УДК 004.4

Терских М.Г.
специалист по разработке распределённых вычислительных систем компании "EXON LV"
студент магистратуры 1 курса
факультет «Информатики и систем управления»
МГТУ им. Н.Э.Баумана
Россия, г. Москва

ТЕХНОЛОГИИ ИЗОЛЯЦИИ ПРИЛОЖЕНИЙ И ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ДЛЯ УПРАВЛЕНИЯ КОНТЕЙНЕРАМИ

Аннотация:

В статье рассматриваются две технологии для обеспечения изоляции приложений. Показано сравнение этих технологий. Приведены инструментальные средства оркестрации контейнеров. Описаны ключевые возможности Docker Swarm, Mesosphere Marathon и Kubernetes. Приведены примеры использования и даны рекомендации.

Ключевые слова: контейнеризация, виртуализация, докер, кубернетес, мезосфер марафон.

Terskikh M.G.
specialist in the development of distributed computing systems of the "EXON LV" company
student of the Master's program of «Informatics and management systems»
Bauman Moscow State Technical University
Russia, Moscow

TECHNOLOGIES OF ISOLATION OF APPLICATIONS AND INSTRUMENTAL MEANS FOR CONTAINERS ORCHESTRATION

Abstract:

The article is devoted to two technologies for securing application isolation. A comparison of these technologies is shown. The tools of container orchestration are given. The key features of Docker Swarm, Mesosphere Marathon and Kubernetes are described. Examples of use are shown and some recommendations are given.

Keywords: containerization, Virtualization, docker, kubernetes, mesosphere marathon.

Изоляция приложений

Мир информационных технологий все больше использует контейнерные технологии. Docker контейнеры стали основной темой

разговоров при обсуждении серверной инфраструктуры и DevOps. Набор инструментов Docker включает в себя движок для управления контейнерами, многоуровневую файловую систему и публично размещенный реестр образов широкоиспользуемых приложений. Как правило, движок Docker автоматизирует развертывание приложений внутри контейнеров Linux [1].

Механизмы ядра Linux, обеспечивающие изоляцию процессов друг от друга, такие как, cgroups , namespaces и chroot , позволяют группам процессов иметь частный вид системы и работать независимо друг от друга на одном хосте Linux. Эта архитектура позволяет избегать непроизводительных издержек при запуске и на уже работающих виртуальных машинах (ВМ). Так как виртуальные машины работают на ОС хоста, каждое новое приложение должно запускать гостевую ОС. Это не очень хорошо масштабируется для микро-сервисных архитектур, где сервисы занимают меньше пространства, а их количество больше. Docker просто запускает приложения как группу процессов, которые остаются разделенными друг от друга, при этом базовые вычислительные ресурсы используются совместно.

Сравнивая контейнеры Linux с виртуальными машинами (ВМ) можно заметить, что последние требуют больше ресурсов, но обеспечивают лучшую изоляцию, так как гипервизоры виртуальных машин эмулируют всю операционную систему, включая виртуальное аппаратное обеспечение. С другой стороны, контейнеры работают на общей операционной системе. Контейнеры, использующие миниатюрные пакеты, в которых содержатся приложение и зависимые от него объекты, оставляют позади большую часть громоздких ВМ (размеры ВМ часто могут достигать десятков гигабайт). В результате хост Docker может запускать гораздо больше приложений на идентичном аппаратном обеспечении. На рисунке 1 показана архитектура обеих технологий.

