Следующая часть нашего Марлезонского балета – это вид deployment-а под названием – Daemon Set. Значит, что такое Daemon Set? У вас есть приложение, и вы хотите, чтобы приложение запускалось на каждой node-е вашего Kubernetes-кого (НРЗ 00:17). То есть на каждом вот в этом вашем физическом сервере. Соответственно, вы объявляете Daemon Set с вашим приложением. И Kubernetes берёт на себя ответственность за запуск на каждой node-е одного Daemon-а.

Механизм Daemon Set довольно специфичный, то есть Kubernetes нам говори, что вот мол ребята, смотрите, вот у нас есть железки, всякие сервера, это всё одно пространство – как один большой мега-сервер, и вообще работайте, не парьтесь. Ну, конечно, это всё реклама, мы понимаем, что у каждой node-ы может быть свой процессор, своя сетевой скорость, свои какие-то метрики. Эти метрики надо экспортировать, вы должны их знать, вы должны понимать, какое здоровье у самого сервера. И нет ничего лучше, чем какое-то, соответственно, приложение, которое запущено непосредственно к нашему серверу.

Прокси процессы – для того, чтобы эти сервера могли вместе связываться и для всяческих авторизаций этих серверов, если у вас защищённая сеть, а также для разделения сетей. Они тоже должны где-то жить. Лучше всего если они будут жить на node-ах, потому что вы точно знаете, что вот на этой node-е у вас всё покрыто этим Daemon Set-ом.

И последнее – это, соответственно, инфраструктурная часть. Это часть, для которой важно – задеплоены вы на какой-то конкретной node-е или нет. Как правило, для клиентов это не столь важно, клиентам важно напихать своих приложений куда-то там в кластер, где они там лежат, на какой node-е, клиентов волновать не должно, должно волновать сисадминов. И вот как бы этот водораздел, вы можете управлять гибко своими структурными приложениями с

помощью Daemon Set. И в общем-то, вообще закрыть доступ клиентам к этой части, Kubernetes-кой API. И просто ими пользоваться самостоятельно.

Мой любимый раздел. Daemon Set – причём здесь Go? Для чего надо использовать. Естественно всё, кто, ещё раз, я буду давать Disclaimer-ы, наверное, на каждом пункте. Прошу это воспринимать, как мое личное мнение на этот счёт и так называемой практики. Это не значит, что всё вы там пишете, у вас Daemon Set написаны на Python. И вы взяли, все переписали на Go. Просто это, соответственно, мнение. И это мнение подтверждено моим опытом работы. Соответственно, во-первых, множество экспортёров метрик для самих node уже существуют, они написаны на GO. Строго говоря, как правило, для каких-то стандартных задач, там, deploy-серверов и запуска node, вам писать что-то, вот именно писать, (HP3 02:27) не потребуется. Как правило, у вас будут там готовые уже exporter-ы на все случаи жизни. Возможно, что-то предоставит vendor. И вам нужно будет только оттуда взять метрики в формате. И положить как-то там себе в базу, чтобы их разобрать. Но, если вам там надо что-то подкрутить, соответственно, как правило, ну их очень много этих exporter-ов. И они…, очень многие из них, написаны на Go. Это довольно удобно если вам нужно разобраться как он работает, а вы уже берете и знаете Go.

Соответственно, мой любимый аргумент, я его буду приводить в каждом пункте. Опять же, из-за того, что мы можем скомпилировать ваши приложения, у нас минимум overhead-а на экспорт, минимум overhead-а на вот эти вот контейнеры, которые экспортируют метрики. Нам нужно только выделить ресурсы, по сути, на то, чтобы собрать эти метрики и куда-то передать наверх. То есть это очень довольно маленький пул ресурсов. И большинство ресурсов кластера у вас остается для клиентских приложений. Ну и, если вы работаете с Kubernetes-ом, нативный Kubernetes API клиент – существует в Go. Он существует, на самом

деле, почти для всех языков. Ну, он довольно в Go удобный. И можно им пользоваться.

Давайте посмотрим на пример Daemon Set-а. Здесь мы уже, в принципе, видим, что это apps, у него отличается kind. Опять же, просто Daemon Set. Здесь, соответственно, мы можем указать, например, namespace – это не только верно для Daemon Set-а, а мы можем, в принципе, в metadata-е указать namespace для того, чтобы не писать Kubernetes apply -n namespace -f file – просто можем указать namespace прямо в файле, он загрузится туда, куда надо.

Соответственно, спецификация. Здесь у нас selector, здесь у нас template. В template-е мы, опять же, указываем Label-ы для имени. Указаны спецификации контейнеров. Ничего особенного. Мы для демонстрации Daemon Set будем устанавливать fluentd. Это логирующий слой для Kubernetes, Kubernetes по умолчанию. Мы помним, там мы делали (НРЗ 05:00), они выдавали нам логи с stdout. У Kubernetes-а нет никаких механизмов для того, чтобы эти логи, например, хранить где-то, чтобы по ним можно было поискать через неделю, через три. Логи очень эфемерны. Pod рухнул – логи стерлись, pod перезагрузился – логи стерлись. Логи стерлись через какое-то время, тем небольшой буфер есть для хранения логов, ну и так далее.

