностям и функциям. Их наполнение зависит от поставленных задач. Они нужны для того, чтобы программист или дизайнер могли сосредоточиться на уникальных задачах, а не заново изобретали колесо. <https://web-creator.ru/articles/about_frameworks>
* <https://tproger.ru/translations/web-frameworks-how-to-get-started/>

***04.04.20***

1. Create a local swarm, the **Char** virtual machine should be its manager.

* <https://habr.com/ru/company/southbridge/blog/310606/>
* <https://habr.com/ru/company/redmadrobot/blog/318866/>
* <https://kamaok.org.ua/?p=2980>
* <https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/>
* <https://docs.docker.com/engine/swarm/swarm-tutorial/>
* <https://docs.docker.com/engine/swarm/>
* <https://blog.gelin.ru/2017/04/docker-swarm-mode.html>
* <https://linux-notes.org/nastrojka-docker-swarm-klastera-v-unix-linux/>
* <https://docs.docker.com/engine/swarm/key-concepts/>
* docker swarm init --advertise-addr $(docker-machine ip Char)
* docker node ls

1. Create another virtual machine with **docker-machine** using the virtualbox driver, and name it **Aiur**.

* docker-machine create --driver virtualbox Aiur
* docker-machine ls

1. Turn **Aiur** into a slave of the local swarm in which **Char** is leader (the command to take control of **Aiur** is not requested).

* <https://docs.docker.com/engine/swarm/swarm-tutorial/add-nodes/>
* <https://docs.docker.com/engine/swarm/swarm-mode/>
* docker-machine ssh Aiur “docker swarm join --token $(docker swarm join-token --quiet worker) $(docker-machine ip Char):2377”
* docker-machine ssh Aiur "docker swarm join --token $(docker swarm join-token worker -q) $(docker-machine ip Char):2377"

1. Create an overlay-type internal network that you will name **overmind**.

* <https://docs.docker.com/network/overlay/>
* <https://docs.docker.com/network/network-tutorial-overlay/>
* https://docs.docker.com/engine/reference/commandline/network\_create/#specify-advanced-options
* You can connect multiple containers to the same network. Once connected, the containers can communicate using only another container’s IP address or name. For overlay networks or custom plugins that support multi-host connectivity, containers connected to the same multi-host network but launched from different Engines can also communicate in this way.
* docker network create -d overlay overmind
* docker network create --driver=overlay overmind
* docker network ls

1. Launch a **rabbitmq** SERVICE that will be named **orbital-command**. You should define a specific user and password for the **RabbitMQ** service, they can be whatever you want. This service will be on the **overmind** network.

* **RabbitMQ** — программный брокер сообщений на основе стандарта AMQP — тиражируемое связующее программное обеспечение, ориентированное на обработку сообщений. Создан на основе системы Open Telecom Platform, написан на языке Erlang, в качестве движка базы данных для хранения сообщений использует Mnesia. Его основная цель ‒ принимать и отдавать сообщения
* <https://docs.docker.com/engine/swarm/swarm-tutorial/deploy-service/>
* <https://hub.docker.com/_/rabbitmq> - docker hub RabbitMQ
* docker service create -d --name orbital-command --network overmind -e RABBITMQ\_DEFAULT\_USER=user -e RABBITMQ\_DEFAULT\_PASS=password rabbitmq
* docker inspect -f "{{.Spec.TaskTemplate.ContainerSpec.Env}}" orbital-command

1. List all the services of the local swarm.

* docker service ls

1. Launch a **42school/engineering-bay** service in two replicas and make sure that the service works properly (see the documentation provided at hub.docker.com). This service will be named **engineering-bay** and will be on the **overmind** network.

