

КАК СТАТЬ АВТОРОМ

Поднял стриминговую обработку – опиши в новом сезоне Java

 **il_da_r**
8 фев 2021 в 11:30

Недостающее введение в контейнеризацию

13 мин 39K

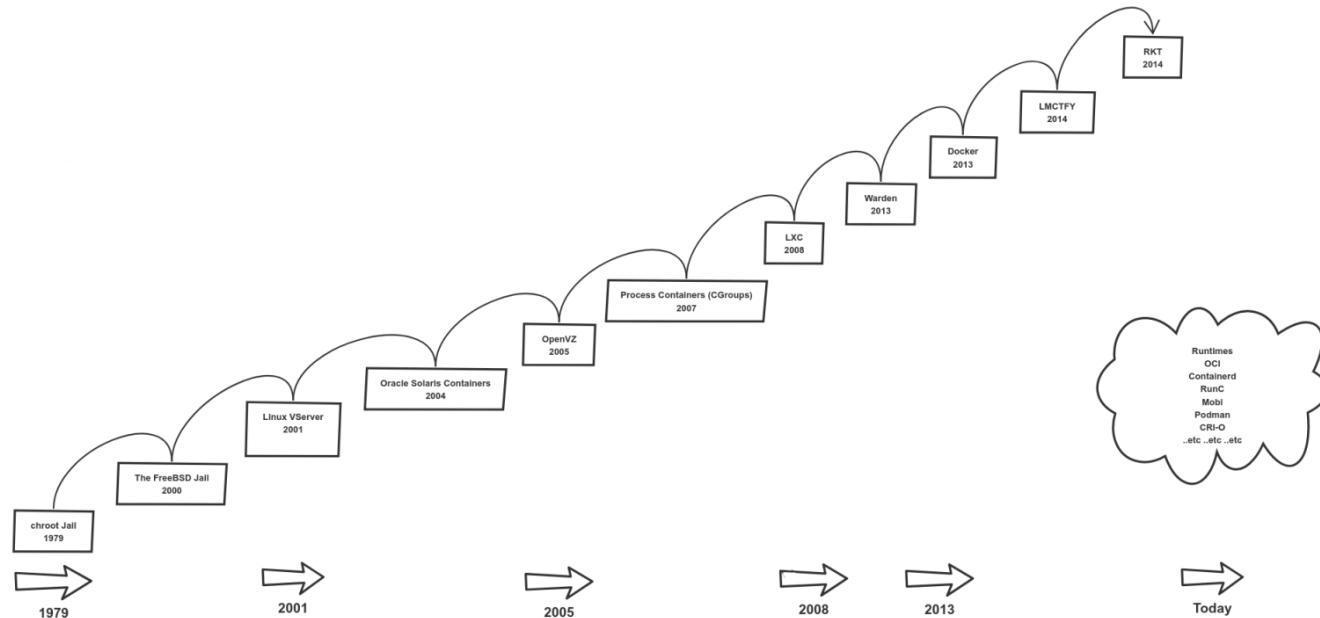
Системное администрирование*, Виртуализация*, DevOps*, Kubernetes*

Перевод

Автор оригинала: Аүмен Еоп Амғі

Эта статья помогла мне немного углубится в устройство и принцип работы контейнеров. Поэтому решил ее перевести. "Экосистема контейнеров иногда может сбивать с толку, этот пост может помочь вам понять некоторые запутанные концепции Docker и контейнеров. Мы также увидим, как развивалась экосистема контейнеров". Статья 2019 года.

Docker - одна из самых известных платформ контейнеризации в настоящее время, она была выпущена в 2013 году. Однако использование изоляции и контейнеризации началось раньше. Давайте вернемся в 1979 год, когда мы начали использовать Chroot Jail, и посмотрим на самые известные технологии контейнеризации, появившиеся после. Это поможет нам понять новые концепции.



Все началось с того, что **Chroot Jail** и системный вызов **Chroot** были введены во время

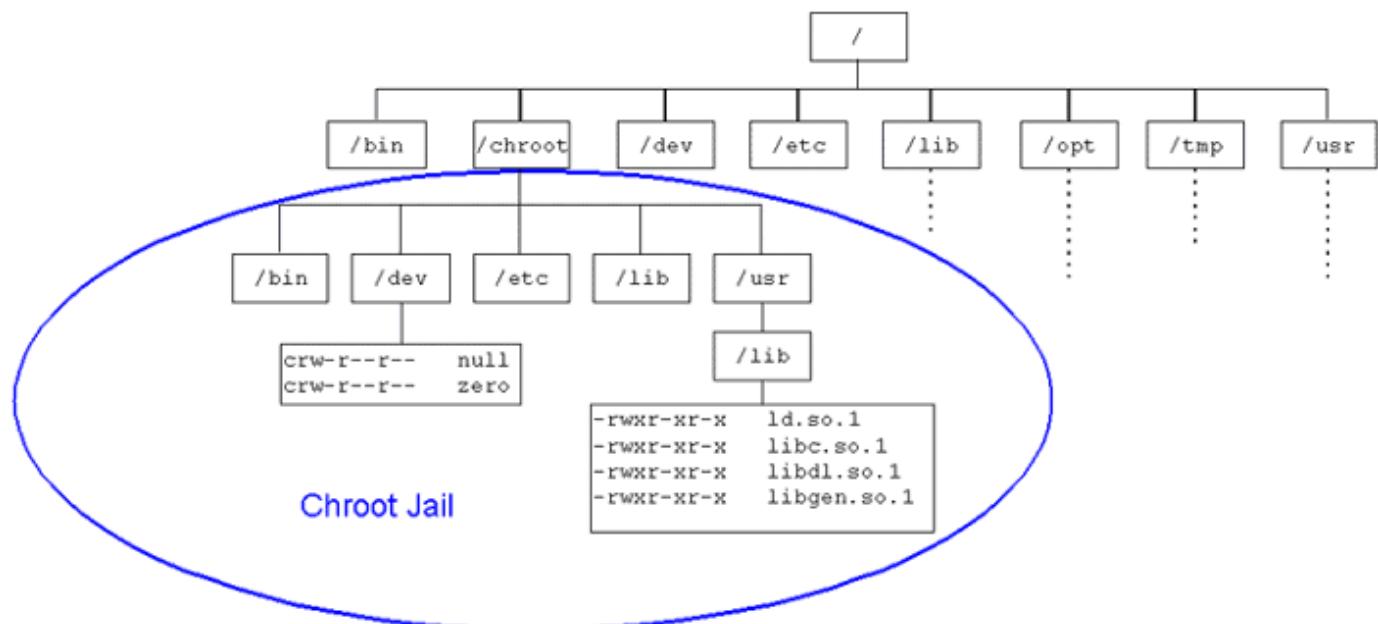
+19

206



2

считается одной из первых технологий контейнеризации. Он позволяет изолировать процесс и его дочерние элементы от остальной части операционной системы. Единственная проблема с этой изоляцией заключается в том, что корневой процесс может легко выйти из chroot. В нем никогда не задумывались механизмы безопасности. **FreeBSD Jail** была представлена в ОС FreeBSD в 2000 году и была предназначена для обеспечения большей безопасности простой изоляции файлов Chroot. В отличие от Chroot, реализация FreeBSD также изолирует процессы и их действия от Файловой системы.



Chroot Jail. Источник: <https://linuxhill.wordpress.com/2014/08/09/014-setting-up-a-chroot-jail-in-crunchbang-11debian-wheezy>

Когда в ядро Linux были добавлены возможности виртуализации на уровне операционной системы, в 2001 году был представлен **Linux VServer**, который использовал chroot-подобный механизм в сочетании с «security contexts (контекстами безопасности)», так и виртуализацию на уровне операционной системы. Он более продвинутый, чем простой chroot, и позволяет запускать несколько дистрибутивов Linux на одном VPS.



В феврале 2004 года Sun (позже приобретенная Oracle) выпустила (Oracle) Solaris Containers, реализацию Linux-Vserver для процессоров X86 и SPARC. Контейнер Solaris - <https://habr.com/ru/articles/541288/>

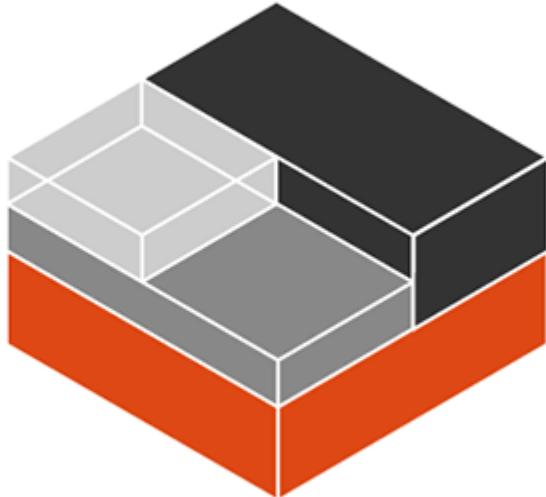
Это комбинация элементов управления ресурсами системы и разделения ресурсов, обеспечиваемых «zone».

