Итак, давайте разберемся с главной командой Docker-а, с командой запуска контейнера. И, значит, здесь мы будем делать как бы все на чистовой, без учета того, какие у нас есть Image-и в системе, какие у нас есть ресурсы и так далее, что у нас там запущено. Мы стянем новый Image, опять же, alpine Image с официального Docker.io. Уже знакомой нам командой ImagePull. И вызовем команду создания контейнера. Команда создания контейнера, естественно, принимает в себя context в самом начале, а потом принимает конфигурацию контейнера.

Конфигурация контейнера – это вещи, которые Docker называет portable. То есть это те конфиги, которые можно спокойно переносить с host-а на host. То есть, ну например, в данном случае у меня есть Image, есть команда, есть (НРЗ 00:54), который нам объясняет, стоит ли прокидывать, значит, (НРЗ 01:00) систему или нет. И соответственно, можно привязать переменные окружения, можно привязать какие-нибудь volume-ы. То есть mount-ы, которые мы будем использовать, которые мы будем монтировать к этому контейнеру. То есть это вы можете прям взять и запустить везде.

И вторая переменная – это HostConfig, конфигурация самого Host-а, и она уже будет зависеть от того, какая система у вас включена, что у вас там внутри крутится. Она уже не является общей, как бы, она является уже конкретной под ваш Image.

И здесь у нас всякие интересные вещи, типа – PidMode-ов, запускателя контейнеров в привилегированном режиме. Соответственно, здесь можно указать спеки mount-ов, consistency, BindOptions и так далее.

И можно указать настройки различных DNS-ок. То есть здесь у нас уже то, что будет зависеть от самой системы, которая запущена внутри этого контейнера. И

поэтому Docker их разделяет. По сути, и то и другое – это конфигурация, но одна портабельная, одна общая, другая более приземленная и более частная.

Соответственно, сетевая конфигурация – это следующий параметр. Сетевая конфигурация у нас будет определять заголовки…, вернее не заголовки, а настройки всех Endpoint-ов, которые у нас существуют в системе, всех точек доступа. Здесь вполне стандартные сетевые настройки. После этого мы можем еще определить платформу. Если честно, про платформу мне сказать особо нечего, здесь мы просто определяем нашу архитектуру, нашу операционную систему, то есть создаём какие-то метаданные. Я этим пользовался один раз. И я уже, честно сказать, не помню, в чем там основной (НРЗ 02:59). Насколько я понимаю, это, действительно, только метаданные.

Последнее, что у нас есть, это контейнер name, то есть имя запущенного контейнера, как оно здесь будет отображаться. И после создания контейнера, мы получим, соответственно, от него вот такую структурку – ContainerCreateCreateBody, которая в себе будет содержать ID контейнера запущенного и какие-то возможные Warning-и, если Docker не сможет что-то с ними сделать

ID мы можем использовать при старте контейнера, соответственно, скормив сюда контексты ID и ContainerStartOptions, о котором я поговорю немного позже. Затем мы запускаем ContainerWait, то есть это асинхронная функция, ну то есть она возвращает (НРЗ 03:50) и ждет до определенного Condition-а, то есть ждет, что контейнер с таким-то ID зайдет в определенный Condition.

Condition-ы бывают, соответственно, NotRunning, то есть это или created, или exited, или dead, removing, или removed. То есть какие-то переходные процессы, соответственно. NextExit – это, соответственно…, ждет, когда контейнер сменится

в статус, который перед тем, как он выключится. Это полезно, если у вас висит какой-то (НРЗ 04:46) и смотрит на контейнеры, и вам нужно какое-то действие предпринять в момент контейнера. И соответственно, removed вызовется, когда контейнер был удален из системы.

Соответственно, мы здесь ждем, когда контейнер перейдет в какое-то состояние. И если контейнер стартанет с ошибкой, то здесь мы ее сразу увидим. Если контейнер стартанет без ошибки, можем посмотреть его StatusCode, затем показываем логи контейнера, о которых я тоже расскажу несколько позже. И после этого, после вывода логов, соответственно, мы видим результаты запуска контейнера.

Теперь давайте запустим наш код и посмотрим, что происходит. Соответственно, первое, что произойдет – это pulling нашего изображения, там нам уже знакомые, абсолютно, логи с нашего Docker daemon-а. Затем, мы ждем ContainerWait. Потому что, после запуска контейнера, у нас выполняется только одна команда, затем – контейнер выходит. То есть у нас этот select когда-то отработает, это не какая-то там персистентная команда, которая оставит контейнер в рабочем состоянии. После чего, после выхода контейнер вернет нам статус Code ноль. Это значит, что с контейнером все хорошо, стандартный код завершения приложения. Мы можем прочитать логи этого контейнера. И прочитать, что там записано.

Теперь я немного расскажу о контейнер ContainerStartOptions. Почему существуют эти CheckpointID и CheckpointDir. В Docker-е также, как и в виртуальных машинах, существует система checkpointer-ов, которая позволяет вам сохранить Statcounter-а. И как бы его в следующий раз…, стартануть с этого Stat-а. Ну то есть сделать что-то там машина, вы говорите: «(НРЗ 06:12) остановись…, остановитесь», получаете CheckpointID оттуда. И потом можете его использовать для того, чтобы запустить контейнер с того же места. CheckpointID,

соответственно, и CheckpointDir, потому что Checkpoint может храниться где угодно, это такое некое сохраненное состояние в виде файла. И соответственно, прекрасно вы можете остановить и продолжить выполнение контейнера.

И теперь давайте внимательно посмотрим на ContainerLogs. ContainerLogs, соответственно, казалось бы, ну что там такого. Берешь, запускаешь контейнеры, печатаешь логи из него, все хорошо, все счастливы. Здесь в чем фишка? Можно увидеть, что у меня есть отдельная команда для печатанья логов, но здесь io.Copy, а в моей другой команде, здесь, соответственно, stdcopy.StdCopy.

В чем прикол? Запущенное приложение в контейнере, как и любое приложение в Linux-е может писать либо в Stdout, либо в Stderr. То есть некоторые приложения это использует для того, чтобы, например, в Stderr только ошибки писать, а в Stdout писать какую-то информативную часть. Ну и удобно, соответственно, переключаться. Вы можете посмотреть, где ошибки, можете посмотреть, где у вас просто информация. И, соответственно, ContainerLogs устроен таким же образом. То есть ContainerLogs умеет выдавать, либо в режиме, соответственно, прямого стрима в Stdout, либо мультиплексировать, соответственно, вывод Stdout и Stderr, который потом можно разделить с помощью функции StdCopy. Которая уж, соответственно, выведет то, что там у вас в ошибках…, в информации в Stdout, то, что у вас в ошибках в Stderr. Ну и соответственно, можно перенаправить куда угодно файл или как хотите. И соответственно…

Еще что интересного из команды ContainerLogs? Опять же, если вы используете io.Copy, то все будет у вас вместе.

Соответственно, еще есть у контейнера опции. То есть показывает ли Stdout, показывает ли Stderr. С какого… Откуда до куда показывает, какие там Stamp-ы…, в смысле, показывает TimeStamp-ы. Делать ли tailing, потому что контейнер, опять

же, ContainerLogs возвращает ReadCloser. То есть, если у вас контейнер постоянно запущен, вы можете установить просто запись, постоянно копировать в логи, логи на экран или куда-то там. Это достаточно удобно. То есть сейчас по опыту все приложения, которые как-то, каким-то образом обслуживают Kubernetes и Docker, они именно фокусируются на том, как бы сделать логи поудобнее. В принципе, достаточно крутой стартовый функционал у вас уже есть с самого начала.