Прежде, чем мы приступим непосредственно к разработке операторов, давайте посмотрим, какие у нас есть непосредственно пути – это сделать. Какие есть инструменты и утилиты, помогающие нам в этом процессе. Я знаю 4 способа сделать кастомный оператор или кастомный контроллер.

Первый из них – это естественно, сделать всё самому, сделать всё руками. В принципе, это довольно доступный способ. Никакого дополнительного софта не требуется, нужен только интерпретатор или компилятор. Ну, в нашем случае нужен Golang. И знания, как это делается, и, в принципе, всё у вас заработает. Ну, соответственно Docker, Kubernetes для тестирования и всё.

Я таким способом не пользуюсь, я использую утилиты, которые помогают мне с, соответственно, построением операторов, и их существует на данный момент 3 штуки. Давайте посмотрим на каждую из них.

И первая утилита, с которой мы познакомимся сегодня — это утилита kubebuilder. Соответственно, самая, наверное, старая утилита для написания кастомных операторов из тех, которые сейчас существуют. Требует версию Go 1.16, для версии 3.1, которой мы будем пользоваться. От 3.1, соответственно. Версию Docker-а 17.03 и кластер Kubernetes-а от версии 1.11.3, и kubectl, соответственно.

Дальше мы устанавливаем kubebuilder. Существуют сборки под Linux и под Mac. Соответственно, под Windows kubebuilder-а нету. И мы, в принципе, готовы генерировать код и собирать наши приложения. То есть, по сути, kubebuilder просто генерирует нам некоторый код для того, чтобы мы не писали никакие (НРЗ 01:48). Не писали, там, те части, которые нам, ну, которые общие для всех операторов, ручками. И избавили себя от необходимости, там, разруливать права доступа и легко создавали новые ресурсы с помощью kubebuilder.

Что ещё хотелось бы добавить про kubebuilder – это то, что у него совершенно офигенный есть tutorial, в котором описаны ещё и основные концепты для генерации, генерации операторов Kubernetes-а. Он настолько хорош, что следующий наш, соответственно, кандидат, претендент использует kubebuilder-овские хуки – вот он здесь, например, Kubebuilder project layout, там ещё про менеджера рассказывается и так далее. С отсылкой к kubebuilder-овской документации.

Ну, а, собственно говоря, следующий проект, который является, ну, как, можно сказать, я бы сказал, конкурентом, но, поскольку это некоммерческий мир разработки, он скорее является наследником, что ли, kubebuilder-а. Даже не знаю, как это бы пояснить. В общем, стандартный способ, который, в принципе, используют большинство моих коллег, который чаще всего можно встретить во всяких tutorial-ах. И которым, в принципе, пользуются кастомные разработчики — это Operator SDK.

По сути говоря, делает он то же самое, что и kubebuilder, позволяет разрабатывать на Go, позволяет писать, управлять этим самым циклом reconciliation – циклом, когда контроллер сверяет состояние с помощью Ansible, либо сделать Helm-чарт, в котором будет содержаться логика. То есть, здесь уже несколько вариантов. На мой взгляд, если писать на Go, то, соответственно, ну, после него это самый, наверно, низкоуровневый как бы подход. И после него будет уже понятно, что происходит в Ansible и Helm-е. То есть, разбирать их смысла отдельно нет.

Касательно документации, как я уже сказал, очень много берется с, очень много берется с Kubebuilder Book. То есть, многие концепты объясняются оттуда. Ничего удивительного, концепты, так сказать, общие для Kubernetes-а, почему бы и не переиспользовать документацию. Соответственно, это второй способ разработки операторов.

Третий способ, который прям совершенно радикально отличается от двух остальных — это Metacontroller. Это add-on, который устанавливается в сам Kubernetes и предоставляет уже контроллер, выполняющий логику других контроллеров. То есть, какая, какая в этом идея? Что большинство кода, который мы пишем для кастомных операторов, он общий. И по сути, мы можем эту вот саму суть оператора, вот эту бизнес-логику вынести отдельно. И использовать Metacontroller для вот таких вот, коротких совершенно, определения суперкоротких функций, коротких контролеров, и их загрузки с помощью самого Metacontroller в Kubernetes-овский кластер.

То есть, Metacontroller на себя берёт отделение логики reconciliation, вот этого вот всего, низкоуровневое, низкоуровневого уровня, всякой этой вот, низкоуровневой приблуды. И вы пишете только бизнес-логику. Подход имеет право на существование. Это мне чем-то напоминает, эти (НРЗ 05:22-05:25) и прочее. То есть, вы просто задаете функцию с логикой и описываете, ну, с помощью документов, документиков, каким образом, что будет происходить.

В рамках этих лекций мы совершенно точно не будем разбираться с Metacontroller-ом, поскольку, опять же, если вы понимаете, как низкоуровнево пишутся операторы и контроллеры, вы с Metacontroller и так разберётесь. Если у вас под капотом операторов происходит какая-то магия, то Metacontroller может ваши потенциальные проблемы только усугубить. Поэтому здесь только есть смысл этим пользоваться, когда хочется сэкономить силы, но вы в принципе понимаете, как это работает. Надеюсь, к концу лекции это будет намного более понятно.