

[КАК СТАТЬ АВТОРОМ](#)[Студента не уволишь: чему преподавание учит айтишников и мен...](#)

simust

5 апр 2022 в 00:12

Основы контейнеризации (обзор Docker и Podman)



16 мин



88K

Системное администрирование*, Виртуализация*, DevOps*

[Тutorial](#)

Привет, Хабр!

К 2022 году о контейнеризации не слышал только ленивый. Большинство специалистов, так или иначе имеющих отношение к ИТ, хотя бы раз в жизни запускали программное обеспечение в контейнерах. Однако так ли эта технология проста и понятна? Давайте разбираться вместе!

Главная задача данной статьи – рассказать о контейнеризации, дать ключевые понятия для дальнейшего изучения и показать несколько простых практических приемов. По этой причине (а еще, безусловно, вследствие недостаточной квалификации автора) теоретический материал достаточно упрощен.

История

Идея изоляции пользовательских пространств берет свое начало в 1979 году, когда в ядре UNIX появился системный вызов `chroot`. Он позволял изменить путь каталога корня / для группы процессов на новую локацию в файловой системе, то есть фактически создавал новый корневой каталог, который был изолирован от первого. Следующим шагом и логическим продолжением `chroot` стало создание в 2000 году FreeBSD jails («тюрем»), в которых изначально появилась частичная изоляция сетевых интерфейсов. В первой половине нулевых технологии виртуализации на уровне ОС активно развивались – появились Linux VServer (2001), Solaris Containers (2004) и OpenVZ (2005).

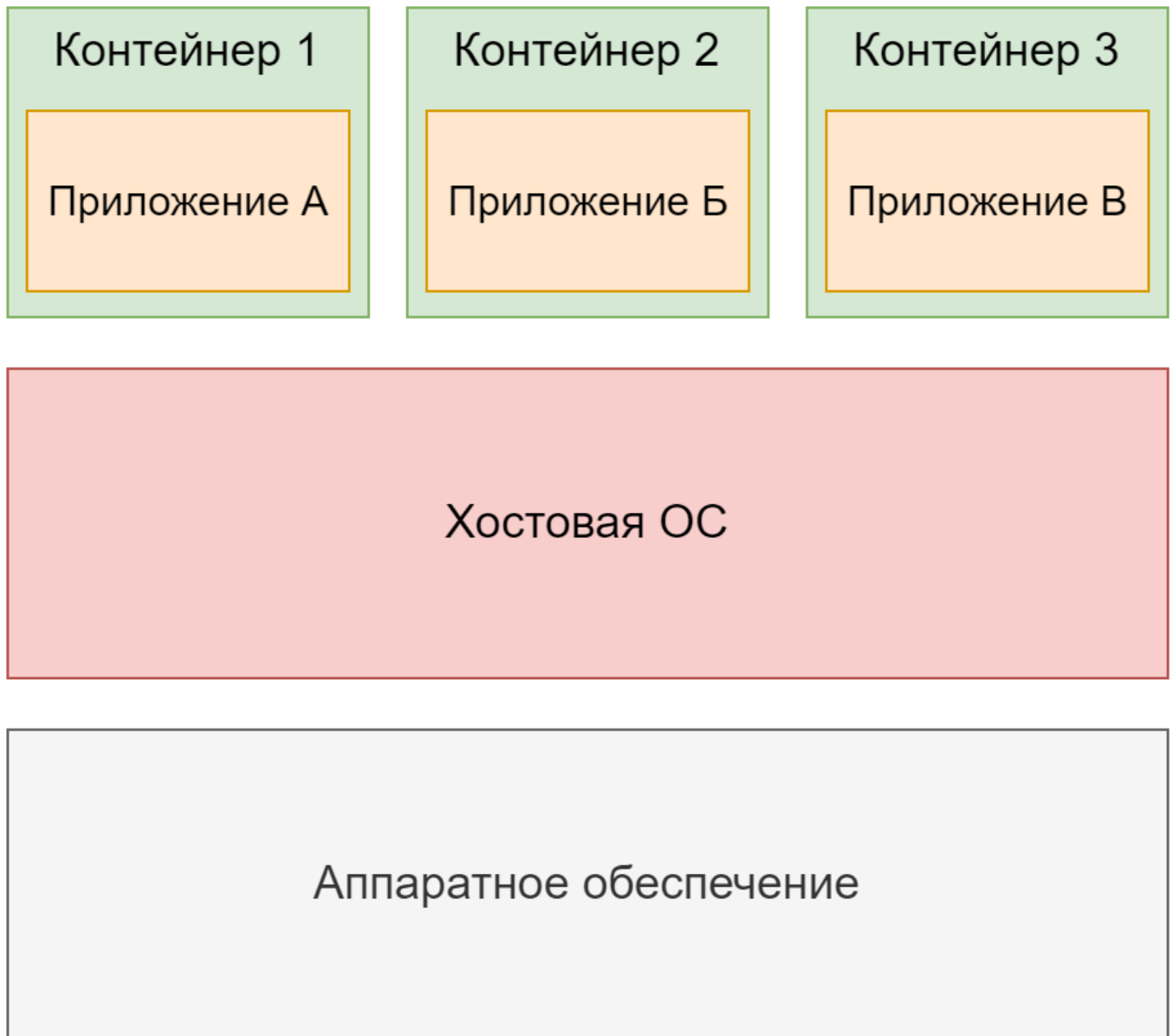
В операционной системе Linux технологии изоляции и виртуализации ресурсов вышли на новый этап в 2002 году, когда в ядро было добавлено первое пространство имен для изоляции файловой системы – `mount`. В 2006–2007 годах компанией Google был разработан механизм Process Containers (позднее переименованный в `cgroups`), который позволил ограничить и изолировать использование группой процессов ЦПУ, ОЗУ и др. аппаратных ресурсов. В 2008 году функционал `cgroups` был добавлен в ядро Linux. Достаточная функциональность для полной изоляции и безопасной работы контейнеров была завершена в 2013 году с добавлением в ядро пространства имен пользователей – `user`.

