Последний паттерн, о котором мы поговорим сегодня, мы затронем его только в общих чертах, потому что ему посвящена следующая лекция, это оператор Kubernetes. Соответственно, оператор Kubernetes – это фича, позволяющая вам создавать уже готовые ресурсы автоматизации.

Например, вы можете написать свой оператор, для вашей базы данных и в данном случае, вы Kubernetes просто указываете, моя база данных, типа «бармен, мне как обычно», и Kubernetes уже знает, что с ним делать. Какие ему ресурсы дать, какие ему подсоединить коннекторы, что там настроить, какие там скрипты запустить, как это будет выглядеть. То есть, вы сокращаете таким образом декларацию. И можете ещё и какие-то условия там внутри задавать.

Соответственно, операторы управляются контроллерами. То есть, смысл в том, что оператор – это связка контроллера, который знает, что делать с этим конкретным ресурсом. И описание самого ресурса, который называется Custom Resource Definition, декларативного. То есть, это все вместе называется оператор. Выбрано так, потому что оператор, по аналогии с человеком-оператором, которому ты говоришь, там, не знаю: «Вызови мне полицию». И он тебя не спрашивает, какой номер у полиции или куда там звонить, он уже знает кому позвонить и что им сказать.

То же самое и здесь. Вы заранее определяете контроллер, который выполняет нужные действия. И привязываете ресурс, с которым он будет работать. Kubernetes видит ресурс, смотрит есть ли такой контроллер у него, который обрабатывает этот ресурс, передаёт задачу контролеру, контролер всё это внутри Kubernetes поднимает.

Вопрос, естественно, зачем нужны операторы? То есть, вы, в принципе, можете все то же самое написать декларативно, как-то упаковать, сделать шаблоны.

Зачем вообще, эти операторы? Во-первых, операторы позволяют классифицировать ресурсы вашей системы. То есть, если у вас есть какая-то типовая задача, который постоянно повторяется, и вы главное, хотите за ним наблюдать, каким-то образом с ним взаимодействовать, есть смысл написать кастомный оператор.

Если… Опять же, взаимодействовать, это что значит? Это значит, что вы хотите, чтобы Kubernetes-овском API, с которым вы работаете, был этот ресурс и вы могли сказать, например: «Покажи мне все ресурсы этого типа. Все ресурсы моей, не знаю, (НРЗ 02:26)». И он такой раз, через kubectl, kubectl get (НРЗ 02:30),и такой раз – и все показал. Классно? Классно.

И вы можете добавлять вызовы в API, в Kubernetes API. Расширить ее по требованию вашей компании, кастомизировав, так называемый API, без лазания в код Kubernetes. Это даже не расширение Kubernetes, те же контроллеры для операторов, могут быть спокойно, обычными deployment-ами внутри вашей системы, они не принадлежат самому Kubernetes-у. Это довольно удобно.

Итак, давайте разберёмся кратенько с оператором. В этой лекции я не буду детально вдаваться в подробности операторов, мы разберём анатомию операторов в следующем занятии. Для того, чтобы показать, что делает оператор, я взял уже готовый оператор, деплоящий базу данных maria, который вам позволяет сохранить усилия, не писать persistent volume claims, всякие там Stateful set-ы. Он позволяет просто, задеплоить оператор, а потом оператору сказать: «Что, задепой мне базу данных». И оператор сам все сделает.

Соответственно, самое главное часть оператора – это deployment, который содержит в себе, непосредственно, оператор. То есть контроллер этого оператора, тот самый кусок софта, который нам будет депоить нашу базу данных. Который

сообщит Kubernetes-у, как обращаться, когда мы хотим… Как ему обращаться с deployment-ами, с Stateful set-ами и прочим, когда нам нужна база данных.

Соответственно, здесь, в принципе, тоже самое. Здесь выполняется команда mariadb-operator, который бесконечно слушает, есть ли какой-то для оператора ресурс и какие-то переменные окружения. И соответственно, оператор поставляется также с ролями, на которых я сейчас не буду подробно останавливаться. Роли, позволяют Kubernetes понять, что, значит, наш оператор обладает доступами ко всем этим ресурсам и может все с ними делать.

Биндит эти ресурсы к оператору, кластер (HP3 04:41), то есть для всего нашего кластера. Создаёт сервисный аккаунт. Мы тут объявляем дополнительные ресурсы для того, чтобы наши операторы могли вызывать persistent volume claims, так называемый, приатачивать наши базы данных. Приатачивать диски к нашим базам данных. В данном случае у меня диски локальные. То есть я запускаю Kubernetes локально, поэтому у меня storageClass, это hostpath. Это значит, что дисковое пространство будет браться, непосредственно, с моей машины.

