Следующий концепт – это Periodic Job, она же джоба, она же, соответственно, работа. Здесь всё, тоже, довольно просто. Kubernetes запускает контейнер с процессом, который имеет какое-то окончание. То есть это не веб-сервис, который бесконечно ждёт каких-то входящих подключений, это не база данных, которая также ждёт каких-то входящих подключений и ещё что-то делает, там, в фоне. Это какой-то конкретный процесс, это миграция, не знаю, там, обработка файлов, сохранение данных. И после того, как она сделает то, что должна сделать, она завершает работу.

Зачем нужны эти самые periodic Job-ы. Ну, во-первых, если у вас есть, например, какая-то, вот, например, у вас есть аналитический сервис, вы собираете какие-то данные, какую-то первичную информацию, а потом раз, скажем, в неделю, вам нужно построить какой-то отчёт, то есть вам нужно эти данные агрегировать, прогнать, положить в какую-то базу и предоставить отчет.

Нужен ли вам процесс, который будет в отдельном контейнере торчать неделю и ждать. Нет, не нужен. Вам нужно поднять контейнер по расписанию, соответственно, сказать Kubernetes-у, что подними мне раз в неделю контейнер, вот с таким-то процессом, и с такими-то настройками. Он, там, соберёт данные с базы, что-то проагрегирует и отдаст клиентам. Он не должен постоянно быть запущенным.

Если, опять же, вам нужно делать работу раз в день, раз в неделю, зачем ждать, зачем тратить ресурсы на ожидание этой работы. И очень часто джобы используют системные администраторы, чтобы обновить сертификаты SSL или, чтобы накатить какие-то, там, update-ы на внутренние сервисы системы. Ну, такие, которые нельзя накатить при загрузке контейнера, естественно, и довольно удобно для этого использовать, именно, джоба, одноразового джоба, которого вы создаёте, и они там продвигаются и, соответственно, что-то делают.

Итак, джобы. Причём тут, вообще, Go. Писать джобы можно на любом языке. Это же, на самом деле, обычный скрипт, обычный скрипт с каким-то известным концом. Ничего страшного, если вы пишите его на «Питоне», на PHP, на Perl-е, на чём угодно. Аргументы, которые я могу дать, это, желательно, чтобы эти работы, опять же, запускались с каких-то легковесных контейнеров. И не тянули за собой интерпретаторы, трансляторы и, что угодно там, дополнительные всякие тулзы, которые служат для запуска приложений. Опять же, вот, в случае с работами, это не так важно, не так критично, но приятно, когда работа работает…, работа работает быстро.

Соответственно, если вы делаете какую-то работу админскую, это как правило, инфраструктурные API, и у golang-а довольно много хуков к инфраструктурным API. Опять же, он такой не один, у «Питона» их, наверное, еще больше, но на низком уровне с golang-ом довольно приятно работать с инфраструктурными API, и для написания всяких скриптов.

И в отличие от языков более высокого уровня, то есть от языков, которые могут подавлять ошибки, у которых это настраивается даже в функциях, golang ошибки выкидывает. На ошибках вы можете паниковать, у вас полный прозрачный контроль. И желательно, в терминологии Kubernetes-а, в работе, если какая-то, там, есть ошибка, желательно всё откатывать и умирать. Потому что таким образом вы чётко видите результаты выполнения. Вы смотрите на список работ, видите, что работа умерла и всё, и вам не нужно заходить в логи, смотреть метрики и проверять, что там ещё у этой работы поломалась. Поэтому golang для таких вещей подходит, ну, довольно, хорошо. Но, опять же, не идеальный, опять же, это не буду ни для чего пропагандировать, не самый лучший, не самый худший инструмент для работы, скорее в области лучших.

Разберёмся со структурой джоба. Здесь так же, как у нас менялись от deployment-а и pod-а версии от v1 и до apps/v1, у нас теперь версия batch/v1. Тип работы – это Job, соответственно, метаданные с именем и spec-и с template-ами. То есть здесь, уже точно так же, как у нас с deployment-ами. Здесь spec и template, и ещё один spec. Можно также перезалить, там, всякие селекторы и прочее.

У нас, мы также определяем контейнеры, потому что джоба поднимает pod вместе с контейнерами, выполняющими работу, и мы можем объявить джоба в restartpolicy. То есть, в данном случае, джоба будет перезапущена только, когда у нас pod по какой-то причине не смог подняться, именно сам pod, как Kubernetes-овский концепт. Если у вас приложение вывалилась из контейнера, джоба перезапущена не будет. Вы можете определить on failure, и тогда джоба будет перезапущена, даже если у вас контейнер провалился. On failure, он же (НРЗ 05:13). Ну, так вот, backofflimit определит, сколько раз, если вы перезагружаете джобу, и ее надо перезагрузите, и актив дедлайн Seconds убьёт джобу, если она выполняется дольше какого-то определенного времени.