В 2008 году была представлена система LXC (Linux Containers), которая позволила запускать несколько изолированных Linux систем (контейнеров) на одном сервере. LXC использовала для работы механизмы изоляции ядра – namespaces и cgroups. В 2013 году на свет появилась платформа Docker, невиданно популяризовавшая контейнерные технологии за счет простоты использования и широкого функционала. Изначально Docker использовал LXC для запуска контейнеров, однако позднее перешел на собственную библиотеку libcontainer, также завязанную на функционал ядра Linux. Наконец, в 2015 появился проект Open Container Initiative (OCI), который регламентирует и стандартизирует развитие контейнерных технологий по сей день.

[Читать подробнее:](#) Недостаточное введение в контейнеризацию

Что такое контейнеры?

Контейнеризация (виртуализация на уровне ОС) – технология, которая позволяет запускать программное обеспечение в изолированных на уровне операционной системы пространствах. Контейнеры являются наиболее распространенной формой виртуализации на уровне ОС. С помощью контейнеров можно запустить несколько приложений на одном сервере (хостовой машине), изолируя их друг от друга.



Процесс, запущенный в контейнере, выполняется внутри операционной системы хостовой машины, но при этом он изолирован от остальных процессов. Для самого процесса это выглядит так, будто он единственный работает в системе.

Механизмы изоляции контейнеров

Изоляция процессов в контейнерах осуществляется благодаря двум механизмам ядра Linux – пространствам имен (namespaces) и контрольным группам (cgroups).

Пространства имен гарантируют, что процесс будет работать с собственным представлением системы. Существует несколько типов пространств имен:

- файловая система (mount, mnt) – изолирует файловую систему
- UTS (UNIX Time-Sharing, uts) – изолирует хостнейм и доменное имя
- идентификатор процессов (process identifier, pid) – изолирует процессы

- сеть (network, net) – изолирует сетевые интерфейсы
- межпроцессное взаимодействие (ipc) – изолирует конкурирующее взаимодействие процессами
- пользовательские идентификаторы (user) – изолирует ID пользователей и групп

Процесс принадлежит не одному пространству имен, а одному пространству имен каждого типа.

Контрольные группы гарантируют, что процесс не будет конкурировать за ресурсы, зарезервированные за другими процессами. Они ограничивают (контролируют) объем ресурсов, который процесс может потреблять – ЦПУ, ОЗУ, пропускную способность сети и др.

Читать подробнее:

- Механизмы контейнеризации: namespaces
- Механизмы контейнеризации: cgroups

Основные понятия

Container image (образ) – файл, в который упаковано приложение и его среда. Он содержит файловую систему, которая будет доступна приложению, и другие метаданные (например команды, которые должны быть выполнены при запуске контейнера). Образы контейнеров состоят из слоев (как правило один слой – одна инструкция). Разные образы могут содержать одни и те же слои, поскольку каждый слой надстроен поверх другого образа, а два разных образа могут использовать один и тот же родительский образ в качестве основы. Образы хранятся в Registry Server (реестре) и версионируются с помощью tag (тегов). Если тег не указан, то по умолчанию используется latest. Примеры: Ubuntu, Postgres, NGINX.