* <https://hub.docker.com/r/42school/engineering-bay> - 42school/engineering-bay
* docker service create --name engineering-bay --replicas 2 --network overmind -e OC\_USERNAME=user -e OC\_PASSWD=password 42school/engineering-bay

1. Get the real-time logs of one the tasks of the **engineering-bay** service.

* <https://docs.docker.com/engine/reference/commandline/service_logs/>
* <https://docs.docker.com/engine/reference/commandline/service_ps/>
* docker service ps engineering-bay
* docker service logs --follow $(docker service ps engineering-bay -f “name=engineering-bay.1” -q)

1. ... Damn it, a group of zergs is attacking **orbital-command**, and shutting down the **engineering-bay** service won’t help at all... You must send a troup of Marines to eliminate the intruders. Launch a **42school/marine-squad** in two replicas, and make sure that the service works properly (see the documentation provided at hub.docker.com). This service will be named... **marines** and will be on the **overmind** network.

* <https://hub.docker.com/r/42school/marine-squad>
* docker service create --name marines --network overmind --replicas 2 -e OC\_USERNAME=user -e OC\_PASSWD=password 42school/marine-squad

1. Display all the tasks of the **marines** service.

* docker service ps marines

1. Increase the number of copies of the **marines** service up to twenty, because there’s never enough Marines to eliminate Zergs. (Remember to take a look at the tasks and logs of the service, you’ll see, it’s fun.)

* <https://docs.docker.com/engine/swarm/swarm-tutorial/scale-service/>
* docker service scale marines=20
* docker service ps marines
* docker service logs $(docker service ps marines --filter “name=marines.20” -q)

1. Force quit and delete all the services on the local swarm, in one command.

* docker service rm $(docker service ls -q)
* --quiet, -q - only display ID’s
* <https://docs.docker.com/engine/swarm/swarm-tutorial/delete-service/>
* docker service ls

1. Force quit and delete all the containers (whatever their status), in one command.

* <https://docs.docker.com/engine/reference/commandline/rm/>
* docker rm -f $(docker ps -a -q)
* docker container rm -f $(docker container ls -q)
* docker ps -a

1. Delete all the container images stored on the **Char** virtual machine, in one command as well.

* <https://docs.docker.com/engine/reference/commandline/rmi/>
* <https://docs.docker.com/engine/reference/commandline/image/>
* <https://docs.docker.com/engine/reference/commandline/images/>
* docker images -a
* docker rmi $(docker images -a -q)

1. Delete the **Aiur** virtual machine without using rm -rf.

* <https://docs.docker.com/machine/get-started/>
* <https://docs.docker.com/machine/reference/>
* <https://docs.docker.com/machine/reference/rm/>
* docker-machine rm -y Aiur
* -y - даем согласие на удаление заранее
* docker-machine ls

***Chapter VI***

***Dockerfiles***

***08.04.20***

\* Метадата и cashe?????

<https://habr.com/ru/post/272145/> - образы и контейнеры Докер в картинках

<https://habr.com/ru/company/infobox/blog/240623/> - Погружаемся в Docker: Dockerfile и коммуникация между контейнерами

<https://habr.com/ru/post/346634/>

***EX00***

#ARG VERSION=latest

#FROM alpine:$VERSION

#defines a variable that users can pass at build-time to the builder with the docker build command

FROM alpine:latest

#определяем базовый/родительский образ для начала процесса построения контейнера

#This is creating the docker image based on the alpine image

#MAINTAINER aagrivan - устарело в пользу LABEL

LABEL maintainer="aagrivan"

#позволяет добавлять в образ метаданные.