Давайте теперь эту джобу запустим и посмотрим, что она создаёт и как это, вообще, выглядит. Итак, (НРЗ 05:33) наш…, нашу джобу – kubectl Apply -f job.yaml. И видим, что у нас создалось джоба. Теперь, если мы проверим список pod-ов, мы увидим, что наш, значит, pod для джаба уже создан и уже запущен. Если мы посмотрим на список…, на список работ, то мы увидим, что у нас заработало, мало того, уже что она была сделана, она уже и выполнилась. Если мы сейчас сделаем describe, мы увидим, что мы запустили в контроллеры, когда была создана и, когда завершилась, вот этот Report от контроллера. Ну, и теперь, если мы увидим job, мы видим, что…, увидим…, посмотрим на pod-ы, мы видим, что статус у него completed, значит ready не один из pod-ов, и мы можем получить с него логи. Логи у нас, соответственно, 2001 символов числа Пи, расчётная задача.

Вот примерно так это работает, эти pod-ы, останутся в статусе completed, пока вы их оттуда не уберёте, либо пока вы их не уберете, либо пока вы не удалите джобу. Джобе можно, насколько я помню, задать ttl, чтобы, вот эти все pod-ы не видеть, и контроллер, тогда, по прошествии какого-то определенного времени, эти самые pod-ы, просто уберет для того, чтобы мы их больше не видели.

Ещё одна вещь, довольно часто используемая, как паттерн Kubernetes, связанный с работами, это CronJob. Разница между работами, помимо, описания, мы видим, что здесь есть Jobtemplate, у него spec, в нем template, вон там дальше, уже все описания джоба. То есть как-бы CronJob, это обёртка над джобом, который вы описываете уже в Jobtemplate. То есть здесь всё, что мы описывали в Job-е, описывается в CronJob-е. Ну, помимо того, что там, name, например, может быть унаследован из metadata-ы CronJob-а. Разница в том, что вы указываете ему schedule, указывайте ему обычные Cron-овский schedule, с какой частотой запускаться внутри кластера, и он сам создаёт работы, когда ему надо.

Давайте создадим, в принципе, эту работу, и я покажу, какой ресурс надо, просто, вызывать. Тут, кроме как ресурса и контроллера, особых отличий от самой работы, ну, и соответственно, запуска по расписанию, просто с какой периодичностью, особых отличий от работы у вас не будет.

Здесь вы видите пример применения мной уже команды apply CronJob, здесь мы видим, что в pod-ах нет у нас никаких pod-ов, и в джобах нет ничего, потому что CronJob-а, у нас всё ещё не заскейджелилась. Несмотря на то, что она должна выполняться каждую минуту. Соответственно, здесь мы видим имя, видим её расписание, обычный Cron файл, соответственно, видим (НРЗ 08:32) она или нет, то есть остановлена она или нет, сколько у нас сейчас активных работ, когда она последний раз выполнялась и её возраст. Соответственно, со suspend отвечает за то, будет ли работа стоять на паузе. Полезно, если вы, например, делаете

какие-то техническое обслуживание кластера раз в неделю, backup-ы снимаете, или там логи удаляете, или, не знаю Elastic кластер компрессируете. А потом, раз, и вы, там, новые диски подключили, и вам нужно какое-то время поработать, без этого всего. Ну, suspend-ите эту работу. Вот таким образом, patching-ом. То есть patching нам позволяет добавить какие-то моменты в json, описывающие работу. То есть, если мы, опять же, сделаем, describe CronJob Hello, мы здесь увидим, опять, вот, этот вот suspend. То есть мы увидим, вот, это описание работы, и мы можем им, естественно, управлять с помощью patching-а. Я (НРЗ 09:30) эту работу, и видим, что она больше не остановлена.

Соответственно, можно теперь увидеть, (НРЗ 09:41) у неё 29 секунд назад, можно посмотреть на работы, увидеть этот Hello, который уже выполнился, описать…, описать, соответственно, эту работу, посмотреть на то, какой контейнер она создала. Соответственно, какой pod. И получить логи с этого pod-а. В смысле, да, нам не нужно указывать, мы просто указываем объект.

Всё, вот наша работа. Она будет создавать новые джобы. И соответственно, будет делать это по расписанию