Registry Server (реестр, хранилище) – это репозиторий, в котором хранятся образы. После создания образа на локальном компьютере его можно отправить (push) в хранилище, а затем извлечь (pull) на другом компьютере и запустить его там. Существуют общедоступные и закрытые реестры образов. Примеры: Docker Hub (репозитории docker.io), RedHat Quay.io (репозитории quay.io).

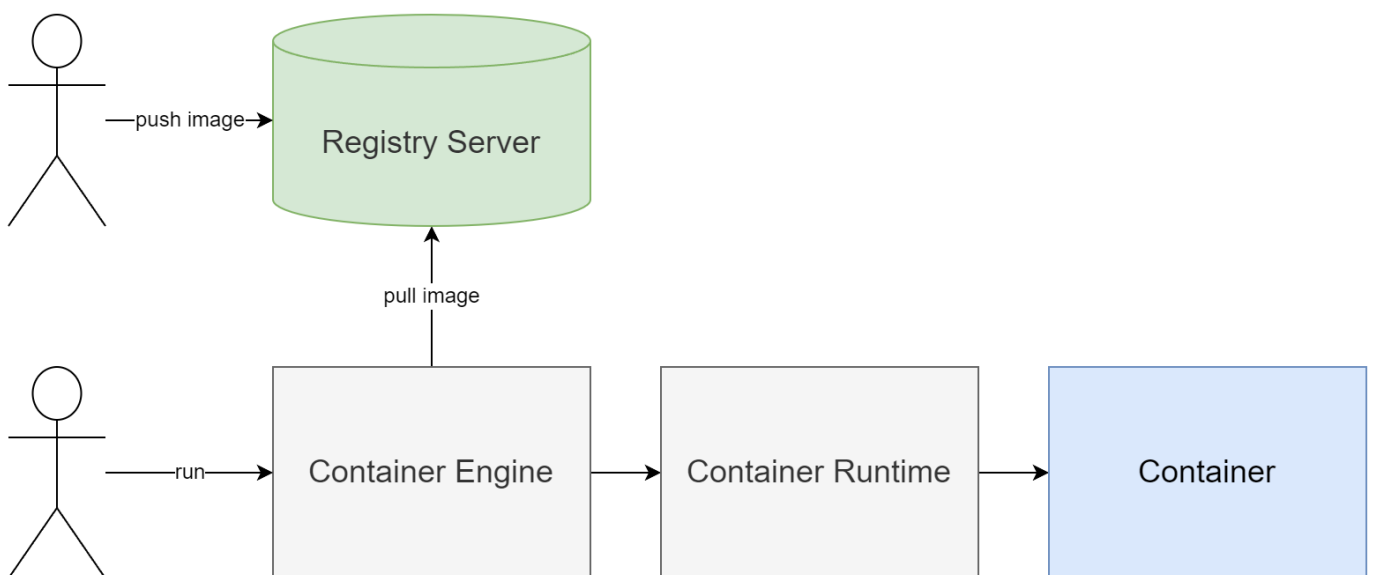
Container (контейнер) – это экземпляр образа контейнера. Выполняемый контейнер – это запущенный процесс, изолированный от других процессов на сервере и ограниченный выделенным объемом ресурсов (ЦПУ, ОЗУ, диска и др.). Выполняемый контейнер сохраняет все слои образа с доступом на чтение и формирует сверху свой исполняемый слой с доступом на запись.

Container Engine (движок контейнеризации) – это программная платформа для упаковки, распространения и выполнения приложений, которая скачивает образы и с пользовательской точки зрения запускает контейнеры (на самом деле за создание и запуск контейнеров отвечает Container Runtime). Примеры: Docker, Podman.

Container Runtime (среда выполнения контейнеров) – программный компонент для создания и запуска контейнеров. Примеры: runc (инструмент командной строки, основанный на упоминавшейся выше библиотеке libcontainer), crun.

Host (хост) – сервер, на котором запущен Container Engine и выполняются контейнеры.

Open Container Initiative (OCI) – это проект Linux Foundation, основанный в 2015 году компанией Docker, Inc, целью которого является разработка стандартов контейнеризации. В настоящее время в проекте участвуют такие компании, как Google, RedHat, Microsoft и др. OCI поддерживает спецификации image-spec (формат образов) и runtime-spec (Container Runtime).