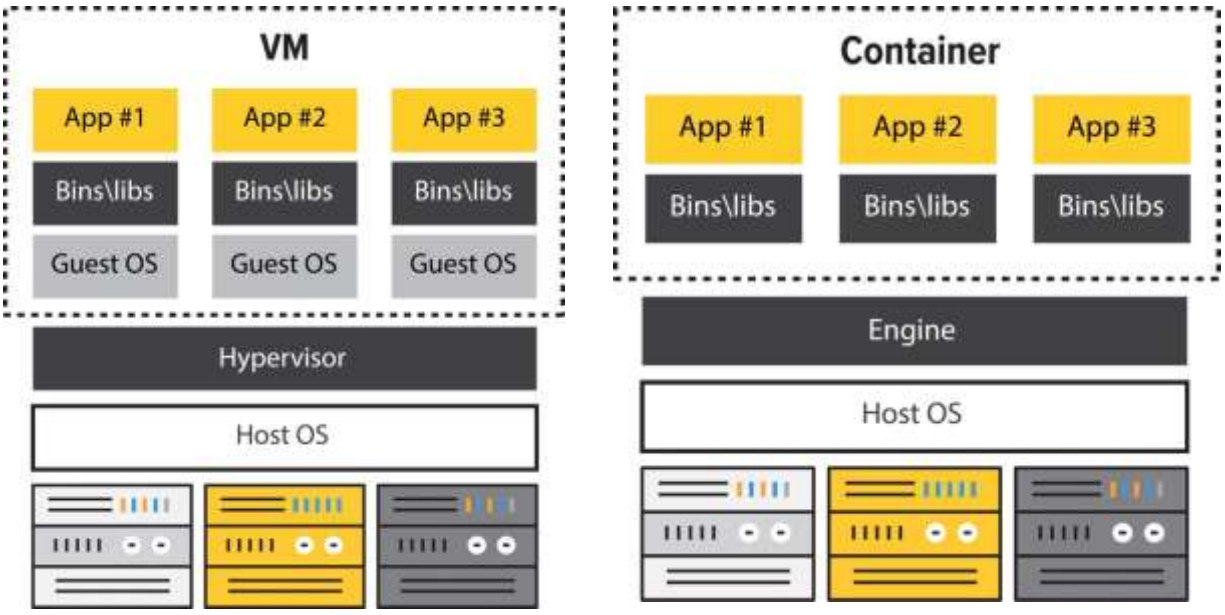

Рисунок 1 – отличие виртуальной машины от контейнера

Кроме того, важным различием между контейнерами и виртуальными машинами является скорость исполнения. Запуск виртуальной машины может занимать несколько минут и часто влечет за собой интенсивное использование ресурсов. Таким образом, запуск нескольких виртуальных машин требует определенного планирования и очередности. В отличие от ВМ, несколько контейнеров, благодаря их легковесности, можно запустить в течение нескольких секунд. Такие характеристики и масштабируемость приводит к новому типу, может быть, даже новому поколению, распределенных приложений, в которых контейнеры автоматически запускаются и останавливаются в зависимости от различных факторов, таких как пользовательский трафик, события, запросы и запланированные задачи.

Инструменты оркестрации

Итак, контейнеры и легкий способ виртуализации приложений - важный элемент любого плана DevOps. Но для организации контейнеров нужны программы для их управления. К таким программам относятся Kubernetes, MesosphereMarathon и DockerSwarm.

DockerSwarm

Как правило, знакомство с контейнерами начинается с Docker, первой контейнерной программы, которая привлекла большую аудиторию пользователей. Естественный инстинкт подталкивает обратиться к программе по управлению контейнерами, созданной теми же людьми, которые проектировали инфраструктуру контейнера. Часто такой программой становится Docker Swarm. Swarm– техническим языком Docker Native Orchestration [2].

Для пользователей, хорошо знакомых с Docker, обучение работе в Swarm не вызывает проблем. Docker Native Orchestration использует одноузловую концепцию Docker и расширяет ее до Swarm. После запуска Docker узлов, установка Swarm не вызывает абсолютно никаких трудностей. Например, для добавления экземпляров Docker в кластер Swarm можно использовать одиночную командную оболочку. Обновленный Swarm также включает в себя поддержку для последовательного обновления, межузловое шифрование Transport Layer Security, простую балансировку нагрузки и легкую абстракцию сервиса.