Подобно контейнерам Solaris, первая версия OpenVZ была представлена в 2005 году. OpenVZ, как и Linux-VServer, использует виртуализацию на уровне ОС и был принят многими хостинговыми компаниями для изоляции и продажи VPS. Виртуализация на уровне ОС имеет некоторые ограничения, поскольку контейнеры и хост используют одну и ту же архитектуру и версию ядра, недостаток возникает в ситуациях, когда гостям требуются версии ядра, отличные от версии на хосте. Linux-VServer и OpenVZ требуют патча ядра, чтобы добавить некоторые механизмы управления, используемые для создания изолированного контейнера. Патчи OpenVZ не были интегрированы в ядро.



В 2007 году Google выпустил CGroups - механизм, который ограничивает и изолирует использование ресурсов (ЦП, память, дисковый ввод-вывод, сеть и т. д.) для набора процессов. CGroups были, в отличие от ядра OpenVZ, встроены в ядро Linux в 2007 году.

В 2008 году была выпущена первая версия LXC (Linux Containers). LXC похож на OpenVZ, Solaris Containers и Linux-VServer, однако он использует CGroups, которые уже реализованы в ядре Linux. Затем в 2013 году компания CloudFoundry создала Warden - API для управления изолированными, эфемерными средами с контролируемыми ресурсами. В своих первых версиях Warden использовал LXC.



LXC

В 2013 году была представлена первая версия **Docker**. Он выполняет виртуализацию на уровне операционной системы, как и контейнеры OpenVZ и Solaris.

В 2014 году Google представил **LMCTFY**, версию стека контейнеров Google с открытым исходным кодом, которая предоставляет контейнеры для приложений Linux. Инженеры Google сотрудничают с Docker над libcontainer и переносят основные концепции и абстракции в libcontainer. Проект активно не развивается, и в будущем ядро этого проекта, вероятно, будет заменено libcontainer.

LMCTFY запускает приложения в изолированных средах на том же ядре и без патчей, поскольку он использует CGroups, namespaces и другие функции ядра Linux.



Фото Павла Червильского для Unsplash

Google - лидер в контейнерной индустрии. Все в Google работает на контейнерах. Каждую неделю в инфраструктуре Google работает более 2 миллиардов контейнеров.

В декабре 2014 года CoreOS выпустила и начала поддерживать rkt (первоначально выпущенную как Rocket) в качестве альтернативы Docker.



Jails, VPS, Zones, контейнеры и виртуальные машины

Изоляция и управление ресурсами являются общими целями использования Jail, Zone, VPS, виртуальных машин и контейнеров, но каждая технология использует разные способы достижения этого, имеет свои ограничения и свои преимущества.

До сих пор мы вкратце видели, как работает Jail, и представили, как Linux-VServer позволяет запускать изолированные пользовательские пространства, в которых программы запускаются непосредственно в ядре операционной системы хоста, но имеют доступ к ограниченному подмножеству его ресурсов.

Linux-VServer позволяет запускать VPS, и для его использования необходимо пропатчить ядро хоста.

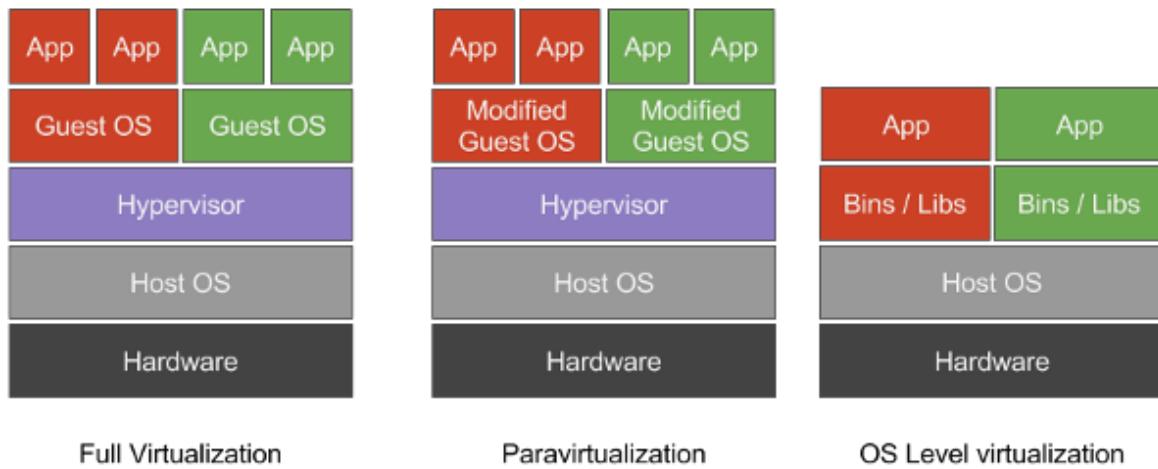
Контейнеры Solaris называются Zones.

«Виртуальная машина» - это общий термин для описания эмулируемой виртуальной машины поверх «реальной аппаратной машины». Этот термин был первоначально определен Попеком и Голдбергом как эффективная изолированная копия реальной компьютерной машины.

Виртуальные машины могут быть «**System Virtual Machines** (системными виртуальными машинами)» или «**Process Virtual Machines** (процессными виртуальными машинами)». В повседневном использовании под словом «виртуальные машины» мы обычно имеем в виду «системные виртуальные машины», которые представляют собой эмуляцию оборудования хоста для эмуляции всей операционной системы. Однако «**Process Virtual Machines**», иногда называемый «**Application Virtual Machine** (Виртуальной машиной приложения)», используется для имитации среды программирования для выполнения отдельного процесса: примером является виртуальная машина Java.

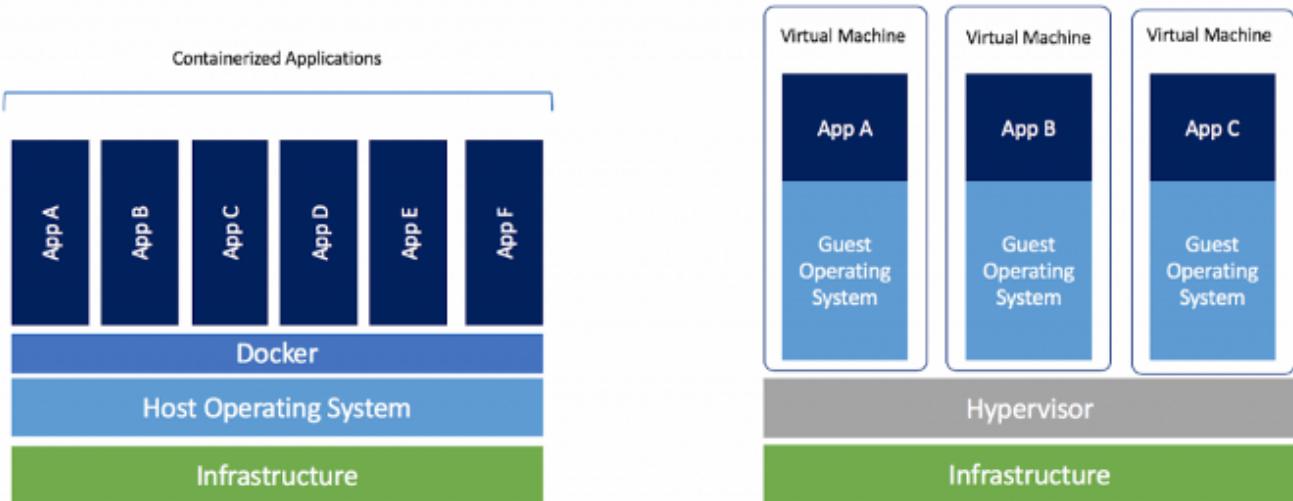
Виртуализация на уровне ОС также называется контейнеризацией. Такие технологии, как Linux-VServer и OpenVZ, могут запускать несколько операционных систем, используя одну и ту же архитектуру и версию ядра.

Совместное использование одной и той же архитектуры и ядра имеет некоторые ограничения и недостатки в ситуациях, когда гостям требуются версии ядра, отличные от версии хоста.