Читать подробнее:

- A Practical Introduction to Container Terminology
- A Comprehensive Container Runtime Comparison
- Различия между Docker, containerd, CRI-O и runc

Подсказки перед практикой

На практике при работе с контейнерами могут быть полезны следующие советы:

- Простейший сценарий – скачать образ, создать контейнер и запустить его (выполнить команду внутри)
- Документацию по запуску контейнера (путь к образу и необходимые команды с ключами) как правило можно найти в реестре образов (например, у Docker Hub есть очень удобный поисковик) или в ReadMe репозитория с исходным кодом проекта. Создать образ и сохранить его в публичный реестр может практически каждый, поэтому старайтесь пользоваться только официальной документацией и проверенными образами! Примеры: Docker Hub/nginx, Docker Hub/debian, GitHub Readme/prometheus
- Для скачивания образов используется команда `pull`, однако в целом она необязательна – при выполнении большинства команд (`create`, `run` и др.) образ скачается автоматически, если не будет обнаружен локально
- При выполнении команд `pull`, `create`, `run` и др. следует указывать репозиторий и тег образа. Если этого не делать, то будут использоваться значения по умолчанию – репозиторий как правило `docker.io`, а тег `latest`
- При запуске контейнера выполняется команда по умолчанию (точка входа), однако можно выполнить и другую команду

Работа с Docker

Docker – это открытая платформа для разработки, доставки и запуска приложений. Состоит из утилиты командной строки `docker`, которая вызывает одноименный сервис (сервис является потенциальной единой точкой отказа) и требует права доступа `root`. По умолчанию использует в качестве Container Runtime `runc`. Все файлы Docker (образы, контейнеры и др.) по умолчанию хранятся в каталоге `/var/lib/docker`.

Для установки необходимо воспользоваться официальным руководством – [Download and install Docker](#), которое содержит подробные инструкции для Linux, Windows и Mac. Стоит сразу отметить, что контейнерам для работы необходимы функции ядра Linux, поэтому они работают нативно под Linux, почти нативно в последних версиях Windows благодаря WSL2 (через Docker Desktop или Linux дистрибутив) и не нативно под Mac (используется виртуализация). Автор рекомендует использовать в тестовой и особенно в промышленной эксплуатации только Linux.

Основные команды

Ниже приведены примеры наиболее распространенных команд:

```
# справочная информация
docker --help # список доступных команд
docker <command> --help # информация по команде
```

```
docker --version # версия Docker
docker info # общая информация о системе

# работа с образами
docker search debian # поиск образов по ключевому слову debian

docker pull ubuntu # скачивание последней версии (тег по умолчанию latest) офици
docker pull prom/prometheus # скачивание последней версии (latest) образа prometi
docker pull docker.io/library/ubuntu:18.04 # скачивание из репозитория docker.io

docker images # просмотр локальных образов

docker rmi <image_name>:<tag> # удаление образа. Вместо <image_name>:<tag> можно
docker rmi $(docker images -aq) # удаление всех образов

# работа с контейнерами
docker run hello-world # Hello, world! в мире контейнеров
docker run -it ubuntu bash # запуск контейнера ubuntu и выполнение команды bash
docker run --name docker-getting-started --publish 8080:80 docker/getting-started
docker run --detach --name mongodb docker.io/library/mongo:4.4.10 # запуск контейн

docker ps # просмотр запущенных контейнеров
docker ps -a # просмотр всех контейнеров (в том числе остановленных)
docker stats --no-stream # просмотр статистики

docker start alpine # создание контейнера из образа alpine

docker start <container_name> # запуск созданного контейнера. Вместо <container_n
docker start $(docker ps -a -q) # запуск всех созданных контейнеров

docker stop <container_name> # остановка контейнера. Вместо <container_name> можн
docker stop $(docker ps -a -q) # остановка всех контейнеров

docker rm <container_name> # удаление контейнера. Вместо <container_name> можно
docker rm $(docker ps -a -q) # удаление всех контейнеров

# система
docker system info # общая информация о системе (соответствует docker info)
docker system df # занятое место на диске
docker system prune -af # удаление неиспользуемых данных и очистка диска
```

Обязательно пробуйте команды на практике, при необходимости прибегая к help или руководствам в Интернете.

Хранение данных

При запуске контейнер получает доступ на чтение ко всем слоям образа, а также создает свой исполняемый слой с возможностью создавать, обновлять и удалять файлы. Все эти изменения не будут видны для файловой системы хоста и других контейнеров, даже если они используют тот же базовый образ. При удалении контейнера все измененные данные также будут удалены. В большинстве случаев это предпочтительное поведение, однако иногда данные необходимо расшарить между несколькими контейнерами или просто сохранить.

