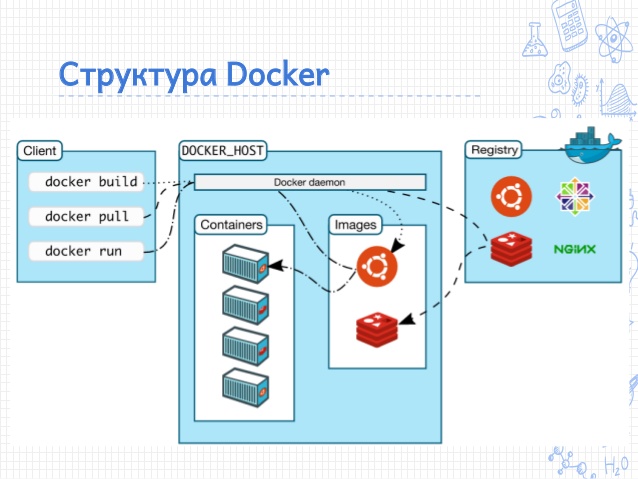
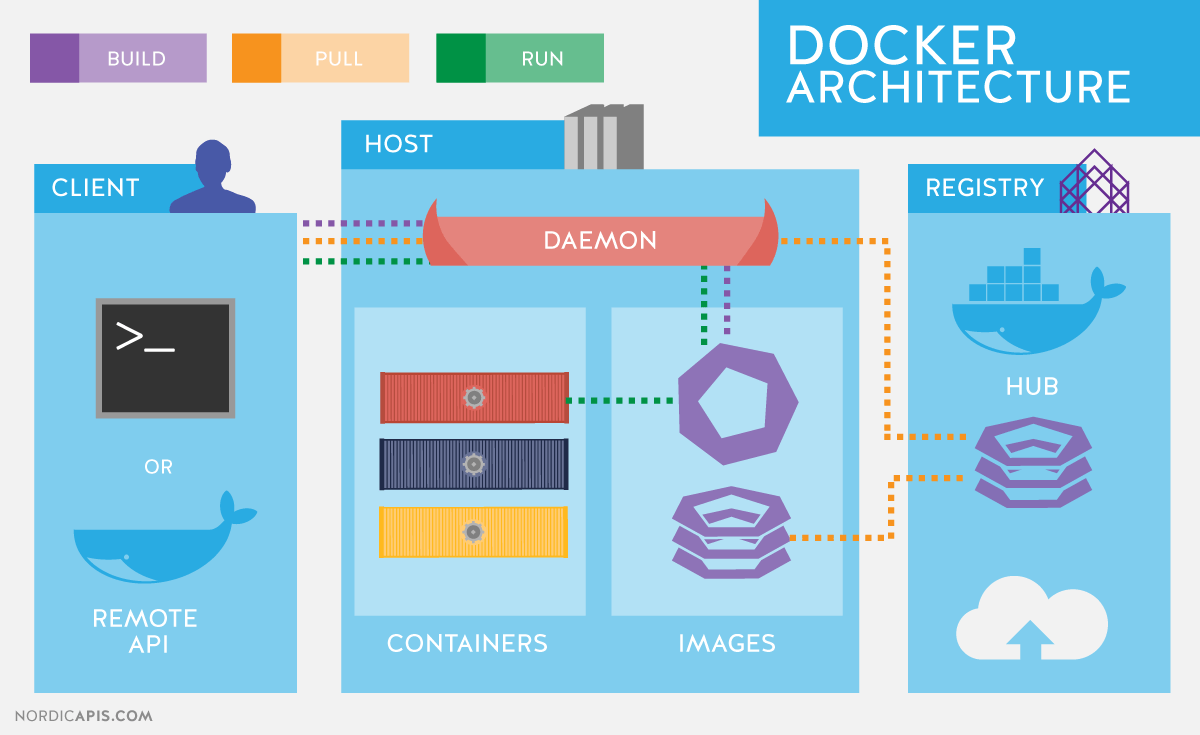
Ми не раз торкалися тематики і розглядали безліч систем для їх побудови. Сьогодні ми познайомимо ще із однією чудовою системою контейнерами Docker.

Почнемо з того, що опишемо базовий функціонал, який стане в нагоді в подальших статтях циклу, і коротко нагадаємо про архітектуру Docker. Docker використовує клієнт-серверну архітектуру та складається з клієнта – утиліти docker, яка звертається до сервера за допомогою **RESTful API**, і демона в операційній [системі Linux](https://sukachoff.ru/uk/noutbuki/poluchaem-informaciyu-ob-oborudovanii-v-linux-nadezhnyi-pico-itx-sbc/)(Див. рис. 1). 

