ОРКЕСТРАЦИЯ DOCKER КОНТЕЙНЕРОВ: КОНЦЕПЦИЯ ZERO DOWNTIME DEPLOYMENT

Новик И. П.
Кафедра программного обеспечения информационных технологий, Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: nolik03@gmail.com

В данной статье обсуждаются проблемы микросервисной архитектуры и методы их решения путем оркестрации упакованных в Docker контейнеры микросервисов, рассматривается концепция бесшовной доставки новых версий микросервисов.

Введение

В настоящее время все большую популярность набирает подход разработкий приложений на основе микросервисной архитектурами. Вместе с преимуществами данного подхода возникают новые сложности в управлении инфраструктурой проектов. При развертывании новых версий приложения мы непременно сталкиваемся с проблемой простоя, образующегося во время переключения production-сервера. Данные простои очень дорого отражаются на бизнесе клиентов, зачастую мы можем видеть, как популярные сайты делают “заставку” на время технических работ по развертыванию новой версии приложения. За данными заставками могут скрываться как устаревшие техники обновления артефактов на серверах (такие как прямой передеплоймент), так и прямая доставка докер контейнеров на хост. В качестве решения проблемы предлагается применение концепции Zero Downtime Deployment которая позволяет обойти данную проблему.

I. Метод стадартной доставки новой версии приложения

Рассмотрим стандартный метод доставки новой версии приложения с передплеойментом (см. рис. 1).

Рис. 1 – процесс доставки приложения с пролемой простоя

Трафик со старой версии приложения на новую не может переключаться мгновенно: предварительно мы должны убедиться, что новая версия не только успешно выкачена, но и «прогрета» (т.е. полностью готова к обслуживанию запросов).

Таким образом, некоторое время обе версии приложения (старая и новая) будут работать одновременно. Что автоматически приводит к конфликту общих ресурсов [1]: сети, файловой системы, IPC и т.д.

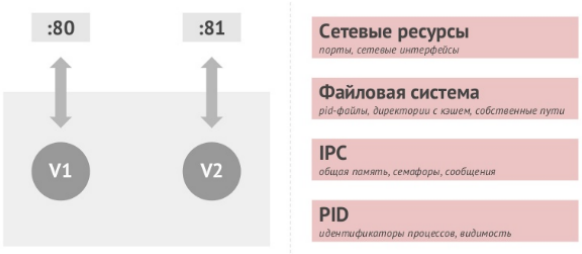

Рис. 2 – конфликты доставки приложения с проблемой простоя

II. Метод доставки новой версии приложения по концепции Zero Downtime Deployment

С использованием Docker проблема простоя решается запуском разных версий приложения в отдельных контейнерах, для которых гарантируется изоляция ресурсов в рамках одного хоста (сервера/виртуальной машины) (см. рис. 3).

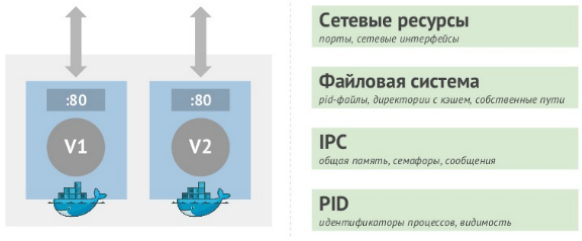

Рис. 3 – изоляция ресурсов путем запуска нескольких контейнеров на одном хосте

Контейнеризация даёт много других плюсов в процессе доставки кода приложения на конечный сервер (deploy). Любое приложение зависит от определенной версии (или диапазона версий) интерпретатора, наличия модулей/расширений и т.п., а также и их версий и относится это не только к исполняемой среде, но и ко всему окружению включая системное программное обеспечение и его версии (вплоть до используемого Linux-дистрибутива). То, что контейнеры содержат не только код приложений, но и предварительно установленное систем-

ное и прикладное программное обеспечение нужных версий, избавляет нас от проблем с зависимостями [2].

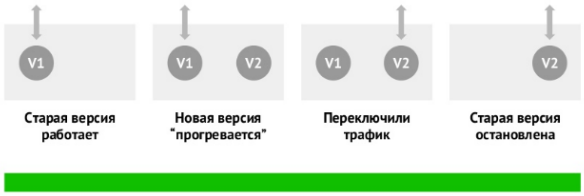

Рис. 4 – процесс доставки приложения по концепции Zero downtime deployment

Общую схему развертывания новых версий с учётом концепции Zero Downtime Deployment можно описать следующим алгоритмом:

1. старая версия приложения работает в первом контейнере;
2. новая версия разворачивается и «прогревается» во втором контейнере, (примечательно, что сама эта новая версия может нести не только обновлённый код приложения, но и любых его зависимостей, а также системных компонентов, например, новую версию OpenSSL или всего дистрибутива);
3. к тому моменту как новая версия полностью готова к обслуживанию запросов, трафик переключается с первого контейнера на второй, и старая версия может быть остановлена.

III. Оркестрация Docker контейнеров при развертывании

Оркестрация - термин, который обозначает распределение контейнеров, управление кластером, и возможность добавления дополнительных хостов. При разделении приложения на отдельные изолированные сервисы, эти сервисы должны восприниматься, как единое целое. В микросервисной архитектуре развёртывание одного сервиса без развёртывания остальных не имеет никакого смысла, поскольку они должны работать сообща.

От инструмента распределения задач, при развертывании микросервисной архитектуры, требуется группировка контейнеров. Управление группами позволяет администратору работать с набором контейнеров, как с единым целым. Запуск и выполнение тесно связанных компонентов, как единого целого, упрощает управление приложением, не жертвуя при этом преимуществами разделения функциональности между отдельными компонентами, что позволяет администраторам пользоваться преимуществами контейнеризации в случае использовании микросервисной архитектуры, минимизирую при этом усилия по управлению системой [3], а также позволяет реализовывать концепцию Zero Downtime Deployment. В ходе проведения эксперимента предлагается использовать оркестратор контейнеров Kubernetes. В ходе эксперимента реализовывается концепция развертывания на оркестраторе и применяется стратегия обновления RollingUpdate, при которой контейнеры будут обновляться плавно, по очереди.

IV. Заключение

В работе рассмотрены методы доставки новых версий приложения. Предложен алгоритм развертывания с учетом концепции Zero Downtime Deployment. Оркестратором для развертрывания с учетом концепции выбрана платформа Kubernetes с применением стратегии обновления RollingUpdate.

Список литературы

1. Vikram, M. Microservices Deployment Cookbook / M. Vikram // PacktPublishing, 2017. – P. 129.
2. Fisher, L. M. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise (2nd Edition) / M. Vikram // Addison-Wesley Professional, 2015. – P. 229.
3. Randall, S. Docker Orchestration / S. Randall // PacktPublishing, 2017. – P. 84.