Рассмотрим два способа хранения данных контейнеров:

- **named volumes** – именованные тома хранения данных
Позволяет сохранять данные в именованный том, который располагается в каталоге в `/var/lib/docker/volumes` и не удаляется при удалении контейнера. Том может быть подключен к нескольким контейнерам
- **bind mount** – монтирование каталога с хоста
Позволяет монтировать файл или каталог с хоста в контейнер. На практике используется для проброса конфигурационных файлов или каталога БД внутрь контейнера

Ниже приведены примеры использования `named volume` и `bind mount`:

```
# справочная информация
docker <command> --help

# named volume
docker run --detach --name jenkins --publish 80:8080 --volume=jenkins_home:/var/
docker volume ls # просмотр томов
docker volume prune # удаление неиспользуемых томов и очистка диска. Для удаления

# bind mount
# запуск контейнера node-exporter с монтированием каталогов внутрь контейнера в
docker run \
-p 9100:9100 \
-v "/proc:/host/proc:ro" \
-v "/sys:/host/sys:ro" \
-v "":"/rootfs:ro" \
--name node-exporter prom/node-exporter:v1.1.2
```

Обязательно пробуйте команды на практике, при необходимости прибегая к `help` или руководствам в Интернете.

[Читать подробнее:](#) [Хранение данных в Docker](#)

Создание образа (Dockerfile)

Создание и распространение образов – одна из основных задач Docker. Рассмотрим два способа создания образа:

- `commit` изменений из контейнера
Необходимо запустить контейнер из базового образа в интерактивном режиме, внести изменения и сохранить результат в образ с помощью команды `commit`. На практике способ удобен для небольших быстрых доработок
- декларативное описание через `Dockerfile`
Основной способ создания образов. Необходимо создать файл `Dockerfile` с декларативным описанием в формате `yaml` через текстовый редактор и запустить сборку образа командой `build`

Ниже приведены примеры использования `commit` и `build`:

```
# справочная информация
docker <command> --help

# commit
# запуск контейнера из образа ubuntu в интерактивном режиме, установка утилиты ping
docker run -it --name ubuntu-ping ubuntu:20.04 bash
apt update && apt install -y iputils-ping
exit
docker commit ubuntu-ping ubuntu-ping:20.04
docker images

# Dockerfile
# создание файла Dockerfile декларативного описания
FROM ubuntu:20.04
RUN apt-get update && apt-get install -y iputils-ping

# запуск команды build из каталога с Dockerfile для создания образа simust/ubuntu-ping
docker build -t ubuntu-ping:20.04 .
docker images

# tag, login, push
docker tag ubuntu-ping:20.04 simust/ubuntu-ping:20.04 # создание из локального образа
docker images
# вход в репозиторий docker.io под пользователем simust и отправка образа
```

```
docker login -u simust docker.io
docker push simust/ubuntu-ping:20.04
```

Обязательно пробуйте команды на практике, при необходимости прибегая к help или руководствам в Интернете.

Читать подробнее: [Изучаем Docker, часть 3: файлы Dockerfile.](#)

Мультиконтейнерные приложения (Docker Compose)

Docker Compose – это инструмент для декларативного описания и запуска приложений, состоящих из нескольких контейнеров. Он использует yaml файл для настройки сервисов приложения и выполняет процесс создания и запуска всех контейнеров с помощью одной команды. Утилита docker-compose позволяет выполнять команды на нескольких контейнерах одновременно – создавать образы, масштабировать контейнеры, запускать остановленные контейнеры и др.

Читать подробнее: [Руководство по Docker Compose для начинающих](#)

Работа с Podman

Podman – это инструмент с открытым исходным кодом для поиска, сборки, передачи и запуска приложений. Является утилитой командной строки с аналогичными docker командами, однако не требует дополнительный сервис для работы и может работать без прав доступа root. По умолчанию использует в качестве Container Runtime crun (ранее runc).

Возможность работать с контейнерами без прав root приводит к нескольким особенностям:

- все файлы Podman (образы, контейнеры и др.) пользователей с правами доступа root хранятся в каталоге /var/lib/containers, без прав доступа root – в ~/.local/share/containers
- пользователи без root прав по умолчанию не могут использовать привилегированные порты и полноценно использовать некоторые команды

Для установки необходимо воспользоваться официальным руководством – Podman Installation Instructions, которое содержит инструкции для Linux, Windows и Mac. Стоит сразу отметить, что контейнерам для работы необходимы функции ядра Linux, поэтому они работают нативно под Linux, почти нативно в последних версиях Windows благодаря WSL2 (через Linux дистрибутив – не забудьте про wsl --set-default-version 2) и не нативно под Mac (используется виртуализация). Автор рекомендует использовать в тестовой и особенно в промышленной эксплуатации только Linux.