#It is good practice to put who is in charge of the file

RUN apk update && \

    apk upgrade && \

    apk add vim

#Run commands are the ones run when the image is being built

ENTRYPOINT ["vim"]

#ENTRYPOINT vim

#has two forms: exec form (preferred) and shell form

#Entrypoint command defines which is the command run in the container when it is being run, but you can still add more to that command

<https://wiki.merionet.ru/servernye-resheniya/9/kak-rabotat-s-dockerfile/>

<https://habr.com/ru/company/ruvds/blog/439980/>

https://docs.docker.com/engine/reference/commandline/build/

# How to build it?

# docker build -f ../School21/Docker/01\_dockerfiles/00/Dockerfile -t ex00 .

--tag, -t - Name and optionally a tag in the ‘name:tag’ format

--file, -f - Name of the Dockerfile (Default is ‘PATH/Dockerfile’)

“.” - построить образ здесь

# How to run it?

# docker run --rm -ti ex00

***EX01***

**TeamSpeak** — компьютерная программа, предназначенная для голосового общения в сети Интернет посредством технологии VoIP. От классического телефона отличается практически неограниченным количеством абонентов, разговаривающих одновременно.

<https://www.hostinger.ru/rukovodstva/kak-sozdat-teamspeak-3-server-rukovodstvo/> - основная статья

<https://riptutorial.com/ru/docker/example/11006/%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F-env-%D0%B8-arg>

<https://hub.docker.com/r/aheil/teamspeak3-server/dockerfile>

<https://hub.docker.com/_/teamspeak>

<https://wiki.merionet.ru/servernye-resheniya/9/kak-rabotat-s-dockerfile/>

<https://teamspeak.com/en/downloads/#server>

ARG DEBIAN\_FRONTEND=noninteractive

FROM debian:latest

LABEL maintainer="aagrivan"

ENV TS3SERVER\_LICENSE=accept

# accept the teamspeak license

WORKDIR /home/teamspeak

# указывает директорию откуда будет выполняться команда

EXPOSE 9987/udp 10011 30033

# 9987 default voice

# 10011 server query

# 30033 file transport

RUN apt-get update -y && \

    apt-get upgrade -y && \

    apt-get install -y wget bzip2 && \

    wget https://files.teamspeak-services.com/releases/server/3.12.1/teamspeak3-server\_linux\_amd64-3.12.1.tar.bz2 && \

    tar -xvf teamspeak3-server\_linux\_amd64-3.12.1.tar.bz2

WORKDIR teamspeak3-server\_linux\_amd64

ENTRYPOINT sh ts3server\_minimal\_runscript.sh

***ПРОВЕРКА:***

docker build -f ../School21/Docker/01\_dockerfiles/01/Dockerfile -t ex01 .

docker run --rm -p 9987:9987/udp -p 10011:10011 -p 30033:30033 ex01

docker ps -a

docker stop <name\_of\_container>

docker rmi -f ex01

Установить Teamspeak 3 client версия:

<https://www.hostinger.ru/rukovodstva/kak-sozdat-teamspeak-3-server-rukovodstvo/>

***EX02***

**Ruby on Rails** — фреймворк, написанный на языке программирования Ruby, реализует архитектурный шаблон Model-View-Controller для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных. Является открытым программным обеспечением и распространяется под лицензией MIT.

***Rails*** - фреймворк для веб-разработки, написанный на языке программирования Ruby. Он разработан, чтобы сделать программирование веб-приложений проще, так как использует ряд допущений о том, что нужно каждому разработчику для создания нового проекта. Он позволяет вам писать меньше кода в процессе программирования, в сравнении с другими языками и фреймворками. Профессиональные разработчики на Rails также отмечают, что с ним разработка веб-приложений более забавна =)

FROM  ft-rails:on-build

EXPOSE 3000

CMD     ["rails", "s", "-b", "0.0.0.0", "-p", "3000"]

# rails server --binding=0.0.0.0 --port 3000

Localhost means quite literally "your local host", usually identified by 127.0.0.1 and all traffic to that address is routed via a loopback interface. If your Web server is listening for connections on 127.0.0.1, this means that it only accepts requests coming from the same host.

0.0.0.0. means that Rails is listening on all interfaces, not just the loopback interface. 0.0.0.0 is not an IP address you can actually access using a client. It's simply a shortcut for the system binding call to use all available IP addresses.

