

Распространение образов

После создания собственных образов у нас возникает естественное желание сделать их доступными для всех, для серверов непрерывной интеграции ПО и для конечных пользователей. Существует несколько способов распространения образов: можно заново создавать их из соответствующих файлов Dockerfile, скачивать из реестров или использовать команду `dockerload` для установки образа из архивного файла.

Самое простое и очевидное решение, обеспечивающее общий доступ к разработанным образам, – использование Docker Hub. Docker Hub – реестр, работающий в режиме онлайн и поддерживаемый компанией Docker Inc. Этот реестр предоставляет свободные репозитории для открытых образов, но пользователи могут оплатить закрытые, защищенные репозитории.

Docker Hub – это не единственный вариант при выборе хранилища частных закрытых репозиторий в облачной среде. В 2020 году основным конкурентом является реестр `quay.io`, предлагающий чуть более расширенные функциональные возможности, по сравнению с Docker Hub, по вполне приемлемым ценам.

Образ можно выгрузить в репозиторий без особых трудностей. Предположим, что у вас уже есть учетная запись в реестре Docker Hub, тогда эта операция выполняется прямо из командной строки:

```
$ docker push <hub-user>/<repo-name>:<tag>
```

В первую очередь необходимо создать псевдоним (алиас) для образа в пространстве имен Docker Hub. Требуемая обязательная форма `<username>/<repositoryname>`, где `<user-name>` – имя вашей учетной записи в реестре Docker Hub (в данном примере `amouat`), а `<repositoryname>` – имя репозитория, который должен существовать в реестре Hub. Кроме того, предоставляется возможность присваивания данному образу тега `0.1.0`.
Выгрузка образа с использованием только что созданного псевдонима. Если указанный репозиторий не существовал ранее, то он создается, и выполняется выгрузка образа с учетом заданного тега.

После успешного выполнения команды `Push` наш образ общедоступен, и любой пользователь может загрузить его командой `docker pull`. При переходе на сайт Docker Hub можно найти свой репозиторий по URL.

Если войти в учетную запись, то предоставляется возможность выполнения разнообразных задач администрирования, таких как создание описания для репозитория, включение других пользователей в группу сотрудников и настройка параметров методики разработки `webhook`. Если нужно обновить репозиторий, то достаточно повторить выполнение команд установки тега и выгрузки образа для требуемого образа. При использовании существующего тега предыдущий образ будет перезаписан.

Настроим функцию автоматической сборки в реестре Docker Hub для нашего образа. После настройки Hub будет заново создаваться образ и сохраняться его в соответствующем репозитории при каждом внесении изменений в исходный код. Для этого потребуется создать и настроить репозиторий исходного кода на GitHub. Можно просто выгрузить исходный код, который написан нами к настоящему моменту, или «клонировать»

официальный код, размещенный на странице книги в GitHub (<https://github.com/using-docker/image-dist/>).

Домашняя страница пользователя на сайте реестра Docker Hub

The screenshot shows the 'Repositories & Images' page on Docker Hub. At the top, there are filters for 'Show repositories from:' (All Registries), 'Show repositories with:' (All), and 'Filter repositories by name:'. Below these are buttons for 'Rescan', 'Allow', 'Disallow', 'Approve Base', 'Delete', 'Export', and 'Clear'. The main table lists repositories with columns: Repository, Security Issues, Runtime Profile, Disallowed, and Registry. The repositories listed are jboss/wildfly, mongo, and nginx. jboss/wildfly has a security bar with a red segment and 1/1 disallowed. mongo and nginx have green security bars and 0/1 disallowed.

Repository	Security Issues	Runtime Profile	Disallowed	Registry
> jboss/wildfly	<div><div></div></div>	- none -	1/1	Docker Hub
> mongo	<div><div></div></div>	Profiling Now	0/1	Docker Hub
> nginx	<div><div></div></div>	- none -	0/1	Docker Hub

Автоматические сборки конфигурируются не из командной строки, а через веб-интерфейс сайта Docker Hub. После входа в свою учетную запись на этом сайте в правом верхнем углу должно появиться спускающееся меню с заголовком Create (Создать). Нужно выбрать пункт Create Automated Build (Создать автоматическую сборку) и указать расположение репозитория с исходным кодом приложения identidock1. После выбора репозитория исходного кода выводится страница параметров конфигурации для автоматической сборки. По умолчанию имя создаваемого репозитория совпадает с именем репозитория исходного кода, но его следует изменить. Рекомендуется добавить краткое описание репозитория. В поле первого тега оставьте без изменений значение Branch, затем введите имя master, чтобы отслеживать исходный код из главной ветви. В поле Dockerfile Location (Расположение Dockerfile) введите /<имя образа>/Dockerfile. Поле самого последнего тега определяет имя, присваиваемое данному образу в реестре Docker Hub. Можно оставить тег latest или заменить его на более информативный, например auto. После установки всех параметров щелкните по кнопке Create (Создать). Выводится страница сборки для нового репозитория. Первую сборку можно запустить, щелкнув по кнопке Trigger a Build (Начать сборку). После завершения процедуры сборки можно скачать новую версию образа (если сборка завершилась успешно). Протестировать функцию автоматизированной сборки можно посредством внесения небольших изменений в исходный код. В нашем случае добавим файл README, который Docker Hub также будет использовать для вывода информации об этом репозитории. В каталоге образа создайте файл README.md с кратким описанием, например таким:

“Мой первый образ”

Сохраните файл и отправьте его в репозиторий исходного кода:

```
$ git add README.md
```

```
$ git commit -m "Added README"
```

```
$ git push
```

Описанный выше способ создания и распространения образов подходит не для всех проектов. Создаваемые образы открыты для всех, если вы не оплатили за-крытого репозитория, и вы находитесь в полной зависимости от работоспособности реестра Docker Hub – если сервер «падает», то вы лишаетесь возможности обновления своих образов, а пользователи не могут их загружать. Кроме того, возникает проблема производительности: при необходимости оперативно создавать и перемещать образы через программный канал (pipeline) вы вряд ли согласитесь с ростом накладных расходов при передаче файлов из Docker Hub и вынужденным интервалом ожидания в очереди на сборку. Для проектов с открытым исходным кодом и для относительно небольших проектов Docker Hub выполняет свою задачу превосходно. Но для более крупномасштабных и более серьезных проектов вам, вероятнее всего, потребуются другие решения.

