Следующая концепция в нашей библиотеке концепций, это концепция Stateful set-а. Соответственно, когда мы запускаем свои приложения, мы можем указать Kubernetes-у, сколько экземпляров приложений мы хотим запустить. То есть мы говорим, что вот у нас приложение, вот контейнер, или два контейнера, или три контейнера, сколько вам нужно – они все в одном pod-е. Это маленькая экосистема приложений. Kubernetes оперирует уже этим pod-ом.

Мы говорим: «Мы хотим 5 этих pod-ов». И Kubernetes нам их, соответственно, распределит как-нибудь. И он будет считать, что эти pod-ы не имеют состояние. То есть, если вдруг наше приложение надо перенести на другую node-у. Или оно вдруг не отвечает. Или вдруг нужны ресурсы под что-нибудь более важное для Kubernetes-а. Мы можем одну из этих node, одну из этих реплик, они так называются Replica Set-ы. Один из этих deployment-ов погасить. И потом зарескеджить где-нибудь еще. То есть, это не так страшно, (НРЗ 01:06) никакие состояния в себе не хранит. И в общем-то, отлично.

Все было бы хорошо, но вопреки…, кстати, почему-то популярному мифу, в Kubernetes-е довольно часто хранят базы данных, это довольно удобно. Несмотря на то, что, конечно, базы данных захощены на железках более производительные. Из-за того, что там нет Kubernetes-а. Но всякие различные пробы, метрики и прочее, удобно держать в Kubernetes-е, удобно скейлить вверх-вниз. И если есть желание, заплатить немножко сверху и получить скейлируемую систему, то довольно-таки удобно. Но, базы данных не являются Stateless, базы данных являются Stateful приложениями. Их вообще задача – это хранить State. Это, в общем-то, то, зачем они были созданы, зачем написаны.

И соответственно, существует ненулевая вероятность, что база данных при погашении ее Kubernetes-ом или при перезагрузке потеряет свои данные, потому что хранилища в Kubernetes-е такие же, как и в Docker-е. То есть они эфемерные,

они не теряются, когда приложение перезагружается. Для того, чтобы базы данных могли существовать внутри экосистемы Kubernetes-а, придумали такое понятие, как Stateful set.

Stateful set – это ваши объявления того, что у вас будет приложение хранить какое-то состояние и его реплики могут чем-то отличаться. То есть, например, в случае с Postgres-ом, несколько instance-ов Postgres-а не могут быть (НРЗ 02:40). Если они хранят одну и ту же базу данных, они запутываются, кто из них мастер, они не поднимутся в таком положении, они могут быть только (НРЗ 02:49), ну и так далее. То есть вы делаете Stateful set, вы говорите, что один из Stateful set-ов мастер, все остальные у нас slave-ы. И сколько бы вы Stateful set-ов не подняли, каждый Stateful set поднимется со своим подмонтированным хранилищем, который в себе будет содержать данные. Строго говоря, вы можете его подмонтировать к вашим клиентским deployment-ам, к Stateful set-ам. Это идет как бы…, как «Инь и Янь». И slave-ы, соответственно, будут знать, что они slave-ы, они будут знать, что они мастеры, они будут знать, что они часть Replica Set-а. И синхронизироваться друг с другом.

Несмотря на то, что, на мой взгляд, применение Stateful set-а довольно очевидно, давайте пройдемся по основным пунктам, зачем. Во-первых, не все приложения могут быть Stateless. Если мы поднимаем несколько экземпляров одного приложения, которые являются Stateful, нам неплохо бы иметь контроль за состоянием этого приложения. И каким-то образом дать Kubernetes-у понять, что вот это вот нельзя перезагрузить или просто выключить.

Replica Set. То есть репликация ваших deployment-ов вам не гарантирует at most once. У вас может быть при Replica Set-е 3, например, 6 запущенных приложений. Просто потому, что 3 у вас сейчас выключаются, 3 у вас стартуют, не знаю, 2 работают, вообще 8. Потому что Kubernetes ими управляет совершенно

произвольно, руководствуясь ему известными принципами, только ему известными принципами. Особенно это касается облачных провайдеров, у которых там могут сервера упасть, что-то перезагрузиться, у data center-а проблема со связью, Kubernetes выкинул эти node-ы из ротации. Запустил другим node-ы в ротацию. И соответственно, при этом у вас может это все перезагружаться. А в случае с базами – это фатально. Потому что 8 реплик Postgres-а и 3 реплики Postgres-а – это довольно-таки разные вещи.

И соответственно, в случае, вообще, Stateful application. И в особенности баз данных data storage, нам важно четко определять количество запущенных контейнеров, мы не можем просто полагаться на то, что Kubernetes об этом думает.

Поскольку Stateful set-ы применяются для deploy баз данных, то тут особо не расскажешь, при чем тут Go. И единственное, что я у многих баз данных встречал exporter метрик и чеков. То ест probe тех самых. Как правило, пишутся…, не как правило, но их много довольно написано на Go. Есть эти exporter-ы для Postgres-а. Там есть, вообще, такой вот (НРЗ 05:24), который в себя включает много обвязки на Go. Есть exporter-ы для Mongo-и (НРЗ 05:31) для Go. И так далее. То есть, здесь можно как-то натянуть (НРЗ 05:36), применить Go для всяких exporter-ов, метрик и чеков.