***Платформа Node или NodeJS*** — платформа, основанная на движке V8, превращающая JavaScript в язык общего назначения.

При разработке на NodeJS используются событийно-ориентированное и асинхронное программирование. NodeJS достаточно сильно расширяет возможности использования языка JavaScript, поэтому на его базе можно разрабатывать как веб-серверные приложения, так и десктопные приложения. Также встроенный в Node пакетный менеджер npm(~~с ним много разных ошибок~~) сильно упрощает работу с пакетами и их зависимостями при любой разработке, связанной с использованием JavaScript.

Мы очень редко используем возможности NodeJS для бэкенд-разработки, так как сознательно предпочитаем разрабатывать эту часть веб-приложений на Ruby on Rails. Но активно используем NodeJS в разработке десктопных приложений и для упрощения работы с фронтендом.

cd ../School21/Docker/01\_dockerfiles/02/

docker build -t ft-rails:on-build ft\_rails

docker build -t ex02 .

docker run -it --rm -p 3000:3000 ex02

Проверка: <docker-machine ip default>:3000

Attention Windows Users

> You should access port 3000 with the following address 127.0.0.1:3000 rather than 0.0.0.0:3000.

localhost:3000 or 127.0.0.1:3000 should work.

0.0.0.0:3000 is the computer's way of saying it's listening on every available address (in the case it has multiple IP addresses). 0.0.0.0 will work on some systems for dialing your own computer, but more reliably the 127.0.0.1 will always loopback to your computer, and localhost should also.

***EX03***

<https://www.youtube.com/watch?v=WZMluVKwKzc> - Как установить GitLab локально и совершенно бесплатно на Linux дистрибутиве?

Михаил Толстой - Тайм-менеджмент

20/80 - 20% усилий дает 80% результатов. 70 лет = 250 месяцев

3 кита тайм-менеджмента

1. Целеполагание - стать целеустремленным:
   1. спорт - «ресурс»
   2. учиться - «развитие»
   3. создавай навыки
2. Планирование:
   1. Вспомни цель
   2. Первое действие про цель - правило лягушки - когда видим план - есть цели, которые мы не хотим делать - «склизкая лягушки» - кушай лягушек на завтрак - когда решаем получаем эмоц подъем.
   3. Контроль
3. Мотивация - никакой внешний стимулятор вас не простимулирует - должны быть внутренне промотивированы.

Личная эффективность зависит от того, сколько раз тебя отвлекают.

Ограничьте источники информации.

«Мы материальны»

1. План дня на бумаге
2. Важные письма
3. Читайте «живые книги» - есть цифра - усваиваемость живых книг выше

«Мы не работаем в одиночку»