Основные команды

Основные команды для docker идентичны командам для podman, однако есть и приятные доработки (например, ключ `--all` для команд `start`, `stop`, `rm`, `rmi`). Формат образов также совместим благодаря спецификации OCI.

Ниже приведены примеры наиболее распространенных команд:

```
# справочная информация
podman --help # список доступных команд
podman <command> --help # информация по команде

# работа с образами
podman search nginx # поиск образов по ключевому слову nginx

podman pull ubuntu # скачивание последней версии (тег по умолчанию latest) офици
podman pull quay.io/bitnami/nginx:latest # скачивание последней версии образа ng
podman pull docker.io/library/ubuntu:18.04 # скачивание из репозитория docker.io

podman images # просмотр локальных образов

podman rmi <image_name>:<tag> # удаление образа. Вместо <image_name>:<tag> можно
podman rmi --all # удаление всех образов

# работа с контейнерами
podman run hello-world # Hello, world! в мире контейнеров
podman run -it ubuntu bash # запуск контейнера ubuntu и выполнение команды bash
podman run --detach --name nginx --publish 9090:8080 quay.io/bitnami/nginx:1.20.1
podman run --detach --name mongodb docker.io/library/mongo:4.4.10 # запуск контейнера

podman ps # просмотр запущенных контейнеров
podman ps -a # просмотр всех контейнеров (в том числе остановленных)
podman stats --no-stream # просмотр статистики. Если у пользователя нет прав дос

podman create alpine # создание контейнера из образа alpine

podman start <container_name> # запуск созданного контейнера. Вместо <container_
podman start --all # запуск всех созданных контейнеров

podman stop <container_name> # остановка контейнера. Вместо <container_name> можн
podman stop --all # остановка всех контейнеров

podman rm <container_name> # удаление контейнера. Вместо <container_name> можно
podman rm --all # удаление всех контейнеров
```

```
# система
podman system info # общая информация о системе
podman system df # занятое место на диске
podman system prune -af # удаление неиспользуемых данных и очистка диска
```

Хранение данных

При запуске контейнер получает доступ на чтение ко всем слоям образа, а также создает свой исполняемый слой с возможностью создавать, обновлять и удалять файлы. Все эти изменения не будут видны для файловой системы хоста и других контейнеров, даже если они используют тот же базовый образ. При удалении контейнера все измененные данные также будут удалены. В большинстве случаев это предпочтительное поведение, однако иногда данные необходимо расшарить между несколькими контейнерами или просто сохранить.

Рассмотрим два способа хранения данных контейнеров:

- **named volumes** – именованные тома хранения данных
Позволяет сохранять данные в именованный том, который располагается в каталоге в `/var/lib/containers/storage/volumes` или `~/.local/share/containers/storage/volumes` и не удаляется при удалении контейнера. Том может быть подключен к нескольким контейнерам
- **bind mount** – монтирование каталога с хоста
Позволяет монтировать файл или каталог с хоста в контейнер. На практике используется для проброса конфигурационных файлов или каталога БД внутрь контейнера

Ниже приведены примеры использования `named volume` и `bind mount`:

```
# справочная информация
podman <command> --help

# named volume
podman run --detach --name jenkins --publish 8080:8080 --volume=jenkins_home:/var
podman volume ls # просмотр томов
podman volume prune # удаление неиспользуемых томов и очистка диска. Для удаления

# bind mount
# запуск контейнера node-exporter с монтированием каталогов внутрь контейнера в j
podman run \
-p 9100:9100 \
-v "/proc:/host/proc:ro" \
```

```
-v "/sys:/host/sys:ro" \  
-v "":"/rootfs:ro" \  
--name node-exporter docker.io/prom/node-exporter:v1.1.2
```

Создание образов (Containerfile)

Создание и распространение образов – одна из основных задач Podman. Рассмотрим три способа создания образа:

- `commit` изменений из контейнера
Необходимо запустить контейнер из базового образа в интерактивном режиме, внести изменения и сохранить результат в образ с помощью команды `commit`. На практике способ удобен для небольших быстрых доработок
- декларативное описание через `Containerfile`
Необходимо создать файл `Containerfile` с декларативным описанием в формате `yaml` через текстовый редактор и запустить сборку образа командой `build`. `Containerfile` и `Dockerfile` полностью идентичны и взаимозаменяемы
- сборка через утилиту `buildah`
Читать подробнее: [Introduction Tutorial](#)