Но, в DockerSwarm еще есть, что улучшать. Например, не поддерживаются проверки состояния контейнеров и автоматические возвраты к предыдущему состоянию. Пользователи Swarm также жалуются на многочисленные проблемы внедрения. Например, многие пользователи жалуются, что Swarm должным образом не поддерживает мульти-хостовую конфигурацию сети контейнера.

Mesosphere Marathon

Marathon является платформой контейнерной оркестровки для DC/OS Mesosphere и ApacheMesos. DC/OS – это распределенная операционная система на базе распределенных систем ядра Mesos. Mesos, в свою очередь, является открытой системой управления кластером. Marathon обеспечивает интеграцию управления между существующими приложениями, сохраняющими состояние, и контейнерными приложениями, не сохраняющими состояние [3].

Marathon имеет множество функций, включая высокую степень доступности, обнаружение сервисов и балансировку нагрузки. Если запустить его на DC/OS, приложения также получит виртуальную IP-маршрутизацию.

Kubernetes

Kubernetes имеет отличную сервисную поддержку со стороны поставщиков. Сегодня основным дистрибьютором Kubernetes выступает промышленный альянс Cloud Native Computing Foundation компании Linux Foundation.

Помимо доказательства своей состоятельности на датацентрах Google, Kubernetes может похвастаться самовосстановлением, автоматизированными последовательными обновлениями и возвратами в исходное состояние, а также оркестровкой хранения. И это далеко не полный список функциональных возможностей Kubernetes.

Большим преимуществом Kubernetes является его способность к взаимодействию. Например, для создания необходимой топологии и инфраструктуры, достаточно написать одну JSON спецификацию и вставить ее в любой запущенный кластер Kubernetes. На рисунке 2 приведены примеры таких JSON файлов, с помощью которых можно создавать Deployment, Pod или Service [4].

Рисунок 2 – примеры JSON файла для запуска компонентов Kubernetes

Еще одно преимущество заключается в том, что "Kubernetes– это облако-агностик",- говорит инженер-программист ЭрезРаби (ErezRabih). "Кластер можно запустить на AWS, GoogleCloud, Microsoft'sAzure, Rackspace и т.д., и он будет работать более или менее одинаково". Действительно, одна из краткосрочных целей разработчиков Kubernetes заключается в том, чтобы позволить запускать контейнеры Docker, управляемые Kubernetes, с помощью мультиоблачной архитектуры, одновременно используя Cluster Federation.

Конечно же, Kubernetes не совершенен. Сразу же на ум приходят два его недостатка. Во-первых, сложность балансировки нагрузки. В конце концов, имея доступ в Kubernetes, можно с легкостью запускать внешний балансировщик нагрузки изнутри Kubernetes, но это все еще на стадии разработки. Во-вторых, Kubernetes отлично справляется с автоматическим фиксированием проблем. Настолько хорошо, что можно даже не заметить повторный запуск контейнера после сбоя. Для решения этой проблемы необходимо добавить централизованную систему протоколирования.

Заключение

Как всегда, выбор зависит от конкретных потребностей пользователя. На данный момент сложно сказать, какое из решений было бы наилучшим.

Абсолютно точно можно сказать то, что, если Вы собираетесь серьезно заниматься контейнерами, Вам потребуются эти программы. С ними Вам не нужно будет тратить время на создание собственных средств по администрированию контейнеров.

Для создания уникального сочетания, учитывая все потребности компании, эти программы можно смешивать и объединять.

Использованные источники:

1. Using Docker: Developing and Deploying Software with Containers 1st Edition, 2016, 354 p
2. Руководство Docker Swarm. URL: https://docs.docker.com/engine/swarm/ (дата обращения: 10.06.2017)
3. Руководство Mesosphere Marathon. URL: https://docs.mesosphere.com (дата обращения: 10.06.2017)
4. Руководство Kubernetes. URL: https://kubernetes.io/docs/home/ (дата обращения: 12.06.2017)