Источник: <https://fntlnz.wtf/post/why-containers>

Системные контейнеры (например, LXC) предлагают среду, максимально приближенную к той, которую вы получаете от виртуальной машины, но без накладных расходов, связанных с запуском отдельного ядра и имитацией всего оборудования.



VM vs Container. Источник: Docker Blog

Контейнеры ОС vs контейнеры приложений

Виртуализация на уровне ОС помогает нам в создании контейнеров. Такие технологии, как LXC и Docker, используют этот тип изоляции.

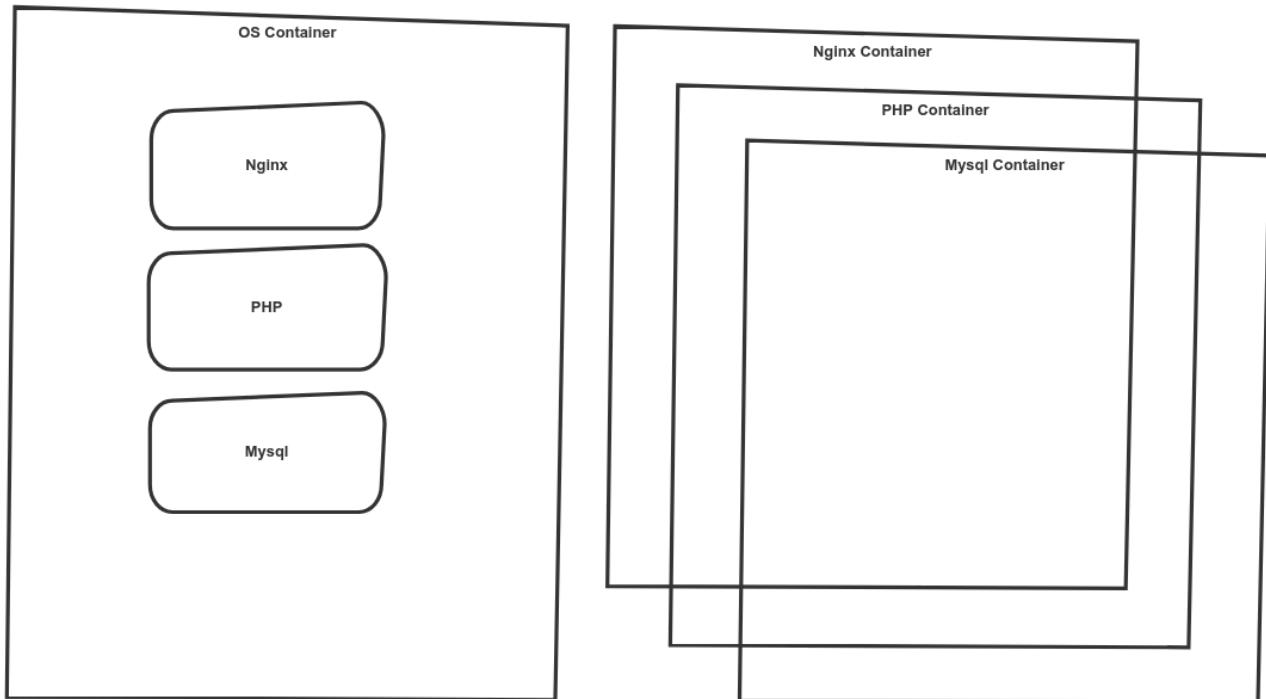
Здесь у нас есть два типа контейнеров:

- Контейнеры ОС, в которые упакована операционная система со всем стеком приложений (пример LEMP).

- Контейнеры приложений, которые обычно запускают один процесс для каждого контейнера.

В случае с контейнерами приложений у нас будет 3 контейнера для создания стека LEMP:

- сервер PHP (или PHP FPM).
- Веб-сервер (Nginx).
- Mysql.



OS Containers

- Run as an OS
- Run multiple services in the same container
- Use native resource isolation (Linux facilities)

Examples:

- LXC
- OpenVZ
- Linux-VServer
- Solaris Containers

App Containers

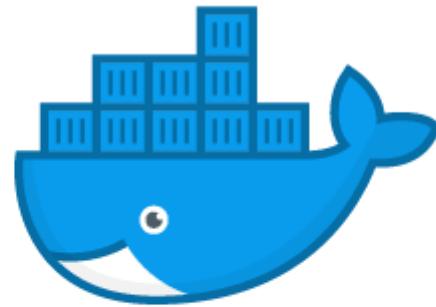
- Run as an isolated application
- Run a single process/services per container
- Was built on top of OS Containers

Examples

- Docker
- RKT

Докер: контейнер или платформа?

Коротко: и то и другое



docker

Подробный ответ:

Когда Docker начал использовать LXC в качестве среды выполнения контейнера, идея заключалась в том, чтобы создать API для управления средой выполнения контейнера, изолировать отдельные процессы, выполняющие приложения, и контролировать жизненный цикл контейнера и ресурсы, которые он использует. В начале 2013 года проект Docker должен был создать «стандартный контейнер», как мы можем видеть в этом манифесте.

Solomon Hykes committed on Feb 1, 2013

1 parent de1c361 commit 0db56e6c519b19ec16c6fb12e3ce7dfa6018c5

Showing 1 changed file with 43 additions and 0 deletions.

Unified Split

43 README.md

... (0) -1, 3 +1, 46 (0)

```

1 +Docker: a self-sufficient runtime for linux containers
2 +=====
3 +
4 +Docker is a runtime for Standard Containers. More specifically, it is a daemon which automates the creation of and deployment of
linux Standard Containers (SCs) via a remote API.
5 +
6 +Standard Containers are a fundamental unit of software delivery, in much the same way that shipping containers
(http://bricks.arge.com/ins/7823-1/12) are a fundamental unit of physical delivery.
7 +
8 +
9 +1. STANDARD OPERATIONS
10 +-----
11 +
12 +Just like shipping containers, Standard Containers define a set of STANDARD OPERATIONS. Shipping containers can be lifted,
stacked, locked, loaded, unloaded and labelled. Similarly, standard containers can be started, stopped, copied, snapshotted,
downloaded, uploaded and tagged.
13 +
14 +
15 +2. CONTENT-AGNOSTIC
16 +-----
17 +
18 +Just like shipping containers, Standard Containers are CONTENT-AGNOSTIC: all standard operations have the same effect regardless
of the contents. A shipping container will be stacked in exactly the same way whether it contains Vietnamese powder coffee or
spare Maserati parts. Similarly, Standard Containers are started or uploaded in the same way whether they contain a postgres
database, a php application with its dependencies and application server, or Java build artifacts.
19 +
20 +
21 +3. INFRASTRUCTURE-AGNOSTIC
22 +-----
23 +
24 +Both types of containers are INFRASTRUCTURE-AGNOSTIC: they can be transported to thousands of facilities around the world,
and manipulated by a wide variety of equipment. A shipping container can be packed in a factory in Ukraine, transported by truck to
the nearest routing center, stacked onto a train, loaded into a German boat by an Australian-built crane, stored in a warehouse
at a US facility, etc. Similarly, a standard container can be bundled on my laptop, uploaded to S3, downloaded, run and
snapshotted by a build server at Equinix in Virginia, uploaded to 10 staging servers in a home-made Openstack cluster, then sent
to 30 production instances across 3 EC2 regions.
25 +

```

Манифест стандартного контейнера был удален.

```

238 -What is a Standard Container?
239 =====
240 -
241 -Docker defines a unit of software delivery called a Standard
242 +Container. The goal of a Standard Container is to encapsulate a
243 -software component and all its dependencies in a format that is
244 -self-describing and portable, so that any compliant runtime can run it
245 -without extra dependencies, regardless of the underlying machine and
246 -the contents of the container.
247 -
248 -The spec for Standard Containers is currently a work in progress, but
249 -it is very straightforward. It mostly defines 1) an image format, 2) a
250 -set of standard operations, and 3) an execution environment.
251 -
252 -A great analogy for this is the shipping container. Just like how
253 -Standard Containers are a fundamental unit of software delivery,
254 -shipping containers are a fundamental unit of physical delivery.
255 -
256 ## 1. STANDARD OPERATIONS
257 -
258 -Just like shipping containers, Standard Containers define a set of
259 STANDARD OPERATIONS. Shipping containers can be lifted, stacked,
260 -locked, loaded, unloaded and labelled. Similarly, Standard Containers
261 -can be started, stopped, copied, snapshoted, downloaded, uploaded and
262 -tagged.
263 -
264 ## 2. CONTENT-AGNOSTIC
265 -
266 -Just like shipping containers, Standard Containers are
267 CONTENT-AGNOSTIC: all standard operations have the same effect
268 -regardless of the contents. A shipping container will be stacked in
269 -exactly the same way whether it contains Vietnamese powder coffee or
270 -spare Maserati parts. Similarly, Standard Containers are started or
271 -uploaded in the same way whether they contain a postgres database, a
272 -php application with its dependencies and application server, or Java
273 -build artifacts.
274 -
275 ## 3. INFRASTRUCTURE-AGNOSTIC
276 -
277 -Both types of containers are INFRASTRUCTURE-AGNOSTIC: they can be
278 -transported to thousands of facilities around the world, and
279 -manipulated by a wide variety of equipment. A shipping container can
280 -be packed in a factory in Ukraine, transported by truck to the nearest
281 -routing center, stacked onto a train, loaded into a German boat by an
282 -Australian-built crane, stored in a warehouse at a US facility,
283

```

Docker начал создавать монолитное приложение с множеством функций - от запуска облачных серверов до создания и запуска образов / контейнеров. Docker использовал **libcontainer** для взаимодействия с такими средствами ядра Linux, как **Control Groups** и **Namespaces**.



