Про структуру Kubernetes-а я рассказал, про api я рассказал. А теперь попробуем вообще понять, зачем это нужно и что с этим делать?

В Kubernetes-е… Соответственно, если Kubernetes рассматривать, как очень простую систему для решения ваших задач, там есть, соответственно, такая штука, которая называется deployment. Которая описывает так же, как вы в Docker-контейнерах описываете, что у вас, значит, есть какой-то image, у него есть какие-то environment variables, какие-то настройки окружения, у него какие-то ресурсы используемые. Вот точно также вы можете это описать в Kubernetes. За исключением того, что, как правило, вы описываете им уже готовый собранный Docker-контейнер. И просто вы пишите несколько таких deployment-ов, он этим самым поднимает эти pod-ы. В pod-ы кладет в каждый pod по одному контейнеру. Это и будет ваш deployment. Запускает ваше приложение. Вы еще прописываете какие порты открыть. И вообще все прекрасно работает.

Но Kubernetes позволяет намного больше. Он позволяет управлять размещением этих pod-ов. Он позволяет, например, вам сказать, что я хочу, чтобы у меня приложение было по одному экземпляру на всех Node-ах моей сети. Или мое приложение имело бы…, перезагружалось бы, если у него что-то отвалится, я хочу определить, как это отвалится. В общем… Короче, существуют различные методы расширения функциональности вашего загруженного приложения, которые называются паттернами. Это определение придумал не я, его придумал Роланд Хасс – инженер Red Hat, который, соответственно, принимал участие, скажем так, в разработке Toolkit-а Kubernetes-а. И он написал, соответственно, книжку. Называется она «Kubernetes patterns». Мы оттуда, в принципе, возьмем его терминологию, эта же терминология в Kubernetes-е встречается. И рассмотрим несколько паттернов Kubernetes-а.

Мы рассмотрим, что такое Health probe/Liveness probe, periodic job, daemon service, stateful service, service discovery и как оно связано с service mesh, что такое sidecar и что такое controller. Естественно, этот список вообще не полный. Я взял основные паттерны, чтобы проиллюстрировать взаимодействие с Kubernetes-ом. И заодно попытаемся вообще понять, а причем здесь вообще golang.

После установки и поднимания вашего основного Kubernetes-кого кластера… Опять же, в моем случае Docker скажет, что кластер поднят. В вашем случае, если вы там, где-то в (НРЗ 02:40) или в Google Cloud-е у вас появится, значит…, там у вас будет контрольная панель. И в ней появится табличка о том, что кластер создан. А вы получите возможность, обращаться к кластеру с помощью команды kubectl, которая является клиентом для того самого api Kubernetes.

Соответственно, к kubectl куча команд. Базовые команды включают в себя: создать, показать, запустить какой-то image, какие-то настроить специфические фичи на объектах. Мы этими командами пользоваться не будем. Они для совсем ручного ввода. Нас будет интересовать в рамках этой лекции команда очень сильно Get. Опять же, команда Edit и Delete в какой-то степени. Я просто покажу, как они работают. И самая главная команда Apply. Команда Apply позволяет вам декларативно объявить ресурсы Kubernetes-а в одном файле и загрузить их, соответственно, в api. И api уже разберется, что с ними делать. Сама выставит все create-ы, поставит все свойства объектам. И поднимет Kubernetes-кий кластер. Ничем остальным мы особо пользоваться не будем. Возможно, будем пользоваться log-ами и exec-ком. Log-и нужны чтобы ввести log-и, а exec нам нужен для того, чтобы нам войти внутрь контейнера и посмотреть, что там происходит.

Соответственно, разберемся немножко с командой get. И пойдем дальше. Kubectl так же, как и Kubernetes, оперирует понятием ресурса. Все, что создается внутри

Kubernetes-а – pod-ы, job-ы, все, что мы будем разбирать дальше, в принципе, является Kubernetes-ким ресурсом. И команда kubectl get позволяет вам получить любой ресурс по его имени.

Соответственно, начнем мы с ресурса namespace. И получаем namespace. Что такое namespace? Namespace – это наименованная область внутри Kubernetes-а, которая ограничивает доступ и ограничивает область видимости и взаимодействия какого-то конкретного набора pod-ов. То есть какого-то конкретного вашего логического unit-а. То есть вашего приложения, так скажем. Если у вас есть какое-то большое приложение, состоящее из многих pod-ов. И вы можете создать под него namespace с именем вашего приложения и туда засунуть все, что к нему относится. И отдельно ограничить к нему права доступа, чтобы, например, пользователи других приложений, задеплоенных в том же кластере, не могли бы к нему добраться.

Так вот, мы получаем namespace. И видим, что по умолчанию Kubernetes создает 4 namespace-а. Соответственно, kube-system – это namespace, который в себе содержит pod-ы, относящиеся к самому Kubernetes-у. Мы можем посмотреть на pod-ы. И воспользоваться командой get с параметром -n. -n – это, соответственно, наш namespace, и получить там pod-ы. Прошу прощения. То есть надо здесь где-то убрать. Соответственно, мы получаем pod-ы. У них есть готовность: сколько pod-ов живо, сколько pod-ов готово. Мы об этом еще поговорим. Сколько раз они перезагружались. И соответственно, их актуальный статус запуска и актуальный возраст. Как вы видите, кластер я поднял не так давно.

Соответственно, если вы не указываете -n и получаете pod-ы, то есть pod-ы – это тоже ресурс внутри Kubernetes-а, вы по умолчанию получаете pod-ы из имени default. Pod-ы из имени default зарезервированы под какие-то ваши эксперименты. Соответственно, это ваш такой умолчальный кластер. Вы можете создать какой-то

другой кластер. И все туда, например, складывать. Соответственно, команда namesape kube-public – это у нас публичное место без каких-либо авторизаций, зарезервировано под какие-либо ресурсы, которые должны быть доступны всем пользователям в кластере. Если у вас, например, есть какие-то внешние серверы, которые должны быть доступны всем пользователям вашего кластера, то endpoint этих сервисов будет лежать в kube-public скорее всего.

И соответственно, последнее kube-node-lease содержит в себе Node-ы Kubernetes-а, которые нужно проверять на предмет их здоровья. То есть? если у вас несколько машин, которые соединены с помощью Kubernetes-а. Kubernetes должен об этом знать и как-то их периодически опрашивать в духе – жив ты или уже умер, стоит ли тебя отключать или стоить ли иметь тебя ввиду, когда ты какие-то pod-ы туда закладываешь. Соответственно, node-lease это место для хранения всех этих Node, которые называются lease. Соответственно, в принципе, мы более-менее разобрались с командой get, то есть мы будем потихоньку смотреть на различные виды ресурсов, которые появляются в Kubernetes-е с помощью команды get. И будем уже разбираться непосредственно в каждом отдельном паттерне отдельно.

Надеюсь, это более-менее понятно. Я буду к этому возвращаться и описывать, соответственно, что вы видите. Давайте уже приступим к паттернам.