##### **Основні компоненти Docker:**

* **Контейнери**– ізольовані за допомогою технологій операційної системи користувацькі оточення, в яких виконуються програми. Найпростіше дати визначення контейнеру Docker як запущеному з образу додатку. До речі, саме цим ідеологічно і відрізняється Docker, наприклад, від LXC ( **Linux Containers**), хоча вони використовують одні й ті самі технології [ядра Linux](https://sukachoff.ru/uk/vosstanovlenie/upravlenie-keshem-cache-v-yadre-linux-2-4-kak-proverit-ispolzovanie/). Розробники проекту Docker сповідує принцип: один контейнер – це одна програма.
* **Образи**– доступні лише для читання шаблони програм. Поверх існуючих образів можуть додаватися нові рівні, які спільно становлять файлову систему, змінюючи або доповнюючи попередній рівень. Зазвичай новий образ створюється або за допомогою збереження вже запущеного контейнера новий образ поверх існуючого, або за допомогою спеціальних інструкцій для утиліти . Для розділення різних рівнів контейнера на рівні файлової системи можна використовувати **AUFS, btrfs, vfs та Device Mapper**. Якщо передбачається використання Docker спільно з **SELinux**, то потрібно **Device Mapper.**
* **Реєстри (registry)**, що містять репозиторії ( **repository**) образів, - [мережеві сховища](https://sukachoff.ru/uk/windows/primenenie-iscsi-v-sistemah-hraneniya-dannyh-setevoe-hranilishche-na-iscsi-chto-eto/)образів. Можуть бути як приватними, і загальнодоступними. Найвідомішим реєстром є .



##### **Для ізоляції контейнерів в операційних системах GNU/Linux використовуються стандартні технології ядра Linux, такі як:**

* Простір імен ( **Linux Namespaces**).
* Контрольні групи ( **Cgroups**).
* Засоби управління привілеями ( **Linux Capabilities**).
* Додаткові, мандатні системи безпеки, такі як **AppArmor або SELinux.**

### Розглянемо перелічені технології трохи докладніше.

Механізм контрольних груп **(Cgroups)**надає інструмент для тонкого контролю за розподілом, пріоритизацією та управлінням системними ресурсами. Контрольні групи реалізовані у ядрі Linux. У сучасних дистрибутивах управління контрольними групами реалізовано через **systemd**, однак зберігається можливість керування за допомогою бібліотеки **libcgroup**та утиліти **cgconfig**. Основні ієрархії контрольних груп (їх також називають контролерами) наведені нижче:

* **blkio**- задає ліміти на операції введення-виводу та на доступ до блокових пристроїв;
* **cpu**- Використовуючи планувальник процесів, розподіляє процесорний час між завданнями;
* **cpuacct**– створює автоматичні звіти щодо використання ресурсів [центрального процесора](https://sukachoff.ru/uk/remont/ustanovka-noveishei-versii-mikrokoda-centralnogo/). Працює спільно з контролером **cpu**описаним вище;
* **cpuset**– закріплює за завданнями певні процесори та вузли пам'яті;
* **devices**– регулює доступ завдань до певних пристроїв;
* **freezer**- Зупиняє або відновлює завдання;
* **memory**– встановлює ліміти та генерує звіти про використання пам'яті завданнями контрольної групи;
* **net\_cls**- Здійснює тегування [мережеві пакети](https://sukachoff.ru/uk/windows/kak-ispolzovat-vozmozhnosti-filtrov-otobrazheniya-wireshark-po/)ідентифікатором класу ( **classid**). Це дозволяє контролеру трафіку ( **команда tc**) та брандмауеру ( **iptables**) враховувати ці теги під час обробки трафіку;
* **perf\_event**– дозволяє проводити моніторинг контрольних груп за допомогою утиліти **perf;**
* **hugetlb**– дозволяє використовувати віртуальні сторінки пам'яті великого розміру та застосовувати до них ліміти.

**Простір імен,**своєю чергою, контролюють не розподіл ресурсів, а доступом до структур даних ядра. Фактично це означає ізоляцію процесів один від одного і можливість мати паралельно «однакові», але не ієрархії процесів, користувачів і мережевих інтерфейсів, що не перетинаються один з одним. За бажання [різні сервіси](https://sukachoff.ru/uk/internet/kak-sbrosit-telefon-na-zavodskie-nastroiki-samsung-kak/)можуть мати навіть свої власні **loopback-інтерфейси**.

##### **Приклади просторів імен, які використовуються Docker:**

* **PID, Process ID**- Ізоляція ієрархії процесів.
* **NET, Networking**- Ізоляція мережевих інтерфейсів.
* **PC, InterProcess Communication**- Управління взаємодією між процесами.
* **MNT, Mount**- Управління точками монтування.
* **UTS, Unix Timesharing System**– ізоляція ядра та ідентифікаторів версії.

Механізм під назвою **Capabilities** дозволяє розбити привілеї користувача root на невеликі групи привілеїв та призначати їх окремо. Цей функціонал у GNU/Linux з'явився починаючи з версії **ядра 2.2.** Спочатку контейнери запускаються вже з обмеженим набором привілеїв.

##### **За допомогою опцій команди docker можете дозволяти та забороняти:**

* операції монтування;
* доступ до сокетів;
* виконання частини операцій із файловою системою, наприклад зміна атрибутів файлів чи власника.