Давайте создадим контейнер с использованием CGroups и Namespaces

В этом примере я использую Ubuntu, но это должно работать для большинства дистрибутивов. Начните с установки CGroup Tools and утилиты stress, поскольку мы собираемся выполнить некоторые стресс-тесты.

```
sudo apt install cgroup-tools  
sudo apt install stress
```

Эта команда создаст новый контекст исполнения:

```
sudo unshare --fork --pid --mount-proc bash  
ps aux
```

```
eon01@eonSpider0x1:~$ sudo unsh[REDACTED]
```

I

Команда "unshare" разъединяет части контекста исполнения процесса

Теперь, используя *cgroup*, мы можем создать группы управления и определить два контроллера: один в памяти, а другой - в процессоре.

```
cgcreate -a $USER -g memory:mygroup -g cpu:mygroup  
ls /sys/fs/cgroup/{memory,cpu}/mygroup
```

```
eon01@eonSpider0x1:~$ sudo unshare --fork --pid --mount-proc bash
root@eonSpider0x1:~# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
root         1  1.0  0.0  22712  5020 pts/1      S    16:13  0:00 bash
root        12  0.0  0.0  37368  3280 pts/1     R+   16:13  0:00 ps aux
root@eonSpider0x1:~# █
```

I

Следующим шагом будет определение лимита памяти и его активация:

```
echo 3000000 > /sys/fs/cgroup/memory/mygroup/memory.kmem.limit_in_bytes
cgexec -g memory:mygroup bash
```

```
eon01@eonSpider0x1:~$ sudo unshare --fork --pid --mount-proc bash
root@eonSpider0x1:~# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  1.0  0.0  22712  5020 pts/1    S   16:13   0:00 bash
root        12  0.0  0.0  37368  3280 pts/1    R+  16:13   0:00 ps aux
root@eonSpider0x1:~# cgcreate -a $USER -g memory:mygroup -g cpu:mygroup
root@eonSpider0x1:~# ls /sys/fs/cgroup/{cpu,memory}/mygroup
/sys/fs/cgroup/cpu/mygroup:
cgroup.clone_children  cputacct.usage_all          cputacct.usage_sys  cpu.shares
cgroup.procs            cputacct.usage_percpu       cputacct.usage_user  cpu.stat
cpuacct.stat            cputacct.usage_percpu_sys  cpu.cfs_period_us  notify_on_release
cpuacct.usage           cputacct.usage_percpu_user  cpu.cfs_quota_us   tasks

/sys/fs/cgroup/memory/mygroup:
cgroup.clone_children      memory.kmem.tcp.failcnt      memory.oom_control
cgroup.event_control       memory.kmem.tcp.limit_in_bytes  memory.pressure_level
cgroup.procs               memory.kmem.tcp.max_usage_in_bytes  memory.soft_limit_in_bytes
memory.failcnt             memory.kmem.tcp.usage_in_bytes  memory.stat
memory.force_empty         memory.kmem.usage_in_bytes   memory.swappiness
memory.kmem.failcnt        memory.limit_in_bytes     memory.usage_in_bytes
memory.kmem.limit_in_bytes  memory.max_usage_in_bytes  memory.use_hierarchy
memory.kmem.max_usage_in_bytes  memory.move_charge_at_immigrate  notify_on_release
memory.kmem.slabinfo        memory.numa_stat          I   tasks
root@eonSpider0x1:~# 
```

Теперь давайте запустим `stress` для изолированного namespace, которое мы создали с ограничениями памяти.

```
stress --vm 1 --vm-bytes 1G --timeout 10s
```

```
root@eonSpider0x1:~#
```

I

Мы можем заметить, что выполнение не удалось, значит ограничение памяти работает. Если мы сделаем то же самое на хост-машине, тест завершится без ошибки, если у вас действительно достаточно свободной памяти:

```
root@eonSpider0x1:~# stress --vm 1 --vm-bytes 1G --timeout 10s
stress: info: [54] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
stress: FAIL: [54] (415) <-- worker 55 got signal 9
stress: WARN: [54] (417) now reaping child worker processes
stress: FAIL: [54] (451) failed run completed in 1s
root@eonSpider0x1:~# stress --vm 1 --vm-bytes 1G --timeout 10s
stress: info: [56] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
stress: FAIL: [56] (415) <-- worker 57 got signal 9
stress: WARN: [56] (417) now reaping child worker processes
stress: FAIL: [56] (451) failed run completed in 0s
root@eonSpider0x1:~# stress --vm 1 --vm-bytes 1G --timeout 10s
stress: info: [58] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
stress: FAIL: [58] (415) <-- worker 59 got signal 9
stress: WARN: [58] (417) now reaping child worker processes
stress: FAIL: [58] (451) failed run completed in 0s
root@eonSpider0x1:~# █
```

I

Выполнение этих шагов поможет понять, как средства Linux, такие как CGroups и другие функции управления ресурсами, могут создавать изолированные среды в системах Linux и управлять ими.

Интерфейс `libcontainer` взаимодействует с этими средствами для управления контейнерами Docker и их запуска.

runC: Использование libcontainer без Docker

В 2015 году Docker анонсировал runC: легкую портативную среду выполнения контейнеров.



гипС - это, по сути, небольшой инструмент командной строки для непосредственного использования libcontainer, без использования Docker Engine.

Цель гипС - сделать стандартные контейнеры доступными повсюду.

Этот проект был передан в дар Open Container Initiative (OCI).

Репозиторий libcontainer сейчас заархивирован. На самом деле, libcontainer не забросили, а перенесли в репозиторий гипС.

Перейдем к практической части и создадим контейнер с помощью гипС. Начните с установки среды выполнения гипС (прим. переводчика: если стоит docker то этого можно (нужно) не делать):

```
sudo apt install runc
```

Давайте создадим каталог (/mycontainer), в который мы собираемся экспортить содержимое образа Busybox.

```
eon01@eonSpider0x1:~$ I
```

```
sudo su
mkdir /mycontainer
cd /mycontainer/
mkdir rootfs
docker export $(docker create busybox) | tar -C rootfs -xvf -
```

Используя команду `runc`, мы можем запустить контейнер `busybox`, который использует извлеченный образ и файл спецификации (`config.json`).

```
root@eonSpider0x1:/mycontainer# ls -l
total 4
drwxr-xr-x 12 root root 4096 Mar 13 17:12 rootfs
root@eonSpider0x1:/mycontainer# ls -l rootfs/
total 48
drwxr-xr-x 2 root root 12288 Feb 14 19:58 bin
drwxr-xr-x 4 root root 4096 Mar 13 17:12 dev
drwxr-xr-x 3 root root 4096 Mar 13 17:12 etc
drwxr-xr-x 2 nobody nogroup 4096 Feb 14 19:58 home
drwxr-xr-x 2 root root 4096 Mar 13 17:12 proc
drwxr-xr-x 2 root root 4096 Feb 14 19:58 root
drwxr-xr-x 2 root root 4096 Mar 13 17:12 sys
drwxrwxrwt 2 root root 4096 Feb 14 19:58 tmp
drwxr-xr-x 3 root root 4096 Feb 14 19:58 usr
drwxr-xr-x 4 root root 4096 Feb 14 19:58 var
root@eonSpider0x1:/mycontainer# I
```

```
runc spec
runc run mycontainerid
```

```
root@eonSpider0x1:/mycontainer# ls -l
total 4
drwxr-xr-x 12 root root 4096 Mar 13 17:12 rootfs
root@eonSpider0x1:/mycontainer# ls -l rootfs/
total 48
drwxr-xr-x 2 root root 12288 Feb 14 19:58 bin
drwxr-xr-x 4 root root 4096 Mar 13 17:12 dev
drwxr-xr-x 3 root root 4096 Mar 13 17:12 etc
drwxr-xr-x 2 nobody nogroup 4096 Feb 14 19:58 home
drwxr-xr-x 2 root root 4096 Mar 13 17:12 proc
drwxr-xr-x 2 root root 4096 Feb 14 19:58 root
drwxr-xr-x 2 root root 4096 Mar 13 17:12 sys
drwxrwxrwt 2 root root 4096 Feb 14 19:58 tmp
drwxr-xr-x 3 root root 4096 Feb 14 19:58 usr
drwxr-xr-x 4 root root 4096 Feb 14 19:58 var
root@eonSpider0x1:/mycontainer# I
```

Команда `runc spec` изначально создает этот файл JSON:

Альтернативой для создания кастомной спецификации конфигурации является использование «`oci-runtime-tool`», подкоманда «`oci-runtime-tool generate`» имеет множество опций, которые можно использовать для выполнения разных настроек.