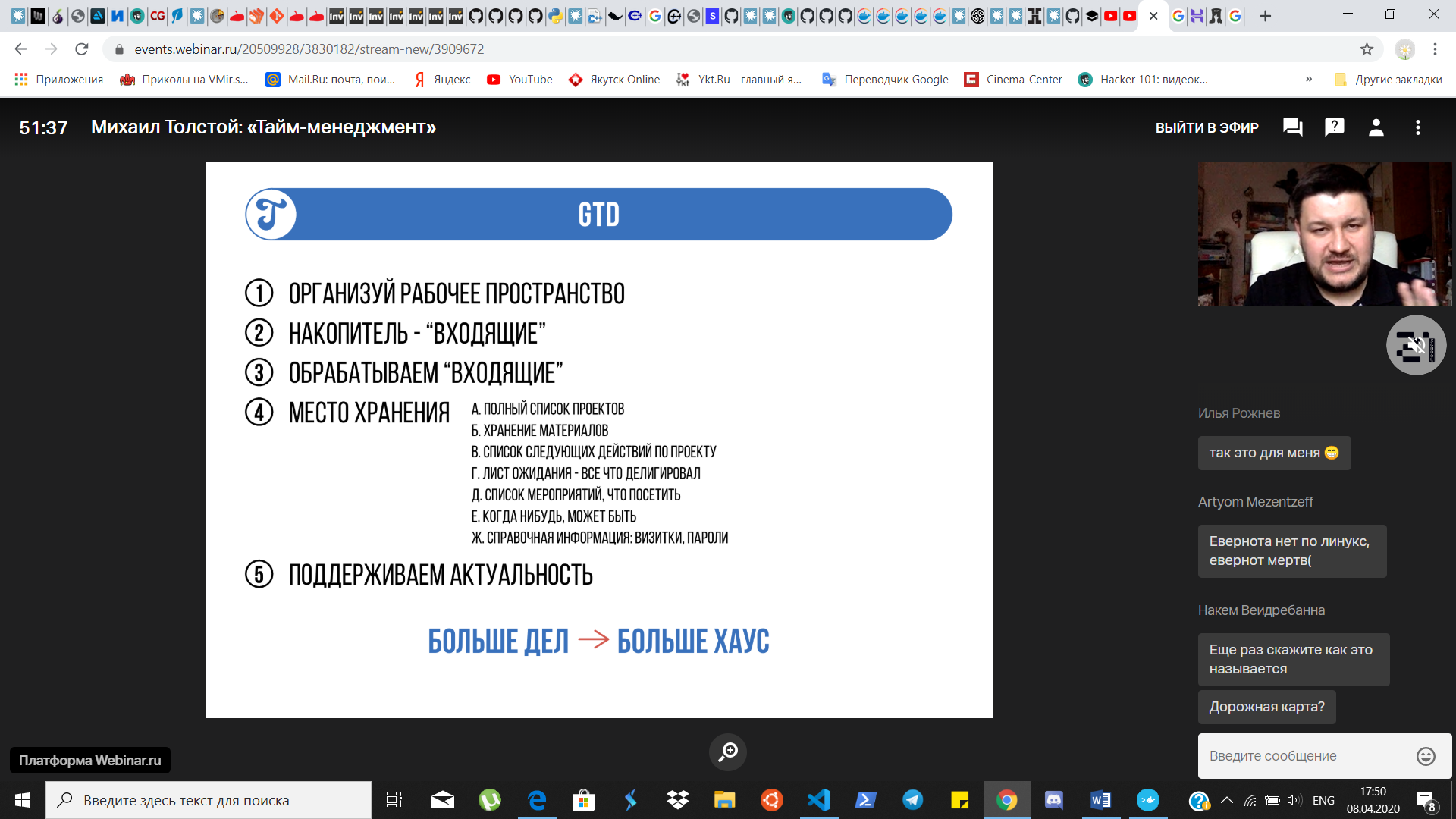
1. Стандарты
2. Конкуренция по времени
3. «Доступ к телу»

