Куда же написание operator-ов без Best practices. Давайте на них взглянем. У нас должен быть 1 operator на приложение, на какой-то компонент приложения. Если, например, ваш сервис состоит из какого-то веб-сервиса, из какой-нибудь Кафки, из какой-нибудь базы данных, у вас должно быть 3 разных controller-а. Они могут внутри одного контейнера, внутри 1 operator-а, управляться 1-м менеджером, но это должно быть 3 разных controller-а. Один отвечает за ваше приложение, другой отвечает за deploy базы данных, третий отвечает за deploy Кафки. И это не должен быть 1 operator, который внутри своего reconciliation цикла такой: «Ага, так, у меня (НРЗ 00:46). Так, вот сейчас посмотрим на Кафку. Так, а вот сейчас мы загрузим приложение», – такого быть у вас, естественно, не должно. 1 operator делает 1 работу. Так и отлаживать проще, и test-ить проще и вообще. Много вещей становится, на самом деле, проще.

Каждый компонент приложения управляется своим оператором. То есть я только что про это сказал, что, если у вас есть несколько компонентов, у вас под него написан каждый controller, есть отдельные CRD, потому что вы можете захотеть настраивать каждый компонент отдельно. А если у вас какой-нибудь…, композиционная логика, когда нужно просто последовательность запустить, то у вас должно быть…, должен быть оператор, который оркеструет operator-а. То есть operator, который говорит другим operator-ам, когда что должно запускаться, создает их работы. Так как, например, работает deployment, создавая pod-ы или Statefulset создавая Persistent Volume Claim.

Если вы делаете несколько operator-ов, controller-ов внутри одного operator-а, внутри одного менеджера, то у вас Custom Resource Definition должны принадлежать конкретным controller-ам, конкретным operator-ам, потому что, если у вас есть какой-то share CRD, который принадлежит и одному, и другому operator-у, у вас может возникнуть ситуация гонки. Возникнуть ситуация, когда вам нужно отладить логику, вы не понимаете, какой operator реализует эту логику. И

очень много всяких прикольных вещей в синхронизации, которые можно избежать и нужно избегать путем жёсткой привязки одного CRD к operator-у. Иногда это неудобно, иногда больно, иногда приходится делать костыли в виде того, что 1 operator читает статус из другого CRD и так далее, но именно конкретно логика работы CRD должна принадлежать 1-му operator-у.

Обратное правило то, что у нас CRD не принадлежит нескольким operator-ам, точно также, что у нас 1 controller на 1 CRD. У нас не делается несколько controller-ов на 1 CRD. Собственно говоря, уже про всё это рассказал. То есть в обе стороны работает, у нас нет нескольких operator-ов, которые обрабатывают одну CRD. У нас нет нескольких середин на 1 controller. 1 CRD, 1 контроллер – это поможет избежать многих проблем.

Мой совет использовать тулзы для построения операторов. Он же, в принципе, советуют разработчики этих тулзов. Это не потому, что хочется их продать, тем более что они бесплатные, а потому что вы можете что-то забыть в процессе построения. Тулзы, и в принципе потратить много времени на установку этого Boilerplate-а, на то, чтобы написать менеджеры, на то чтобы скачать все эти библиотечки, подключить. Не вижу в этом смысла, когда есть удобные тулзы.

И совет, который не настолько очевиден, было для меня, когда я начинал с operator-ами: не хардкодьте имена ресурсов или namespace. Если у вас что-то нужно опрашивать или что-то получать, или где-то что-то создавать, лучше сделать это конфигурируемым в вашем CRD или в ваших настройках operator-а, нежели чем хардкодить это внутри, потому что потом вы этот код задеплоили, про него забыли. Уже несколько версий прошло, его надо сменить, надо опять открывать. Зачем, если проще вынести? В общем, в целом – все конфигурируемые вещи лучше выносить в отдельную конфигурацию.

Следующий набор Best practices – это опять же про хардкодинг namespace-ы. Ничего, что operator-а, не откуда он берет ресурсы, не на что он смотрит – мы не делаем хардкоженным. Мы делаем конфигурируемым, даже если вы говорите, что: «Мой operator смотрит на system и больше никуда». И сделайте ему конфигурацию system. Кто поменяет, будет пользоваться – его проблема. Можно в description-е warning написать. Просто если вдруг захотите из system куда-то переехать – вы переедите, а если у вас всё захардкоженно – вы уже никуда не переедите.

Совет, который я видел, дают на всяких форумах, конференциях и так далее, использовать семвер, то есть основная версия…, в смысле, мажорная версия, минорная версия и опциональная версия. Эти 3 циферки через точечку, мне нравится этот подход, в принципе. Я так deploy-ю контейнеры. То есть для операторов я так deploy-ю контейнеры. В целом, если вы правильно ведете патчноты, то хорошая практика. Если у вас патчнотов нет или вы документацию ведете плохо, используете любое версионирование, это абсолютно не можно. Можно использовать: «1, 2, 3, 4, 5». То есть тут только в купе с документацией работает, сам по себе я не знаю… В этом, на мой взгляд, не так много смысла.

Используем OpenAPI spec для схем CRD в принципе, если вы пользуетесь тулзами, у вас не будет такой проблемы использовать какие-либо другие spec-и. У вас всегда будет OpenAPI, просто уже стандарт. Даже если есть возможность использовать что-то ещё.

Операторы должны предоставлять метрики для сбора, тут не строгое требование, но с метриками вам будет проще. Вы будете просто знать, сколько у вас там Cronjob operator сделал job-ов. Сколько у вас Mariaddb operator имеет deployment-ов и так далее, но это вам будет…, сильно облегчит задачу. Плюс, вы

можете всегда трекать здоровье operator-ов и видеть на вашей инфраструктурной графане, например, что все operator-ы работают нормально.

И последний совет – operator-ы убирают за собой. Если operator создает какие-то дополнительные ресурсы, пишет какие-то базы, хранят данные в конфигах, и они им не нужны, то лучше в конце цикла recancelation убирать за собой или сделать отдельные requiring для того, чтобы operator мог убрать данные, оставшиеся после себя. Это просто правило хорошего тона – не позволяет вам не захламить кластер в нулевое состояние и счастливо и долго продуктивно работать.