Для получения дополнительной информации см. [Runtime-tools](#).

Используя сгенерированный файл спецификации JSON, вы можете настроить время работы контейнера. Мы можем, например, изменить аргумент для выполнения приложения.

```

32 // Process contains information to start a specific application inside the container.
33 type Process struct {
34     // Terminal creates an interactive terminal for the container.
35     Terminal bool `json:"terminal,omitempty"`
36     // ConsoleSize specifies the size of the console.
37     ConsoleSize *Box `json:"consoleSize,omitempty"`
38     // User specifies user information for the process.
39     User User `json:"user"`
40     // Args specifies the binary and arguments for the application to execute.
41     Args []string `json:"args,omitempty"`
42     // CommandLine specifies the full command line for the application to execute on Windows.
43     CommandLine string `json:"commandLine,omitempty" platform:"windows"`
44     // Env populates the process environment for the process.
45     Env []string `json:"env,omitempty"`
46     // Cwd is the current working directory for the process and must be
47     // relative to the container's root.
48     Cwd string `json:"cwd"`
49     // Capabilities are Linux capabilities that are kept for the process.
50     Capabilities *LinuxCapabilities `json:"capabilities,omitempty" platform:"linux"`
51     // Rlimits specifies rlimit options to apply to the process.
52     Rlimits []POSIXRlimit `json:"rlimits,omitempty" platform:"linux,solaris"`
53     // NoNewPrivileges controls whether additional privileges could be gained by processes in the container.
54     NoNewPrivileges bool `json:"noNewPrivileges,omitempty" platform:"linux"`
55     // ApparmorProfile specifies the apparmor profile for the container.
56     ApparmorProfile string `json:"apparmorProfile,omitempty" platform:"linux"`
57     // Specify an oom_score_adj for the container.
58     OOMScoreAdj *int `json:"oomScoreAdj,omitempty" platform:"linux"`
59     // SelinuxLabel specifies the selinux context that the container process is run as.
60     SelinuxLabel string `json:"selinuxLabel,omitempty" platform:"linux"`
61 }
62

```

Давайте посмотрим, чем отличается исходный файл `config.json` от нового:

```

{
  "ociVersion": "1.0.0-rc2-dev",
  "platform": {
    "os": "linux",
    "arch": "amd64"
  },
  "process": {
    "terminal": false,
    "consolesize": {
      "height": 0,
      "width": 0
    },
    "user": {
      "uid": 0,
      "gid": 0
    },
    "args": [
      "sleep", "10"
    ],
    "env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "TERM=xterm"
    ],
    "cwd": "/",
    "capabilities": [
      "CAP_AUDIT_WRITE",
      "CAP_KILL",
      "CAP_NET_BIND_SERVICE"
    ],
    "rlimits": [
      {
        "type": "RLIMIT_NOFILE",
        "hard": 1024,
        "soft": 1024
      }
    ],
    "noNewPrivileges": true
  },
  "root": {
    "path": "rootfs",
    "readonly": true
  }
}
-- INSERT --
{
  "ociVersion": "1.0.0-rc2-dev",
  "platform": {
    "os": "linux",
    "arch": "amd64"
  },
  "process": {
    "terminal": true,
    "consolesize": {
      "height": 0,
      "width": 0
    },
    "user": {
      "uid": 0,
      "gid": 0
    },
    "args": [
      "sh"
    ],
    "env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "TERM=xterm"
    ],
    "cwd": "/",
    "capabilities": [
      "CAP_AUDIT_WRITE",
      "CAP_KILL",
      "CAP_NET_BIND_SERVICE"
    ],
    "rlimits": [
      {
        "type": "RLIMIT_NOFILE",
        "hard": 1024,
        "soft": 1024
      }
    ],
    "noNewPrivileges": true
  },
  "root": {
    "path": "rootfs",
    "readonly": true
  }
}

```

18,16-37 Top 5,1-8 Top

Давайте теперь снова запустим контейнер и заметим, как он ожидает 10 секунд, прежде чем завершится.

Стандарты сред исполнения контейнеров

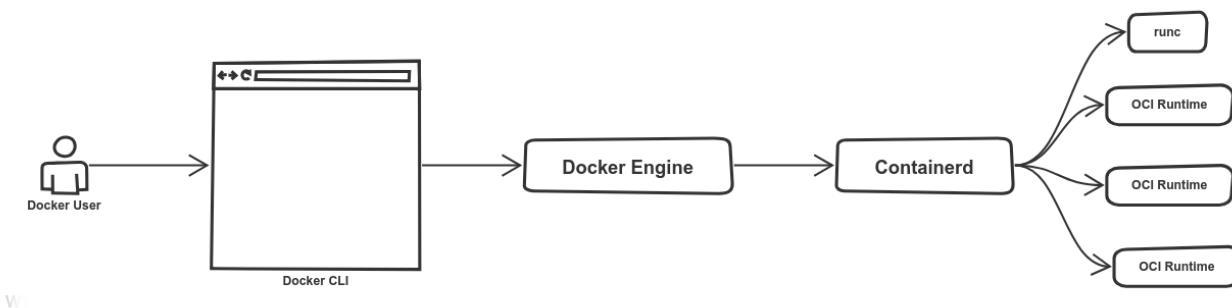
С тех пор, как контейнеры стали широко распространеными, различные участники этой экосистемы работали над стандартизацией. Стандартизация - ключ к автоматизации и обобщению передового опыта. Передав проект гипС OCI, Docker начал использовать containerd в 2016 году в качестве среды выполнения контейнера, взаимодействующей с базовой средой исполнения низкого уровня гипС.

```
docker info | grep -i runtime
```

```
eon01@eonSpider0x1:~$ docker info | grep -i runtime
```

Containerd полностью поддерживает запуск пакетов OCI и управление их жизненным циклом. Containerd (как и другие среды выполнения, такие как cgroups) использует гипС

для запуска контейнеров, но реализует также другие высокоуровневые функции, такие как управление образами и высокоуровневые API.



Интеграция containerd со средами выполнения Docker и OCI

Containerd, Shim и RunC, как все работает вместе

RunC построен на libcontainer, который является той же библиотекой, которая ранее использовалась для Docker Engine.

До версии 1.11 Docker Engine использовался для управления томами, сетями, контейнерами, образами и т. д.

Теперь архитектура Docker разбита на четыре компонента:

- Docker engine
- containerd
- containerd-shim
- runC

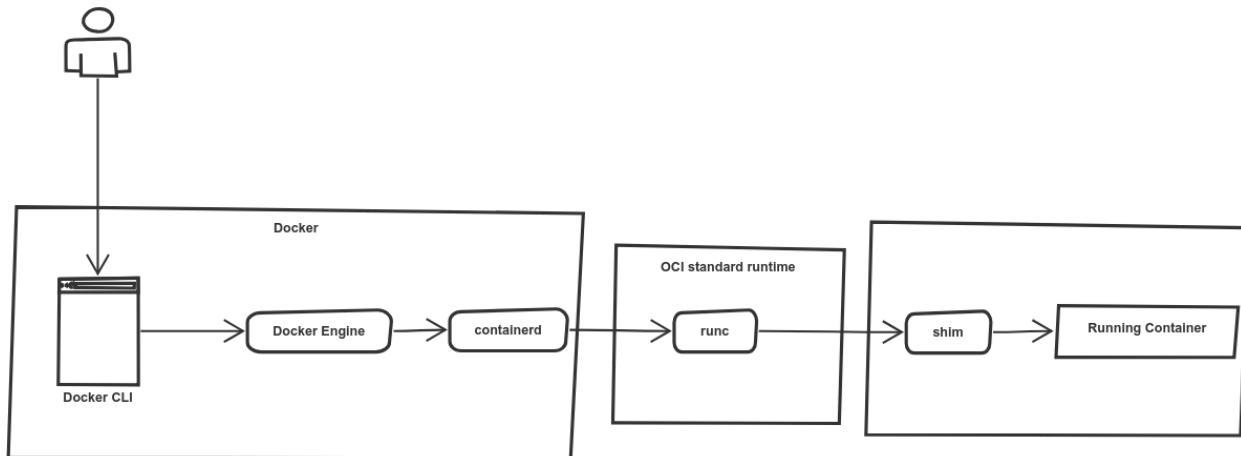
Бинарные файлы соответственно называются docker, docker-containerd, docker-containerd-shim и docker-runC.