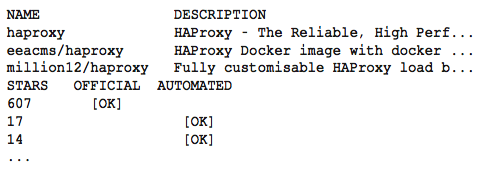
Детальніше ознайомитися з привілеями можна за допомогою man-сторінки **CAPABILITIES(7)**.

### Пошук образів та теги Docker

Спробуймо знайти контейнер на Docker Hub.

$ docker search haproxy

|  |  |
| --- | --- |
|  | $search haproxy |



Тут ми отримали список різних образів HA Proxy. Найвищий елемент списку – це HA Proxy із офіційного репозиторію. Такі образи відрізняються тим, що в імені немає символу **«/»** , що відокремлює ім'я репозиторія користувача від імені контейнера. У прикладі за офіційним показано два образи haproxy з відкритих репозиторіїв користувачів **eeacms та million12.**

Образи, подібні до двох нижніх, можете створити самі, зареєструвавшись на Docker Hub. Офіційні ж підтримуються спеціальною командою, яка спонсорується Docker, Inc. Особливості офіційного репозиторію:

Число зірок у виведенні команди **docker search** відповідає популярності образу. Це аналог кнопки **Like** в соцмережах або закладок для інших користувачів. **Automated** означає, що образ збирається автоматично із спеціального сценарію засобами Docker Hub. Зазвичай слід віддавати перевагу образам, що автоматично збираються внаслідок того, що його вміст може бути перевірено.

Завантажуємо офіційний образ HA Proxy:

$ docker pull haproxy

Використання детального tag: latest

Повне ім'я образу може виглядати так:

**[ім'я користувача]ім'я образу[:тег]**

Переглянути список завантажених образів можна командою

**docker images:**

### Запуск контейнерів

Для запуску контейнера не обов'язково заздалегідь завантажувати образ. Якщо він доступний, то буде завантажено автоматично. Давайте спробуємо запустити контейнер з Ubuntu. Ми не будемо вказувати репозиторій, і буде завантажено останній офіційний образ, підтримуваний Canonical.

$ docker run -it ubuntu root@d7402d1f7c54:/#

|  |  |
| --- | --- |
|  | $ run - it ubuntu  root @ d7402d1f7c54 : / # |

Окрім команди **run**, ми вказали дві опції: **-i**– контейнер повинен запуститися в інтерактивному режимі та **-t**– має бути виділено псевдотермінал. Як видно з висновку, у контейнері ми маємо привілеї користувача root, а як ім'я вузла відображається ідентифікатор контейнера. Останнє може бути справедливим не для всіх контейнерів і залежить від розробника контейнера. Перевіримо, що це дійсно оточення Ubuntu:

*root@d7402d1f7c54:/# cat /etc/\*release | grep* DISTRIB\_DESCRIPTION DISTRIB\_DESCRIPTION="Ubuntu 16.04.1 LTS"

|  |  |
| --- | --- |
|  | root @ d7402d1f7c54 : / # cat /etc/\*release | grep DISTRIB\_DESCRIPTION  DISTRIB\_DESCRIPTION = "Ubuntu 16.04.1 LTS" |

Команду *uname* ***-a*** для таких цілей використовувати не вийде, оскільки контейнер працює з ядром хоста.

Як одна з опцій можна було б задати унікальне ім'я контейнера, на яке можна для зручності посилатися, крім **ID-контейнер.** Вона задається як **-name<ім’я>.** Якщо опція опущена, ім'я генерується автоматично.

Імена контейнерів, що автоматично генеруються, не несуть смислового навантаження, проте як цікавий факт можна відзначити, що імена генеруються випадковим чином з прикметника та імені відомого вченого, винахідника або хакера. У коді генератора для кожного імені можна знайти короткий опис того, чим відомий цей діяч.

Переглянути список запущених контейнерів можна командою

docker ps

Однак якщо віддати команду контейнера, створеного з образу mysql, ми не виявимо. Скористаємося опцією **-a** яка показує всі контейнери, а не тільки запущені:

Очевидно, що під час запуску контейнера не було вказано обов'язкових параметрів. Ознайомитися з описом змінних середовища, необхідних для запуску контейнера, можна знайти офіційний образ MySQL на Docker Hub. Повторимо спробу, використовуючи опцію **-e**, яка задає змінні оточення у контейнері:

$ docker run --name mysql-test -e MYSQL\_ROOT\_PASSWORD=docker -d mysql

Останнім параметром є команда, яку ми хочемо виконати всередині контейнера. У цьому випадку це командний інтерпретатор **Bash**. Опції **-it**аналогічні за призначенням використаним раніше у команді **docker run.**

Фактично після запуску цієї команди у контейнер **mysql-test** додається ще один процес – **bash**. Це можна побачити за допомогою команди pstree.