Кроме использования реестра Docker Hub, существуют и другие возможности. Можно делать все вручную, выполняя операции экспорта и импорта образов или просто создавая новые версии образов из файлов Dockerfile на каждом Docker-хосте. Оба решения не являются оптимальными: создание образов из Dockerfile, повторяемое многократно, – процесс медленный, к тому же при этом могут возникать различия между образами на разных хостах. Процедуры экспортирования и импортирования образов в известной степени не надежны и являются источниками потенциальных ошибок. Остается одна возможность: использовать отдельный реестр, сопровождение которого обеспечивается самим автором или независимым провайдером.

Организация.собственного.реестра

Частный реестр Docker существенно отличается от официального реестра Docker Hub. Оба реестра поддерживают реализацию прикладного программного интерфейса, позволяющего пользователям выгружать, загружать и выполнять поиск образов, но Docker Hub – это удаленный сервис с закрытым исходным кодом, тогда как частный реестр представляет собой приложение с открытым исходным кодом, работающее на локальной системе. Кроме того, Docker Hub обеспечивает поддержку учетных записей пользователей, ведение статистики и веб-интерфейс, а в частном реестре Docker все эти функции отсутствуют.

Простейшим способом создания локального реестра является использование официального образа. Быстрый запуск реестра осуществляется командой:

```
$ docker run -d -p 8080:8080 registry:2
```

Теперь у нас есть работающий реестр, и мы можем присваивать образам соответствующие теги и выгружать их в этот реестр. При использовании механизма docker-machine остается возможность указания адреса localhost, поскольку он правильно интерпретируется механизмом Docker, который работает на том же хосте, что и реестр:

```
$ docker tag amouat/<<название образа>:0.1 localhost:8080/<<название образа >:0.1
```

```
$ docker push localhost:8080/<<название образа>:0.1
```

если сейчас мы удалим локальную версию, то в любой момент можно извлечь ее из реестра:

```
$ docker rmi localhost:8080/<<название образа>:0.1
```

```
Untagged: localhost:8080/ <<название образа>:0.1
```

```
$ docker pull localhost:8080/ <<название образа>:0.1
```

Если мы ничего не удалим то после pull запросы мы получим следующую строку

```
688189e56d17: Already exists (Образ уже существует)
```

```
Digest: sha256:d20affe522a3c6ef1f8293de69fea5a8621d695619955262f3fc28852e173108
```

Механизм Docker обнаружил, что уже существует образ с тем же содержимым, поэтому в действительности происходит только повторное добавление тега. Обратите внимание на то, что реестр сгенерировал аутентификационный дайджест(digest) для этого образа. Это уникальное хэш-значение на основе содержимого образа и его метаданных. Образы можно извлекать из реестра по дайджесту, например:

```
$ docker pull localhost:8080/<<название образа>@sha256:\
```

```
D20affe522a3c6ef1f8293de69fea5a8621d695619955262f3fc28852e173108
```

Главным преимуществом использования дайджеста является абсолютная гарантия того, что извлекается в точности тот образ, который нужен пользователю. При извлечении (загрузке) по тегу можно оказаться в ситуации, когда имя тегированного образа было изменено в какой-то момент, но пользователь не знает об этом. Кроме того, использование дайджестов обеспечивает целостность образа, и пользователь может быть уверен в том, что образ не был поврежден во время передачи или во время хранения.

Главное обоснование использования собственного частного реестра – необходимость организации централизованного хранилища для группы разработчиков или для всей организации. Это означает, что потребуется возможность загрузки образов из реестра, выполняемой удаленным демоном Docker. Но при попытке обращения извне к локальному реестру, который мы только что запустили, выводится следующее сообщение об ошибке:

```
Error response from daemon: unable to ping registry endpoint
```

(Демон вернул ошибку: не получен ping-ответ от целевого хоста реестра)

```
https://192.168.99.100:8080/v0/v2 ping attempt failed with error: Get  
https://192.168.99.100:8080/v2/:
```

(v2 попытка ping – ошибка)

```
tls: oversized record received with length 20527
```

(tls: получена запись с размером, превышающим допустимый, – длина 20527)

v1 ping attempt failed with error: Get <https://192.168.99.100:8080/v1/ping>:

tls: oversized record received with length 20527

Здесь аргумент localhost заменен на IP-адрес сервера. Ошибка будет возникать при любой попытке загрузки образа с помощью демона как на другом компьютере, так и на той же системе, где работает реестр.

Демон Docker запретил соединение с удаленным хостом, так как этот хост не имеет действительного сертификата TLS . До этого установление соединения разрешалось только потому, что в механизме Docker предусмотрено особое исключение для загрузки с серверов, расположенных на локальном хосте. Чтобы решить эту проблему нужно Перезапустить каждый демон Docker, которому требуется доступ к нашему реестру, с аргументом `--insecure-registry192.168.1.100:8080`