K8S 68: Kubernetes. Workload. Serverless

Описание:

В данном задании мы обсудим с вами такой вид архитектуры приложений, как Serverless. Serverless - это такой подход к разработке приложений, при котором не требуется настраивать окружение под каждое приложение. А приложения являются, по сути, простыми функциями, которые заточены под выполнение одной конкретной задачи и запускаются только в тот момент, когда происходит обращение к конкретной функции. Данный подход позволяет утилизировать ресурсы даже более эффективно, чем с контейнерами, а заодно позволяет гибко горизонтально масштабироваться, поскольку на каждый запрос у нас запускается личный обработчик, который быстро выполняет задачу (ну или не очень быстро, но и не очень часто), после чего заканчивает свою работу, освобождая ресурсы под последующие запросы.
Как правило, такой подход связан с 3 понятиями:

1. Функция - собственно, тело функции, которая должна отработать.
2. Событие - условие или действие, при котором функция вызывается.
3. Окружение - то, в чем запускается функция. Если проводить аналогию с Docker, то тот образ, в котором должно быть запущено приложение (как было сказано ранее, у каждого облачного провайдера - свой набор).

Наиболее известной общедоступной платформой, реализующий подход Serverless (можно также говорить, что первыми начали предоставлять услугу FasS - Functions as a Service), является сервис компании Amazon Amazon Web Services Lambda. После ее успеха практически каждый облачный провайдер с целью конкурентоспособности реализовал у себя подобный сервис с небольшими отличиями (к примеру, AWS поддерживает разработку на C#, остальные же провайдеры этим похвастаться не могут).
Интересно сравнить микросервисную и Serverless инфраструктуру - чем они похожи, в чем различны:

Microservice	**Serverless**
Запущен постоянно	Запускается только в тот момент, когда к ней происходит обращение

Как правило, обрабатывает множество запросов в ходе своей работы	Обрабатывает только один запрос, после чего заканчивает свою работу
Может быть как с сохранением состояния (stateful), так и без (stateless)	Только stateless
Реализация возможна на любом языке программирования, так как запускается в каком-то окружении (VM/контейнер/...)	Реализация возможна на языках программирования, которые поддерживает конечная платформа
Стоимость работы напрямую связана с ресурсами, которые требуются для работы	Стоимость, как правило, связана только с временем работы единичного экземляра, умноженным на количество запросов к функции

Как видно, у каждого подхода есть свои особенности работы, которые нужно учитывать при выборе реализации конкретного микросервиса. Однако, опять же, не обязательно выбирать только один поход в рамках общей архитектуры конечного продукта!
Рассмотрим пример использования serverless архитектуры, и как ее можно интегрировать в имеющуюся микросервисную архитектуру:

- Фоновые задачи по расписанию - те же cron-задачи, когда изменения кода производятся часто, а под каждый скрипт создавать образ кажется оверхедом.
- Задачи по запросам извне - скажем, обработка аналитики, отправляемой из браузера клиента, где задача - принять данные и положить в БД или очередь на обработку.
- Обработка ивентов из соседних приложений - можно строить pipeline по функциям, где по выполнению одной задачи у нас вызывается другая и так далее, полная обработка входных данных проходит за несколько функций.
- Преобразование данных - когда на входе у нас имеются одни данные, а сервис далее требует другой формат данных.

Разработка при такой архитектуре проекта, естественно, отличается от разработки обычного сервиса, где в рамках одного сервиса большое количество обработчиков - функций, классов, обработчиков фоновых задач, систем отправки метрик и прочих радостей жизни. Все, что у нас есть у конкретной функции, - одна задача, которую она обрабатывает при запросе и погибает. Ни более, ни менее.

Как ни смешно, но корневая идея разработки такая же, как и в любом языке программирования, - можно реализовывать все самому на своем языке программирования с костылями, велосипедами и прочими радостями жизни курильщика (переводя в реалии serverless - затачиваться под определенного провайдера с его API), либо использовать фреймворки, которые под капотом уже решили те проблемы, над которыми вы могли бы мучиться.
Так, наиболее известным фреймворком FaaS для работы с облачными провайдерами (и не только) является фреймворк Serverless, который сводит обращение к разным провайдерам до разных шаблонов.
А теперь к самому вкусному - как вообще Serverless связан с Kubernetes? Ответ довольно прост - если есть спрос, то родится и предложение. Так и получилось - была необходимость иметь FaaS, но при этом не иметь Vendor lock и вообще запускать все это в своем Bare metal kubrenetes-кластере (эдакий smoothie enterprise), вот и появились решения (да-да, не одно - спасибо Open Source за это).
И конечно, мы с вами познакомимся с одной из реализаций этого подхода на примере Kubeless - одного из многих решений в Kubernetes, но, в то же время, одного из немногих, которые имеют связку с фреймворком Serverless, который мы будем использовать в рамках нашей задачи.
Данное решение позволяет все, что умеет тот же Lambda (кроме привязки к инфраструктурным зависимостям), но без Vendor Lock, с возможностью расширяемости (к примеру, можно использовать Kafka как источник событий) и практически с любым языком программирования, который вам только в голову приходит (при необходимости, в любой момент можно дописать свой runtime, который будет поддерживать язык программирования, который не поддерживается на данный момент).
Внутри же Kubeless решает вопрос создания функций за счет создания при каждом запросе Pod или Deployment, которые убиваются по окончанию работы.
Но в дебри мы лезть не будем - ограничимся запуском простого приложения, которое срабатывает по HTTP-запросу.

Полезные ссылки:

- Что такое Serverless Архитектура и в чём её преимущества?
- RubyRussia 2019: Николай Сверчков о serverless (habr)
- Основы Serverless приложений в среде Amazon Web Services (habr)
- AWS Lambda
- Google Cloud Functions
- ramitsurana/awesome-kubernetes -- Serverless Implementations (github)
- Serverless framework
- kubeless (github)

Задание:

1. Из инкубатора скачайте последнюю версию helm-чарта kubeless и разархивируйте, чтобы диретория с чартом называлась kubeless.
2. Скопируйте из чарта файл values.yaml в корень репозитория.
3. Запустите у себя в Kubernetes-кластере Kubeless, используя скачанный helm-чарт с именем kubeless в namespace kubeless с запущенным UI интерфейсом и окружением kubeless под функции (команды и вывод сохранить в файле kubeless_install.txt).
4. Установите у себя Kubeless CLI.
5. Выведите список контейнеров в окружении kubeless (команды и вывод сохранить в файле kubeless_containeers_list.txt).
6. Сохраните следующий код:

```
def hello(event, context):
    return "hello world, your data is {}".format(event)
```

в корне репозитория в файле под именем hello.py.

7. Сделайте деплой функции из файла hello.py с работой в Python 3.7 и именем hello-rebrainme при помощи команды kubeless (команды и вывод сохранить в файле deploy_function.txt).
8. Выведите список функций через kubectl и через kubeless (команды и вывод сохранить в файлах kubectl_function_list.txt и kubeless_function_list.txt соответственно).
9. Выведите список deployment в окружении kubeless (команды и вывод сохранить в файле kubeless_deployments_list.txt).
10. Выведите список pods в окружении kubeless (команды и вывод сохранить в файле kubeless_pods_list.txt).
11. Сделайте запрос к функции через kubeless function call и получите вывод работы (команды и вывод сохранить в файле kubeless_function_call.txt).
12. На проверку отправьте репозиторий в GitLab и все сохраненные фрагменты вызова команд и их логов.