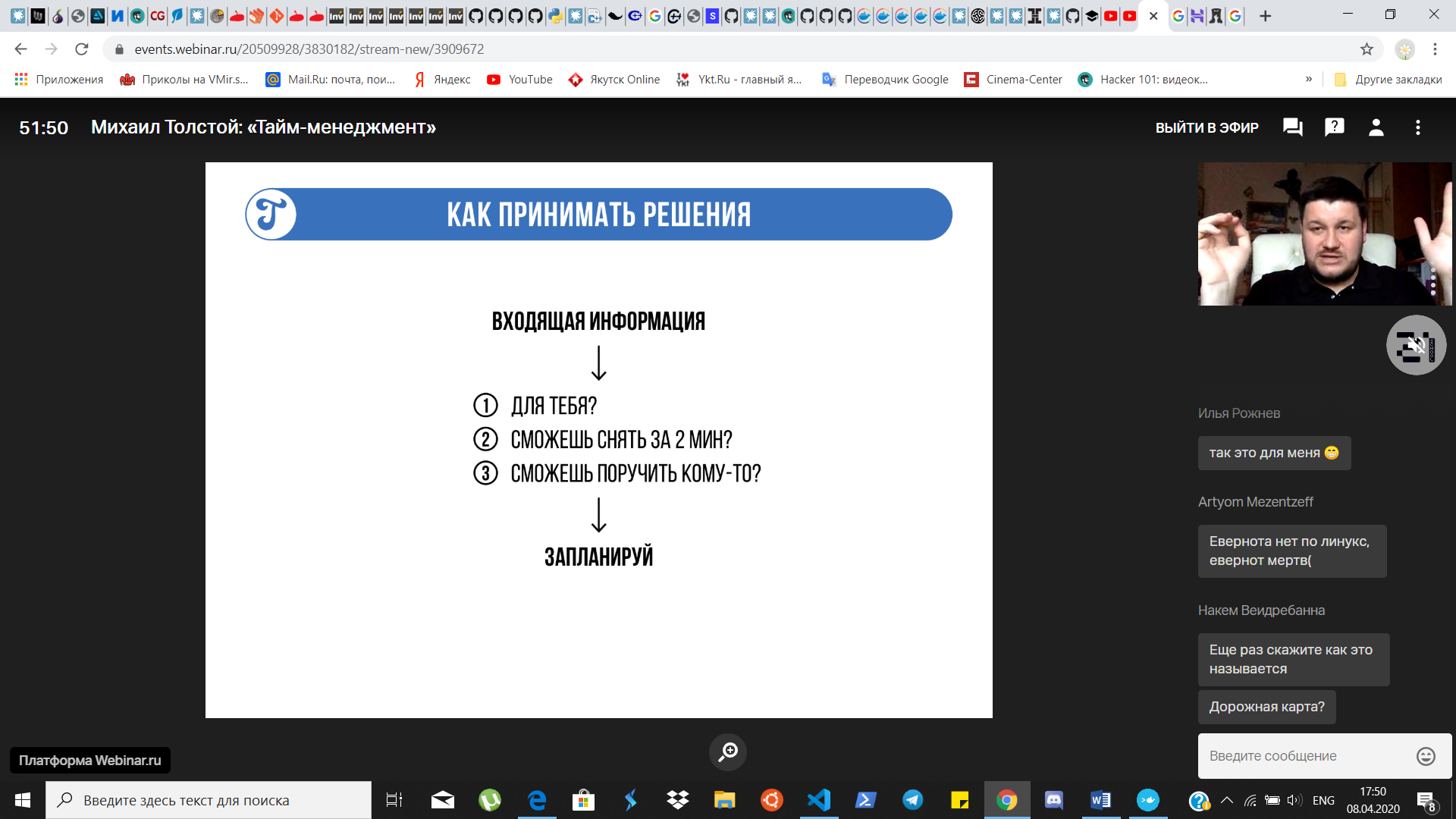
Правила тайм-менеджмента:

1. Я умею управлять своим временем (пунктуальность)
2. Начинай день с планирования его
3. Избавься от прокрастинации (емайл в том числе)
4. Используй помодоро (концентрация)
5. Фильтруй и сортируй входящую информацию
6. Не берись за дела, которые не в твоих целях
7. Знай цену часа своего времени

Дорожная карта - на 1 год в январе - coggle - приложение

Яндекс.диск: правило 30 секунд - должен найти за 30 сек.





1. Get Things Done? - вытащить все мысли из головы
2. Распредели по секциям
3. Поставь время задачам - не всем задачам поставить

Технология - Будильник - каждый час звучит будильник - 1 минута - что я делал в предыдущий час - вечером грустим какие мы ультранеэффективные.

Неделя не заканчивается - пока не решена 1 главная задача. Потом будет главная задача дня, месяца, года.

Что будет написана на вашей надгробной табличке?

Есть 72 часа, чтобы применить то, что записали.