И давайте разберемся со Stateful set-ом. Здесь, как мы видим, у нас в одном файлике 2 ингредиента. Один из них – сам Stateful set, другой из них – сервис. Сейчас разберемся со Stateful set-ом и вернемся к сервису. Соответственно, Stateful set тут все то же самое – metadata, имя, selector-ы. Соответственно, template, в котором есть label-ы, спецификация, mount-ы, порты, контейнер – здесь ничего необычного. Соответственно, мы указываем 3 реплики, 3 уникальных pod-а будут созданы. То есть они будут знать, что они уникальные. Kubernetes-ы о них

будет знать, что они уникальны. И будет ими управлять по особым принципам. То есть, если у вас, например, у вас упала node-а, содержащая в себе deployment с тремя репликами, у вас может такое случиться, что у вас две реплики терминируются, две работают и еще одна создается. Stateful set, соответственно, не создает реплику, пока не убедится, что…, не создает pod, пока не убедится, что pod, такой же pod мертв. То есть, это рассчитано как раз на стабильную работу и долгую работу со State-ом.

Здесь у нас есть volumeClaimTemplates. Это создает PersistentVolumeClaim, если такой есть VolumeMount контейнера. То есть, если мы говорим, что мы хотим приаттачить какое-то постоянное хранилище, а мы обычно хотим в случае со Stateful set-ом. И соответственно, у нас здесь создаются каждый раз новые какие-то Mount-ы. И, поскольку это nginx, это сетевой слой, мы создаем к нему так называемый Headless Service. То есть, мы видим, что сервис с метаданными, который работает с nginx-ами. То есть у нас поднимается label nginx, к ним привязывается сервис. У сервиса есть port: 80. Он знает, по какому port-у он работает. Но у сервиса нет кластера Пи, а это значит, что, когда вы…, идет какой-то запрос извне на этот сервис. Я просто поясню. Сервис – это сетевой концепт, это указание kube proxy, к так называемому сетевому уровню на то, чтобы вы…, куда отрезеректить запрос, какой-то конкретный. Вы можете указать port-ы, вы можете указать динамический диапазон IP-адресов. И указать в вашем сервисе, куда это все должно ссылаться.

Соответственно, сервис делает Форвардинг, потому что обычно снаружи Kubernetes-а есть только один IP-адрес – IP-адрес машины, на которой висит сам Kubernetes, а внутри вот эти вот кластеры IP которые, они все закрыты, вы не можете к ним напрямую обратиться, для этого вам нужен как раз сервис. Опять же, у различных облачных провайдеров существуют всякие различные свои (НРЗ 08:33), которым нужно специфически указывать сервис, либо которые вообще без

сервиса работают, которым можно указать метки, метки приложений. То есть вот это вот вообще избежать. У них там свой механизм форвардинга. Но, как правило, такой стандартный путь у нас есть. И, соответственно, у нас стоит Stateful set, наш сервис обслуживает наш Stateful set.

Давайте посмотрим, как это выглядит в задеплоином варианте, в Kubernetes-е самом. Начнем мы наш путь с описания Stateful set-а, который называется web, вот он, тут. И здесь, что мы видим, у нас, значит, namespace, наши реплики. Три нужно, три есть. Опять же update будет один, заодно и выключает реплики и заменять их на более свежие версии. Опять же, вот эти контейнеры и все такое. И здесь мы видим, report от контролера. То, что контролер, значит, у нас, создал Claim. И, значит, создал pod. Claim pod, Claim pod.

Соответственно, Claim – это, как раз, создание наших жестких дисков, создание наших монтированных папок. Если мы сейчас получим pod-ы, мы увидим, что у нас 3 их, видим, что они…, у них ординальные, ординальный порядок. То есть, у них имена 0, 1, 2 и так далее, что идентифицирует каждый из запущенных pod-ов. И есть еще такой ресурс PersistentVolumeClaim. То есть диски, которые привязаны к этим pod-ам. Опять же, очень важно, потому что они выделяются на системе. И в случае перезагрузки pod-а, они снова будут…, тот же самый диск снова примонтирован к нему. Это важно, потому что таким образом Kubernetes сохраняет State этих самых дисков.

Что еще интересного… Давайте посмотрим на сервис. Мы, опять же, видим сервис. Тут в Kubernetes-е сервис есть кластер IP, а у nginx-а нет Кластера IP. Поскольку сервис предназначен для того, чтобы агрегировать группу pod-ов, у них будет селектор. Сервис – это, вообще, логическое ограничение группы pod-ов для того, чтобы, например, у вас есть 3 реплики вашего приложения, у каждой есть уникальный IP-шник. И вам нужно как-то работать с этими тремя репликами, но не

к каждой обращаться по IP-шнику, а как-то вот единым путем получить. Вы его просто указываете селектор в сервисе. И сервис сам об этом, соответственно, позаботится. У него вот такие Endpoint-ы. То есть эти Endpoint-ы указывают как раз на контейнеры внутри этого сервиса. Если мы сейчас, например, опишем pod web-0, то мы увидим его IP. И соответственно, его открытый порт, который мы описали как раз в описании сервиса. Таким образом, сервис знает про порты, и можно работать с группой pod-ов, как с одним каким-то унифицированным сервисом. И соответственно, здесь мы тоже видим нашу картину с pod-ами, которые имеют каждый порядковый номер, и каждый из них имеет подмонтированные диски. И все это было сделано контроллером Stateful set-а.