Давайте перечислим этапы запуска контейнера с использованием новой архитектуры docker:

1. Docker engine создает контейнер (из образа) и передает его в containerd.
2. Containerd вызывает containerd-shim
3. Containerd-shim использует runC для запуска контейнера
4. Containerd-shim позволяет среде выполнения (в данном случае runC) завершиться после запуска контейнера

Используя эту новую архитектуру, мы можем запускать «контейнеры без служб» (“daemonless containers”), и у нас есть два преимущества:

1. гипС может завершиться после запуска контейнера, и нам не нужны запущенными все процессы исполнения.
2. containerd-shim сохраняет открытыми файловые дескрипторы, такие как `stdin`, `stdout` и `stderr`, даже когда Docker и /или containerd завершаются.



«Если runC и Containerd являются средами исполнения, какого черта мы используем оба для запуска одного контейнера?»

Это, наверное, один из самых частых вопросов. Поняв, почему Docker разбил свою архитектуру на гипС и Containerd, вы понимаете, что оба являются средами исполнения.

Если вы следили за историей с самого начала, вы, вероятно, заметили использование сред исполнения высокого и низкого уровня. В этом практическая разница между ними.

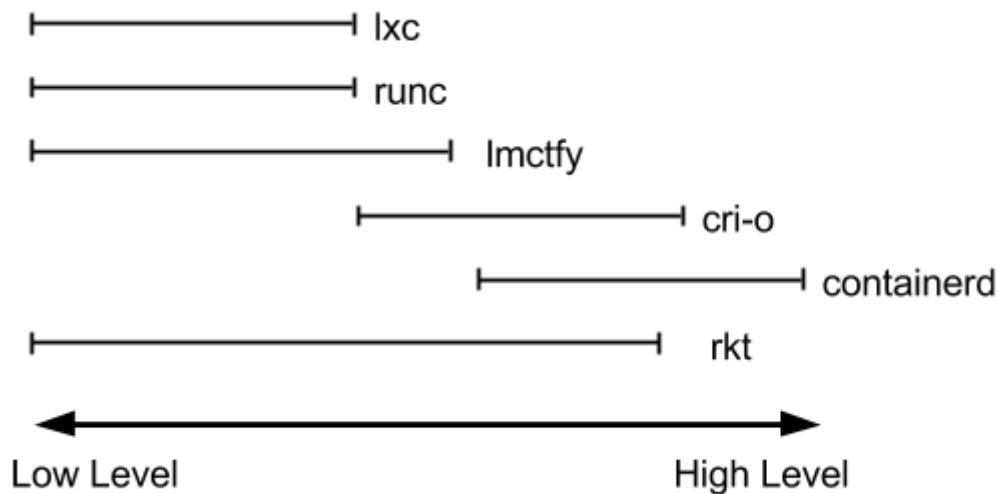
Обе они могут называться средами исполнения, но каждая среда исполнения имеет разные цели и функции. Чтобы сохранить стандартизацию экосистемы контейнеров, среда исполнения низкоуровневых контейнеров позволяет запускать только контейнеры.

Среда исполнения низкого уровня (например, гипС) должна быть легкой, быстрой и не конфликтовать с другими более высокими уровнями управления контейнерами. Когда вы создаете контейнер Docker, он фактически управляет двумя средами исполнения `containerd` и `гипС`.

Вы можете найти множество сред исполнения контейнеров, некоторые из них стандартизованы OCI, а другие нет, некоторые являются средами исполнения низкого

уровня, а другие представляют собой нечто большее и реализуют уровень инструментов для управления жизненным циклом контейнеров и многое другое:

- передача и хранение образов,
 - завершение и наблюдение за контейнерами,
 - низкоуровневое хранилище,
 - сетевые настройки,
 - и т.п.



Мы можем добавить новую среду исполнения с помощью Docker, выполнив:

```
sudo dockerd --add-runtime=<runtime-name>=<runtime-path>
```

Например:

```
sudo apt-get install nvidia-container-runtime  
sudo dockerd --add-runtime=nvidia=/usr/bin/nvidia-container-runtime
```

Интерфейс среды исполнения контейнера (Container Runtime Interface)

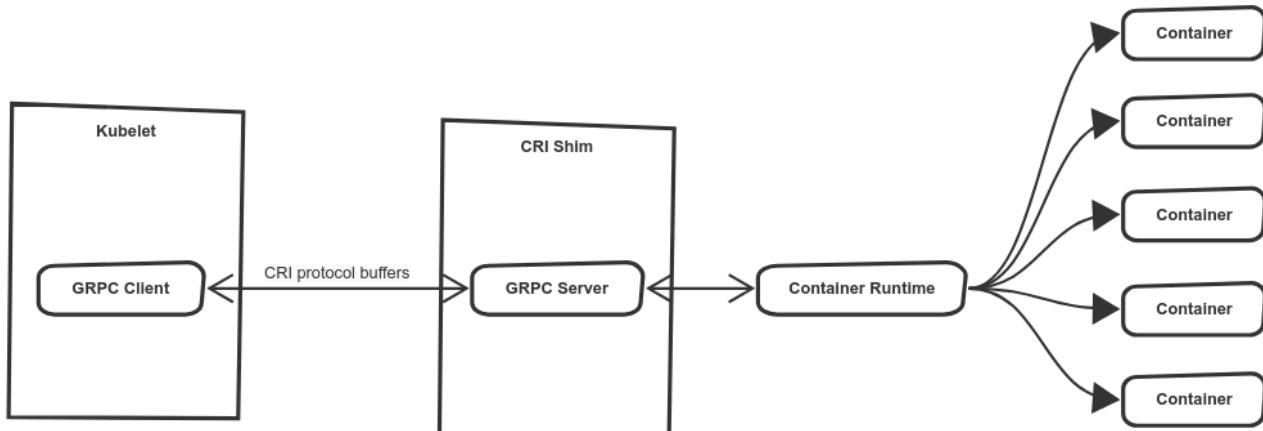
Kubernetes - одна из самых популярных систем оркестровки. С ростом числа сред выполнения контейнеров Kubernetes стремится быть более расширяемым и

взаимодействовать с большим количеством сред выполнения контейнеров, помимо Docker.

Первоначально Kubernetes использовал среду исполнения Docker для запуска контейнеров, и она по-прежнему остается средой исполнения по умолчанию.

Однако CoreOS хотела использовать Kubernetes со средой исполнения RKT и предлагала патчи для Kubernetes, чтобы использовать эту среду исполнения в качестве альтернативы Docker.

Вместо изменения кодовой базы Kubernetes в случае добавления новой среды исполнения контейнера создатели Kubernetes решили создать CRI (Container Runtime Interface), который представляет собой набор API-интерфейсов и библиотек, позволяющих запускать различные среды исполнения контейнеров в Kubernetes. Любое взаимодействие между ядром Kubernetes и поддерживаемой средой выполнения осуществляется через CRI API.



Вот некоторые из плагинов CRI:

CRI-O:

CRI-O - это первая среда исполнения контейнера, созданная для интерфейса CRI Kubernetes. CRI-O не предназначен для замены Docker, но его можно использовать вместо среды исполнения Docker в Kubernetes.



CRI-O: OCI-based Kubernetes Runtime

Containerd CRI :

С cri-containerd пользователи могут запускать кластеры Kubernetes, используя containerd в качестве базовой среды исполнения без установленного Docker.



gVisor CRI:

gVisor - это проект, разработанный Google, который реализует около 200 системных вызовов Linux в пользовательском пространстве для дополнительной безопасности по сравнению с контейнерами Docker, которые работают непосредственно поверх ядра Linux и изолированы с помощью namespaces.



Google Cloud

gVisor

Google Cloud App Engine использует gVisor CRI для изоляции клиентов.