Давайте задепоим этот оператор и посмотрим, что у нас образуется в Kubernetes, в результате наших манипуляций. Давайте теперь задепоим наш оператор, с помощью уже нам известной команды kubectl apply -f. Я буду депоить папочку целиком. И вот, мы видим, что у нас persistent volume-ы наши созданы. У нас есть наш deployment для нашего оператора, для нашего контроллера, если быть точным. У нас созданы для этого контроллера role и так далее.

И мы, на самом деле, можем это сейчас, всё увидеть. Мы можем видеть, что наш контроллер упал с ошибкой. Почему он упал с ошибкой? Потому что, для работы контроллеру, тут вот у меня stacktrace, который не очень красиво отформатирован. Для работы контроллеру, ему нужны серди CRD. Customs Resource Definition. Это,

как раз то, что, в принципе, определяет оператора. Ресурсы, за которыми оператор наблюдает и, которые он может обрабатывать. Давайте на них тоже посмотрим

Customs Resource Definition – это такой сигнал контроллеру о том, что если у вас создан ресурс такого типа, его надо обработать. Точно, примерно, так же, точно примерно, в общем, так же как вы, например, создаёте Stateful set, daemon set и так далее. То есть, принципе для Kubernetes, это тоже всё ресурсы, у них есть контроллеры. Это как раз то, что я показывал в предыдущих частях лекции.

Здесь мы создаём свой ресурс. У него есть имя, он mariadb. Есть множественное число, есть единственное число, соответственно, есть у него scope, где он этот ресурс валиден, есть версия. И версиях указана у нас схема, какое у него там описание, какие у него поля должны быть заполнены обязательные, какие не должны быть заполнены. В общем, и без этих ресурсов, если вы их слушаете и определяете, ваш контроллер работать не будет. Он будет, соответственно, падать. Kubernetes-у нужно знать про эти ресурсы для того, чтобы ваш контроллер мог работать. Давайте эти ресурсы мы, соответственно, задепоим. Сходим в оператор и сделаем kubectl apply -f на CRDS. И, соответственно, посмотрим теперь pod-ы.

Ну и вот, наш контроллер поднялся, счастливо описал в логах, что всё он, значит, нашёл, все наши ресурсы. Стартовал Worker-ов, для различных контроллеров. И теперь, когда у нас есть ресурсы, мы можем вот такой вот Kunststuck сделать. Мы можем сделать получение ресурсов типа mariadb. Их сейчас нету, но по факту, теперь у вас появился новый Endpoint в вашем Kubernetes-овском API, и он расширился на эти вот кастомные ресурсы. Вы их можете посмотреть.

В папочке Kubernetes у меня есть, соответственно, уже описание под кастомный ресурс. Можно увидеть, что у него есть a API version. У API version появился свой

Endpoint со своим, соответственно, постфиксом. Свой тип ресурса. Вы можете, опять же, указать имя и спецификацию. Вот, как вы ее указали, значит, в ваших definition-ах, так она здесь и будет.

То есть, здесь мы указали, что нам нужно знать про database storage path, про storage size, про имя базы данных и так далее. Здесь это вот, всё есть у нас в виде наших, значит, свойств. И теперь, мы этот ресурс можем, соответственно, загрузить в базу данных. Вернее, загрузить в наш Kubernetes-овский кластер, просто применив ресурс. То есть, здесь уже нет…, вы можете заметить, нет никаких deployment-ов, persistent volume claim-ов. Вот простой файлик, который просто описывает какие-то базовые вещи. Давайте, теперь мы заходим бабочку Kubernetes, делаем там kubectl apply -f MariaDB. Maria, я сказал.

И вот, у нас создаётся не deployment, не deployment-ов, persistent volume claim, не Stateful set, у нас создается ресурс нашего типа mariadb. И теперь можно здесь, где mariadb. И вот, мы видим наш ресурс, который существует уже 15 секунд. Мы можем ему также describe-нуть. Describe mariadb. И здесь мы видим, что у нас есть какие-то там managet fields, что у нас тип ресурса mariadb. Что у него вот такие вот спецификации. Что вот у него есть вот такой вот менеджер и что…

Соответственно, что этот ресурс существует как бы отдельно в Kubernetes. И теперь, если мы посмотрим на deployment, мы увидим, что у нас есть mariadb main server. То есть, мы его не создавали самостоятельно, его создал наш контроллер за нас. Приделал к нему persistent volume claim, создал Environment variables, который уже сам сервис потребляют. Засунул туда image, который ему нужно. Сделал ему выбрать label-ы, чтобы было понятно, к чему, что принадлежит.

И соответственно, deployment контроллер у нас скалировал реплику. И, в общем-то, как-то вот так это работает. Опять же, реплика в свою очередь создаёт

pod-ы. У нас поднялся счастливо pod. И как бы у нас готова, в принципе, к применению база данных. То есть, нас не волнует Complexity базы данных, мы все это Complexity спрятали внутри нашего оператора, который этим всем занимается, и он всё поднял. Как минимум, это одно из применений оператора, другие применение операторов, мы рассмотрим в следующей лекции.