В общем, это плохой механизм для того, чтобы какую-то в нем дебажную информацию хранить. И уж тем более по ней искать. Поэтому есть вот такой fluentd – это универсальный логирующий уровень, он нужен для того, чтобы собирать логи с системы, с контейнеров, с stdout. И куда-то их переправлять, например, в Elasticsearch – в базу данных для хранения, или там какую-либо другую базу данных, ну, в общем, как-то с ними работать. Опять же, в рамках этой лекции я просто показываю, как работают механизмы Kubernetes, мы не

останавливаться подробно, к сожалению, на тулзах и утилитах, необходимых для полноценной enterprise работы Kubernetes-а.

Соответственно, у контейнера существуют ресурсы, здесь мы видим, первый раз – лимиты, то есть максимальное количество ресурсов нужное контейнеру, которое может контейнер использовать, в данном случае мы ограничиваем память 200 МБ.

И request-ы. Request – это необходимое количество ресурсов для запуска контейнера нужное scheduler-у, чтобы найти место на node-е. То есть, если вы указываете request-ы, что CPU должно быть 100m, а memory должны быть 20 мегабайт – то scheduler будет искать node-у, у которой есть свободные ресурсы в этом объёме и на неё, значит, повесит pod. Если, значит, нет такой node-ы, у вас она повиснет в scheduling kube, она будет…, scheduler, значит, занесет ее в свой список и будет ждать пока какие-то ресурсы освободятся.

Значит, после этого…, у нас есть Mount-ы. Мы их монтируем, значит, по определенным путям. Mount-ы у нас создают пути внутри контейнера, в который мы их монтируем. И соответственно, Mount-ы должны совпадать с volume-ами – их имена. И соответственно, в volume-ах есть свой путь на целевой машине. Соответственно, если вы используете какие-то облачные решения, у вас в качестве Mount-ов могут выступать контейнеры S3. Обычно есть…, существуют специальные там, различные компоненты, которые вы может добавлять в ваши файлы-описания, которые характерны для отдельного облачного хостинга, но в целом, это, плюс-минус, выглядит как-то так.

Из последнего, из интересного, что ещё здесь есть – TerminationGracePeriodSeconds – это как раз одна из настроек Kubernetes-а, которая его отличает этот Docker-а, как будто все другое не отличает.

Соответственно, она отвечает за то, что Kubernetes с командой (НРЗ 08:08) завершится. Эта настройка будет…, он послал ему сигнал, соответственно, приложение начнет завершаться. Тут сколько секунд мы ждём, пока оно не завершится. Если сессия не завершается в течение этого времени – Kubernetes просто его убьёт с 9.

Давайте запустим этот файлик и посмотрим, как мы видим этот Daemon Set. Начинаем с того, что получаем, значит, kube-system, существующий Daemon Set. Опять же, тут…, как бы API полностью друг друга повторяют, это всё аналогичной командой get daemonset – и видим, что уже у нас один set есть, который создан Kubernetes-ом по умолчанию. Да, надо бы еще, неплохо бы постоянно бы указывать, что же мы хотим получить. Соответственно, здесь мы видим желаемое количество node, текущее количество node, которое заскейджиалено. Это зависит от, непосредственно от числа node – если у вас 15 node, их у вас будет 15, если там одна, как у меня, потому что это локальный Kubernetes кластер – он заскейджилил одну.

Дальше, соответственно, здесь вы видите все описания, environment, переменные окружения, Mount-ы. Ну то есть все, как обычно. Вернее, не всё, как обычно, а всё, как и должно быть, все эти параметры описания.

Давайте теперь сделаем apply на наш DaemonSet. Итак, мы видим, что, значит, DaemonSet created. Мы можем, опять же, теперь получить DaemonSet из kube-system. Мы видим второй DaemonSet – значит, мы можем теперь его описать. И увидеть, что все наши, во-первых, node-ы аскейджилины, у нас теперь все работает. У нас up-to-date pod-ы. То есть, если какие-то обновления приходят, то у вас эти данные могут разниться. Никаких pod-ов, которых там по два на node-у. Или node, на которых нет pod-ов, у нас тоже нет. Наши, соответственно, Template-ы. То есть это не те pod-ы, которые у вас запущены. Это требования для

pod-а, который будет запущен. И наши, соответственно, Mount-ы и условия. То есть наш контролер. Наш контроллер DaemonSe-at проанализировал всё, что здесь есть, всю эту информацию, и создал наш pod.

Теперь мы можем, соответственно, зайти в kube-system. И посмотреть на наши pod-ы. Вот он наш pod, у нас всё работает, если бы у нас было несколько node, опять же, было бы несколько pod-ов. В принципе, как-то так, опять же, в (HP3 10:57) это всё точно так же, как и со всеми остальными deployment-ами. То есть, удалив pod, не удалив DaemonSet – pod будет заскейджен заново.

И соответственно, таким образом мы обеспечивая стабильную работу нашей системы. Ну, вернее, стабильный Recovery от нагрузок. Естественно, там, если у вас есть какие-то логи, то при этом у вас…, ну вот, видим, второй под поднялся. Если у вас есть какие-то логи – то, естественно, в момент перезагрузки pod-а, они будут потеряны.

Вот как-то так. То есть понять уже, я думаю, паттерн, deployment, DaemonSet. Они просто…, они делают, в принципе, одно и тоже, то есть – их структура похожа. Просто отличаются пути, которыми они появляются, которыми заскейджилятся pod-ы. Если deployment, например, ему всё равно какое количество, вернее, всё равно где размещаются pod-ы. DaemonSet не всё равно, он размещает pod-ы по n