Среда исполнения gVisor интегрируется с Docker и Kubernetes, что упрощает запуск изолированных контейнеров.

CRI-O Kata Containers

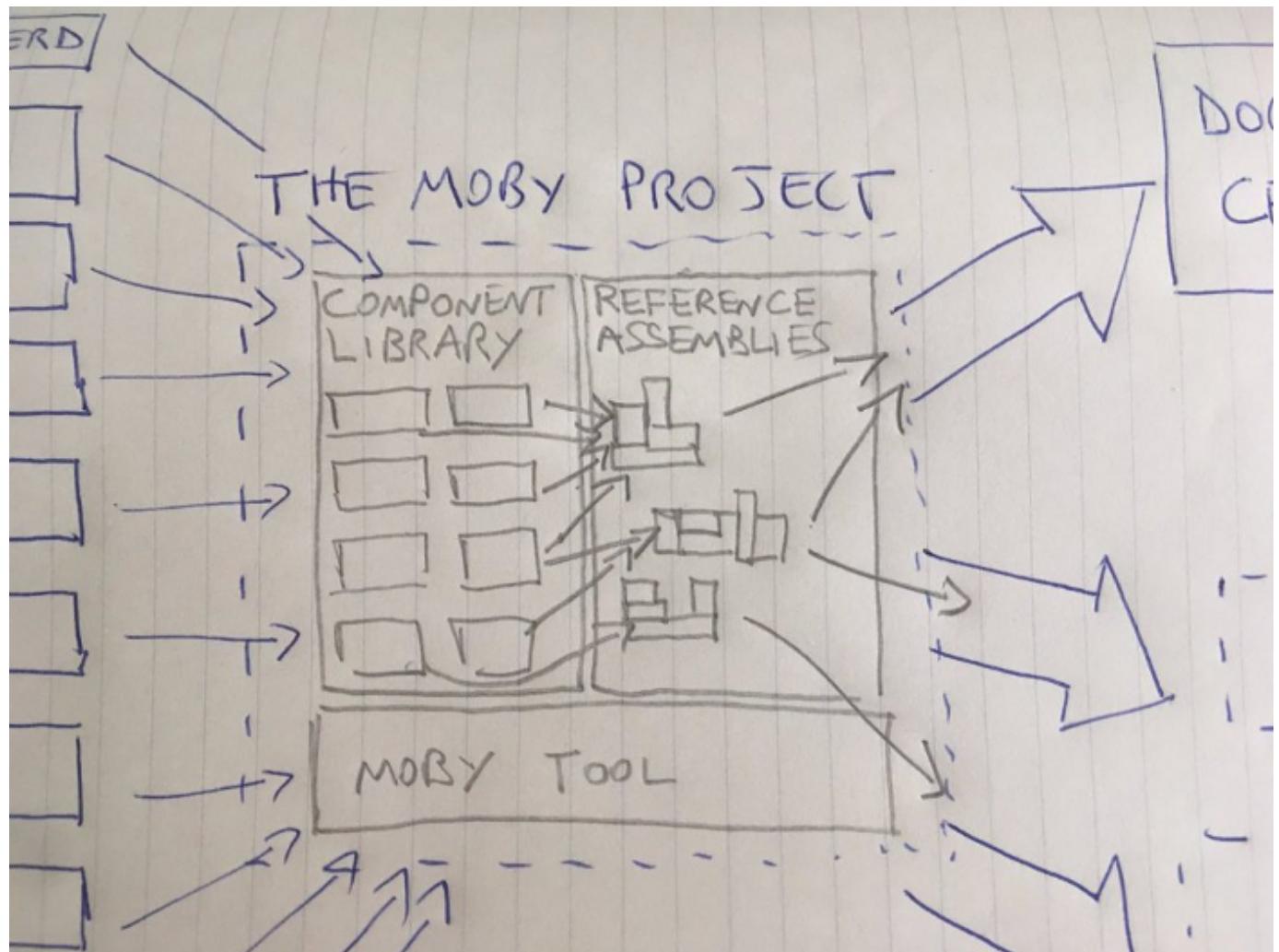


Kata Containers - это проект с открытым исходным кодом, создающий легкие виртуальные машины, которые подключаются к экосистеме контейнеров. CRI-O Kata Containers

позволяет запускать контейнеры Kata в Kubernetes вместо среды выполнения Docker по умолчанию.

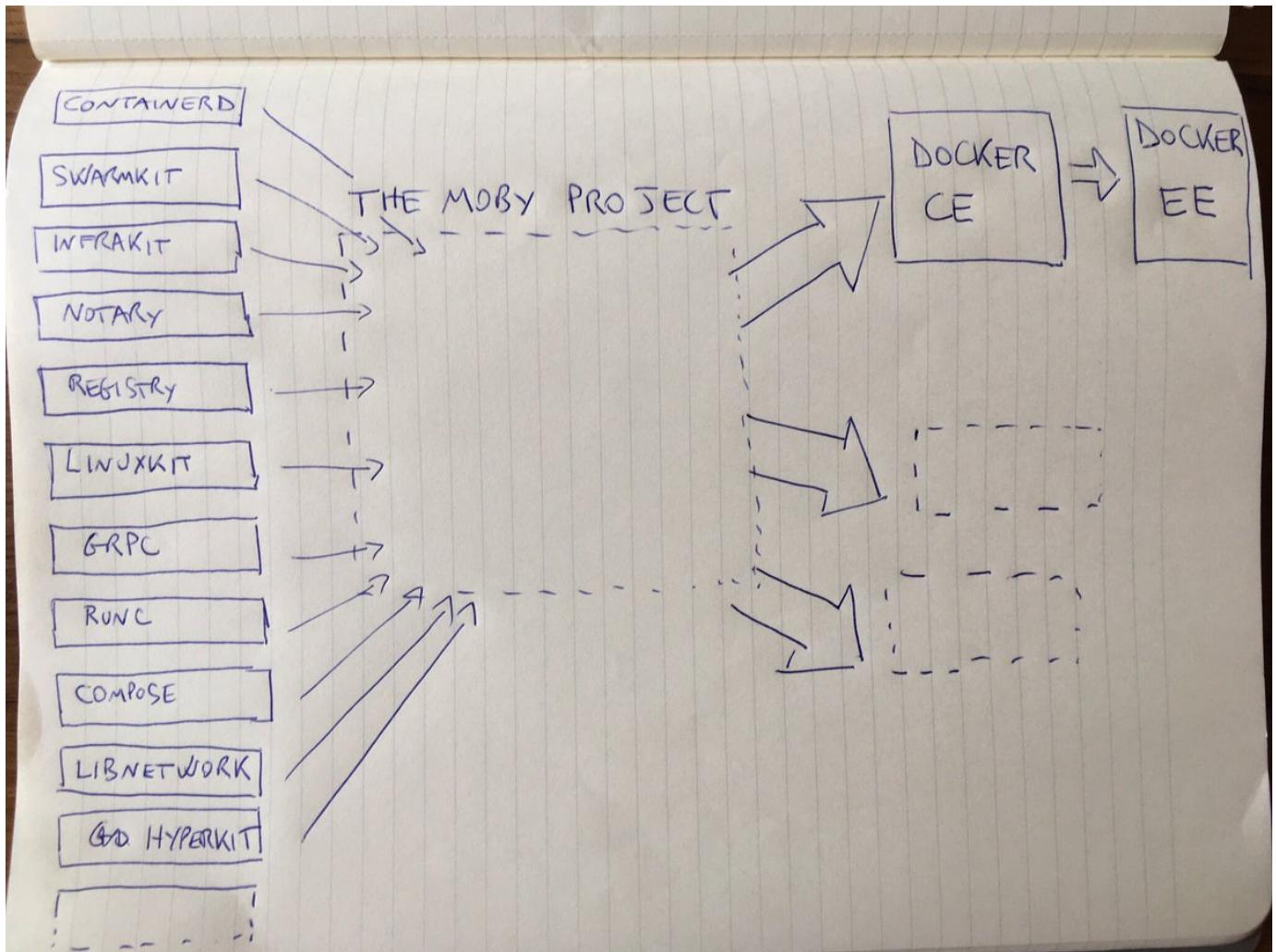
Проект Moby

От проекта создания Docker как единой монолитной платформы отказались и родился проект Moby, в котором Docker состоит из множества компонентов, таких как RunC.



Источник: Solomon Hykes Twitter

Moby - это проект по организации и разделения на модули Docker. Это экосистема разработки. Обычные пользователи Docker не заметят никаких изменений.