Ниже приведены примеры использования `commit` и `build`:

```
# справочная информация  
podman <command> --help  
  
# commit  
# запуск контейнера из образа ubuntu в интерактивном режиме, установка утилиты ping  
podman run -it --name ubuntu-ping ubuntu:20.04 bash  
apt update && apt install -y iputils-ping  
exit  
podman commit ubuntu-ping simust/ubuntu-ping:20.04  
podman tag ubuntu-ping:20.04 ubuntu-ping:20.04  
  
# Containerfile  
# создание файла Containerfile декларативного описания  
FROM ubuntu:20.04  
RUN apt-get update && apt-get install -y iputils-ping  
  
# запуск команды build из каталога с Containerfile для создания образа ubuntu-ping  
podman build -t ubuntu-ping:20.04 .  
  
# tag, login, push
```

```
podman tag ubuntu-ping:20.04 quay.io/simust/ubuntu-ping:20.04 # создание из локал  
# вход в репозиторий quay.io под пользователем simust и отправка образа  
podman login -u simust quay.io  
podman push quay.io/simust/ubuntu-ping:20.04
```

Обязательно пробуйте команды на практике, при необходимости прибегая к help или руководствам в Интернете.

Читать подробнее: Изучаем Docker, часть 3: файлы Dockerfile (Dockerfile = Containerfile)

Мультиконтейнерные приложения (Podman Compose и Podman Pod)

Podman Compose – это инструмент для декларативного описания и запуска приложений, состоящих из нескольких контейнеров. Фактически Podman Compose есть ни что иное, как реализация Docker Compose для Podman с учетом его особенностей (например, возможности работать с контейнерами без прав доступа root). Он использует yaml файл для настройки сервисов приложения и выполняет процесс создания и запуска всех контейнеров с помощью одной команды.

Читать подробнее: Руководство по Docker Compose для начинающих (Docker Compose = Podman Compose)

Podman Pod – это группа из одного или нескольких контейнеров с общим хранилищем и сетевыми ресурсами, а также спецификацией для запуска контейнеров. Концепция подов появилась и реализуется в Kubernetes.

Читать подробнее: Podman: Managing pods and containers in a local container runtime

Подсказки после практики

На практике при работе с контейнерами могут быть полезны следующие советы:

- Для администрирования приложений в контейнерах следует использовать функционал systemd unit
Управлять приложениями в контейнерах как обычными сервисами Linux очень удобно – настройка, запуск, остановка, восстановление при сбоях и др. действия становятся простыми и прозрачными

Читать подробнее: Как запустить Docker / Podman контейнеры в качестве службы Systemd

- Docker или Podman?
Как определить, что лучше использовать – Docker или Podman? Критериев много, однозначного ответа нет, да и разница на сегодняшний день не так велика. Однако автор рекомендует использовать Podman во всех дистрибутивах, где приложила руку RedHat. Ubuntu, Debian и др. – Docker, RHEL, Fedora – Podman



За помощь в подготовке статьи автор выражает искреннюю благодарность @novikov0805, @Eviil и @KoPashka

Все статьи серии:

- 1. Основы Linux (обзор с практическим уклоном)
- 2. Основы виртуализации (обзор)
- 3. Основы контейнеризации (обзор Docker и Podman)
- 4. Основы мониторинга (обзор Prometheus и Grafana)

Только зарегистрированные пользователи могут участвовать в опросе. Войдите, пожалуйста.

По какой теме Вы хотели бы видеть следующую статью?

69.86%	Основы мониторинга (обзор Prometheus и Grafana)	102
54.79%	Основы Ansible	80
1.37%	Другое (напишу в комментариях)	2

Проголосовали 146 пользователей. Воздержались 25 пользователей.

