Первая команда, с который мы начнем, это, соответственно, команда Docker Pull. Команда скачать изображение. И положить его в свой репозиторий для того, чтобы потом использовать его в дальнейшем.

И начнем мы с того, что посмотрим на консольную версию этой команды, на версию, реализованную в клиенте Docker. Почему? Потому что она делает примерно то же самое, что делает клиент в Docker SDK. И поэтому давайте на нее посмотрим. Соответственно, что она может делать? Она может скачивать изображение. То есть по умолчанию у нас только один параметр сюда – это NAME, у которого может указываться TAG или DIGEST. Соответственно, SSH-ключ для проверки целостности или для указания версии самого архива.

Соответственно, можно скачать все тэги или какой-то конкретный тэг. Можно указать, то есть – скачать все тэги для какого-то конкретного взятого изображения. Что это значит? Это значит, что изображение у нас включает в себя какие-то библиотеки. И вследствие обновления этих библиотек…, вследствие того, что автор решил включить в изображение что-то еще, естественно, версии разные. И эти версии, они все лежат в официальном Docker-ском репозитории, они все по-разному называются по тэгам. Вы можете, соответственно, выбрать, что скачать или скачать их все. Соответственно, эта команда выполняет именно скачивание всех версий тэгов одного изображения. Полезно, если у вас, например, есть какая-то локальная Docker Daemon, и вы хотите, чтобы разработчики сами там выбирали, и вы хотите, чтобы все это было закэшировано, чтобы никто ничего снаружи не качал. Пожалуйста, качайте все по тэгам, кладете в ваше изображение и даете разработчикам.

Вы можете указать платформу, потому что изображения могут собираться под разной архитектурой – под linux/386, под linux 464-битный, соответственно, под arm-овый linux. И то, что нас не очень интересует, это уже атрибут,

непосредственно, самого Docker-клиента – это скрыть, соответственно, логирование. В принципе, тут все просто. По сути, мы…, это такой замаскированный Wget с rest API под Docker, который там еще и положит все в репозиторий.

Давайте теперь посмотрим, как это в клиенте реализовано. Давайте начнем наше путешествие по Docker-клиенту, с создания самого Docker-клиента. Здесь, соответственно, что интересного? Здесь создается клиент с vararg-ом, со списком опций. Опция – это, по сути, некая функция, которая берет в себя ссылку на клиента и что-то там с ним делает, то есть загружает в него какие-то данные, инициализирует поля, что угодно, это уже на совести создателя опции. Мне этот подход нравится, потому что он позволяет не предполагать, что там сделано на уровне клиента, а просто посмотреть на списки опций, что сделано в опциях. И соответственно, из этого исходить.

Первое, что мы делаем, это у нас (НРЗ 03:11) – чиста функция, нет, не чиста функция. Функция высшего порядка – это называется по-русски. Которая, соответственно, вгружает в себя настройки из переменных окружения. То есть Docker_host, Docker версия API, путь к TLS-ключу, HTTPS-ключу. И соответственно, нужно ли верифицировать этот самый – TLS или нет? Если ничего из этого нет, будут использованы настройки по умолчанию, соответственно, Docker or host с какой-то стандартной версией API. Об этом, на самом деле, можно не парится. И без, естественно, какого-либо шифрования. То есть почему можно не париться? Потому что версию API мы можем задать либо жестко, с помощью WithVersion, которая вгрузит версию и переопределит, соответственно, версию API. Но это нужно из-за того, что у нас в Docker-е…, если вы помните, куча версий, и какая-то может быть из них несовместима или, наоборот, содержать в себе слишком много настроек.

Но есть вот эта вот одна чудесная опция, которой я в большинстве случаев пользуюсь – это WithAPIVersionNegotiation, которая вызывает версию Docker-а, и сама там по себе определяет, с какой версией клиент максимальной может работать. Всегда ее использую, в принципе. То есть случаев, когда мне нужно было жестко задать версию, у меня пока еще, наверное, не было. Соответственно, здесь я расставляю panic-и, опять же, это не по фен-шую, но для демонстрации, опять же, важно понимать, что у нас что-то здесь упало.

Соответственно, следующая команда – это ImagePull, то есть аналог этого DockerPull-а. В чем аналог? Мы сюда передаем контекст для управления синхронным запросом. Соответственно, мы указываем, какое изображение мы хотим скачать. Это аналогично команде вот этой, то есть аналогично указателю имени Image-а. И указываем PullOptions. PullOptions можно воспринимать, как некий аналог вот этих вот самых опций в нашем Docker-е. То есть вот этих вот самых – platform, all-tags и так далее. Сейчас вы увидите.

То есть, если мы на них посмотрим, то All – наша переменная для скачивания всех тэгов, RegistryAuth – это переменная, которая нам позволит, в случае провала регистрации, невозможности получить через защищенный, соответственно, registry изображение, поскольку registry может быть защищенным и позволять доступ только по логину и паролю. Например, вы сами можете поднять у себя такой safe, так называемый, registry. Соответственно, функцию для запроса привилегий. И платформа. То есть можно указать вот эту самую платформу, одну из. И в принципе, это все.

То есть ImagePull отдает, соответственно, readcloser – это стандартный, если вы помните, ответ на сетевой какой-то поток данных. Сетевой поток данных в нашем случае – это не sum Image. Sum Image будет…, положится в наш локальный registry, если таковой есть. Это поток ответов, так сказать, от Docker API.

То есть в нашем случае, я могу просто в конце функцию закрыть, когда закончится сетевой запрос, когда закончится поток. И просто скопировать его в Stdout. Давайте запустим и посмотрим, как это работает, что ж там пишется в нашем потоке и что у нас, соответственно, будет качаться. Можно увидеть, что Pulling from library/alpine, тут он получил Digest и скачал изображение для Alpine Digest. То же самое мы можем сделать, если мы вызовем Docker Pull Alpine, то, в принципе, можно увидеть, что клиент, соответственно, тут какие-то свои поверх дает подсказки, но, по сути, ответ от DockerAPI точно такой же.