Источник: Solomon Hykes Twitter

Moby помогает в разработке и запуске Docker CE и EE (Moby - это исходный код Docker), а также в создании среды разработки для других сред исполнения и платформ.

Open Containers Initiative

Как мы видели, Docker пожертвовал RunC Open Container Initiative (OCI), но что это?

OCI - это открытая структура, запущенная в 2015 году Docker, CoreOS и другими лидерами контейнерной индустрии.

Open Container Initiative (OCI) направлена на установление общих стандартов для контейнеров, чтобы избежать потенциальной фрагментации и разделения внутри экосистемы контейнеров.



OPEN CONTAINER INITIATIVE

Он содержит две спецификации:

- `runtime-spec`: спецификация исполнения
- `image-spec`: спецификация образов

Контейнер, использующий другую среду исполнения, можно использовать с Docker API.
Контейнер, созданный с помощью Docker, должен работать с любым другим движком.

На этом статья заканчивается.

Буду рад замечаниям и возможно неточностям в статье оригинального автора. Это позволит избежать заблуждений в понимании внутреннего устройства контейнеров. Если нет возможности комментирования на Хабре, можете обсудить тут в комментариях.

Теги: контейнеры, docker, gupc, containerd, kubernetes

Хабы: Системное администрирование, Виртуализация, DevOps, Kubernetes

Редакторский дайджест

Присыпаем лучшие статьи раз в месяц

×

Электропочта



13

0

Карма Рейтинг

Ильдар @il_da_g

Системный администратор/DevOps-инженер

Github Telegram

Реклама



Ищите работников
для своего бизнеса на hh.ru

Реклама. ООО «Хэдхантер». Подробнее на сайте hh.ru

Найти

0+

Комментарии 2

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ



anton-argemenko

вчера в 14:01

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

Простой

13 мин

12K

Обзор

+49

55

48

21 час назад

Герои напильника и паяльника: итоги сезона DIY

8 мин

8.1K

Сезон DIY

Спецпроект

+41

42

6

spiritus_sancti

22 часа назад

RGB-усилители. Особенности, проблемы, выбор

Простой

6 мин

5.4K

Туториал

+39

24

11



ru_vds

18 часов назад

Профилирование Python — почему и где тормозит ваш код

Средний

10 мин

3.2K

Туториал

Перевод

+31

90

3

**BiktorSergeev**

15 часов назад

WebOne: даём жизнь старым браузерам

6 мин

3.1K

+27

26

7

**AnnaVlMogozova**

18 часов назад

Как повысить эффективность коммуникаций в команде: учимся решать конфликты

7 мин

2.2K

+19

43

2

**Yu-Leo**

23 часа назад

Обзор электронной книги Meebook P10 Pro

Простой

9 мин

6.1K

Обзор

+18

18

8

**it_union**

1 час назад

ЗАО Гейм Инсайт Труп

Простой

4 мин

2.4K

+17

6

5

**fedorborovitsky**

вчера в 14:13

Российский софт в идеальном штурме

6 мин

7.6K

[Мнение](#)[+15](#)[16](#)[38](#)

zatim

52 минуты назад

Видеокарта VGA для микроконтроллера

[Средний](#) [17 мин](#) [500](#)[Туториал](#)[+13](#)[6](#)[3](#)

Приманка для экспертов в Нетологии: опыт преподавания, рост в софтах, лекции по 4 часа

[Спецпроект](#)[Показать еще](#)

МИНУТОЧКУ ВНИМАНИЯ

[Турбо](#)[Спецпроект](#)

Вышел каменный цветок? Поможем рассказать

Проверь свою SQL-экипировку в teste для аналитиков

ЗАКАЗЫ

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

100000 руб./за проект • 62 отклика • 233 просмотра

Серверное ПО, Photoshop на виндовом сервере

500000 руб./за проект · 28 откликов · 115 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

Консультация (аудит) о подготовке сайта к высоким нагрузкам

2500 руб./в час · 3 отклика · 54 просмотра

Больше заказов на Хабр Фриланс

Реклама

 tinkoff.ru РЕКЛАМА

Присоединяйтесь к Тинькофф

 **ТИНЬКОФФ**

Реклама | 0+



**Найдите вакансию
на tinkoff.ru**

АО «Тинькофф Банк» (Банк) вошел в рейтинг «Лучшие работодатели России — 2022» по версии журнала Forbes (Форбс) среди 118 компаний по совокупным результатам исследования по критериям: экология, сотрудники и общество, корпоративное управление. Реклама. Рекламодатель: АО Тинькофф Банк, лицензия №2673

Подробнее

ЧИТАЮТ СЕЙЧАС

ЗАО Гейм Инсайт Труп

 2.4K  5

Теневое правление Илона Маска

7.3K 29

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

12K 48

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

41K 117

Всего два месяца — и новый релиз ядра Linux. Что появилось в ядре 6.5, что изменилось и что удалили. Новые возможности

4.6K 4

Приманка для экспертов в Нетологии: опыт преподавания, рост в софтах, лекции по 4 часа

Спецпроект

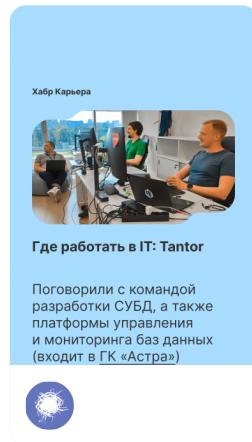
ИСТОРИИ



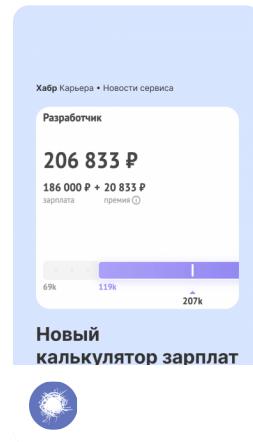
Воспитай айтишника
Подборка статей о том, как научить детей программированию



Ошибки разума
Статьи о когнитивных искажениях и их проявлениях в жизни

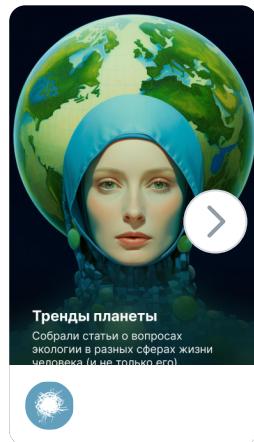


Где работать в IT: Tantor
Поговорили с командой разработки СУБД, а также платформы управления и мониторинга баз данных (входит в ГК «Астра»)



Разработчик
206 833 ₽
186 000 ₽ + 20 833 ₽
зарплата премии
69k 119k 207k

Новый калькулятор зарплат



Тренды планеты
Собрали статьи о вопросах экологии в разных сферах жизни человека (и не только его)

Как учить детей
программированию

Когнитивные
искажения

Где работать в IT:
Tantor

Калькулятор
зарплат 2.0

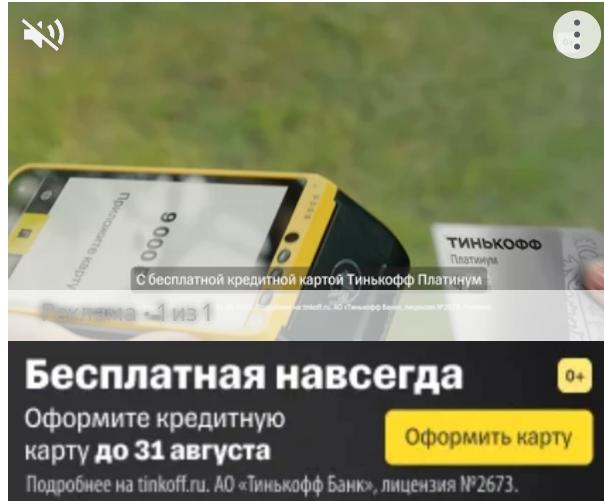
«Зелёная» подборка

РАБОТА

Системный администратор
101 вакансия

DevOps инженер

50 вакансий

[Все вакансии](#)[Реклама](#)**Ваш аккаунт**[Войти](#)[Регистрация](#)**Разделы**[Статьи](#)[Новости](#)[Хабы](#)[Компании](#)[Авторы](#)[Песочница](#)**Информация**[Устройство сайта](#)[Для авторов](#)[Для компаний](#)[Документы](#)[Соглашение](#)[Конфиденциальность](#)**Услуги**[Корпоративный блог](#)[Медийная реклама](#)[Нативные проекты](#)[Образовательные программы](#)[Стартапам](#)[Специпроекты](#)[Настройка языка](#)[Техническая поддержка](#)[Вернуться на старую версию](#)