Теги: обучение, контейнеризация, docker, podman

Хабы: Системное администрирование, Виртуализация, DevOps

Редакторский дайджест



Присылаем лучшие статьи раз в месяц

Электронпочта



42

Карма

0

Рейтинг


Семен Устинов @simust
Пользователь

Реклама
АМА


Обувь
уровнем выше
по той же цене

Кроссовки
от 900 Р

Выбрать




Как новые



 Комментарии 22


Публикации


ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ




anton-artemenko
19 часов назад


Идеальные паразиты человека и «тихая пандемия»: привет, ветрянка и герпес


 Простой


 13 мин

 8.1K

Обзор

 +39

 44

 38

16 часов назад

Герои напильника и паяльника: итоги сезона DIY

 8 мин

 6.7K

 Сезон DIY

 Спецпроект

 +38 34 6

spiritus_sancti

17 часов назад

RGB-усилители. Особенности, проблемы, выбор



Простой



6 мин



4.3K

Тutorial

 +38 19 10

ru_vds

13 часов назад

Профилирование Python — почему и где тормозит ваш код



Средний



10 мин



2.3K

Тutorial

Перевод

 +27 74 3

ViktorSergeev

10 часов назад

WebOne: даём жизнь старым браузерам



6 мин



2K

 +21 16 5

Duke_nukeum

22 часа назад

DIY: Большая Красная Кнопка для принятия Очень Важных Решений



Простой



2 мин



4.7K

Тutorial

 +18 19 17

AnnaVIMorozova

13 часов назад

Как повысить эффективность коммуникаций в команде: учимся решать конфликты

 7 мин  1.5K +17 35 2

fedorborovitsky

19 часов назад

Российский софт в идеальном шторме

 6 мин  6.5K[Мнение](#) +17 14 32

TimReset

20 часов назад

Запускаем IDEA и CLion на Android

 Средний  18 мин  2.3K[Тutorial](#) +17 35 7

RationalAnswer

21 час назад

Heineken выкупил за 1 евро производитель туалетных освежителей, а Binance огребают из-за россиян

 8 мин  18K[Дайджест](#) +16 19 235

Никакой магии, мы просто берём выпускников курсов и растим из них тестировщиков

[Турбо](#)[Показать еще](#)

МИНУТОЧКУ ВНИМАНИЯ



Вышел каменный цветок? Поможем рассказать



Проверь свою SQL-экипировку в тесте для аналитиков

ЗАКАЗЫ

Разработка приложения под iOS

40000 руб./за проект · 5 откликов · 30 просмотров

Доработать онлайн обучение

50000 руб./за проект · 6 откликов · 48 просмотров

Мобильная верстка + новые страницы сайт автодилера [next.js][react]

100000 руб./за проект · 61 отклик · 228 просмотров

Диагностирование и решение серверной ошибки, переезд в кубер

80000 руб./за проект · 11 откликов · 87 просмотров

Продублировать существующий kubernetes кластер

5000 руб./за проект · 5 откликов · 35 просмотров

Больше заказов на Хабр Фрилансе

ЧИТАЮТ СЕЙЧАС

Реально ли без опыта в 2023 году найти работу в IT? История одного джуна

👁 36K 💬 78

Теневое правление Илона Маска

👁 3.1K 💬 8

Почему тип поля enum на уровне базы — зло

👁 13K 💬 120

WordPress предложила зарегистрировать домен на 100 лет

4.1K 16

СМИ: Binance допускает полный уход из России


2.2K 13

Никакой магии, мы просто берём выпускников курсов и растим из них тестировщиков

Турбо

ИСТОРИИ

Воспитай айтишника
Подборка статей о том, как научить детей программированию



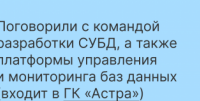
Как учить детей программированию

Ошибки разума
Статьи о когнитивных искажениях и их проявлениях в жизни



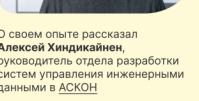
Когнитивные искажения

Где работать в IT: Tantor
Поговорили с командой разработки СУБД, а также платформы управления и мониторинга баз данных (входит в ГК «Астра»)



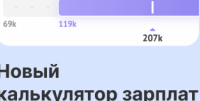
Где работать в IT: Tantor

Кем работать в IT: руководитель разработки
О своем опыте рассказал Алексей Хиндикийкин, руководитель отдела разработки систем управления инженерными данными в АСКОН



Кем работать в IT: руководитель разработки

Новый калькулятор зарплат
Разработчик
206 833 Р
186 000 Р + 20 833 Р
зарплата премия
69% 119% 207k



Калькулятор зарплат 2.0

РАБОТА

DevOps инженер
50 вакансий

Системный администратор
101 вакансия

Все вакансии

Реклама



Ваш аккаунт	Разделы	Информация	Услуги
Войти	Статьи	Устройство сайта	Корпоративный блог
Регистрация	Новости	Для авторов	Медийная реклама
	Хабы	Для компаний	Нативные проекты
	Компании	Документы	Образовательные
	Авторы	Соглашение	программы
	Песочница	Конфиденциальность	Стартапам
			Спецпроекты



Настройка языка

Техническая поддержка

Вернуться на старую версию

