Артамонова Александра Александровна
Artamonova Aleksandra Aleksandrovna
Студент
Student
Национальный исследовательский ядерный университет «МИФИ»
National Research Nuclear University MEPhI

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ДВУХ СИСТЕМ УПРАВЛЕНИЯ КОНТЕЙНЕРАМИ: DOCKER SWARM И KUBERNETES
A COMPARATIVE ANALYSIS OF TWO CONTAINER MANAGEMENT SYSTEMS: DOCKER SWARM AND KUBERNETES

Аннотация на русском языке: Одной из наиболее распространенных технологий виртуализации является контейнерная виртуализация. Для облегчения создания, развертывания, запуска и управления кластером контейнеров были созданы специальные программные решения. В статье приводится сравнительный анализ двух самых распространенных систем управления кластерами контейнеров – Docker Swarm и Kubernetes, рассматриваются их достоинства и недостатки.

The summary in English: One of the most common virtualization technologies is container-based virtualization. To facilitate the creation, deployment, launch and management of the container cluster, special software solutions were created. The article provides a comparative analysis of the two most common container cluster management systems – Docker Swarm and Kubernetes and describes their advantages and disadvantages.

Ключевые слова: *виртуализация, контейнерная виртуализация, Docker, Docker Swarm, Kubernetes*

Key words: virtualization, container-based virtualization, Docker, Docker Swarm, Kubernetes

С развитием компьютерных технологий начал расти и объем данных, которые собираются, накапливаются, обрабатываются и передаются. Из-за огромного роста машинных данных требования к мощностям серверов, анализирующих, управляющих и использующих эти данные, постоянно растет. Возникает необходимость использовать решения, которые помогут эффективно использовать ресурсы и обеспечивать безопасность, эффективную передачу данных и надежность. Для достижения этих целей были созданы технологии виртуализации. Серверная виртуализация стала ожидаемым курсом действий с начала прошлого десятилетия в различных сферах – от предоставления традиционных услуг до постоянно расширяющихся облачных сервисов [1][2]. Вездесущий характер виртуализации также отражает все большее количество ресурсоэффективных

приложений, особенно в мире встраиваемых систем. Она предлагает множество преимуществ в нескольких областях встроенных вычислений, включающих в себя более эффективное использование ЦП, повышенную безопасность, легкую миграцию и снижение затрат.

Основными технологиями виртуализации являются виртуализация на основе использования гипервизоров и контейнерная. Хотя существуют случаи, в которых использование контейнерной виртуализации невозможно, метод гипервизорной виртуализации не лишен своих недостатков [3] – вероятность снижения эффективности и простоя вычислительных ресурсов, сложности при переносе различных приложений между виртуальными операционными системами, возможность потерять все виртуальные машины при выходе из строя одного гипервизора, на котором они установлены и др.

Контейнерная виртуализация использует возможности ядра для создания изолированной среды для процессов. В отличие от виртуализации на основе гипервизоров, контейнеры не получают свое собственное виртуальное оборудование, а используют аппаратное обеспечение хостовой системы. Так как в данном случае не возникает необходимости эмулировать аппаратные средства и загружать полноценную операционную систему, контейнеры могут работать более эффективно, чем гипервизоры.

Одна из особенностей контейнеров состоит в том, что ими можно управлять как кластером инкапсулированных приложений. В качестве примера можно предположить, что мы запустили сотню контейнеров на одном сервере. Эти контейнеры подключены к сети и работают отдельно друг от друга. Однако в таком случае трудно получить доступ к целому набору контейнеров, объединенных общей задачей, по одному. Поэтому появились системы управления контейнерами с удобными API. Две наиболее популярные системы управления контейнерами (или системы оркестровки контейнеров) – это Docker Swarm и Kubernetes. Ниже приводится

сравнительный анализ двух этих систем, рассматриваются их недостатки и преимущества.

Docker Swarm

Docker – это инструмент с открытым исходным кодом, который упрощает создание, развертывание и запуск приложений с помощью контейнеров. Контейнеры Docker позволяют разработчику упаковывать приложение со всеми необходимыми ему компонентами и зависимостями, например, библиотеками, и поставлять его как один пакет. Хотя на текущий момент Docker является одним из самых популярных приложений, предоставляющий подобный функционал, существуют и другие альтернативы – к примеру, можно упомянуть инструмент Rocket, являющийся аналогом Docker.

Docker Swarm – это инструмент кластеризации и планирования Docker контейнеров. С помощью Swarm администраторы и разработчики могут устанавливать кластеры контейнеров и управлять ими. Кластеризация – важная функция контейнерной технологии, поскольку она создает совместную группу систем, с помощью которой можно добиться отказоустойчивости при неполадках с одним или несколькими узлами. Кластер Docker Swarm также предоставляет администраторам и разработчикам возможность добавлять или удалять наборы контейнеров при надобности. Администратор контролирует Swarm через диспетчер, который организует и планирует контейнеры. Менеджер Swarm позволяет пользователю создавать основной экземпляр менеджера и несколько запасных на случай сбоя главного. Пользователь может развертывать управляющие и рабочие узлы во время выполнения.

Docker Swarm использует стандартный программный интерфейс Docker для взаимодействия с другими инструментами, например, Docker Machine (инструмент, позволяющий управлять удаленными хостами Docker с

локального компьютера).

Swarm использует возможности планирования для обеспечения наличия достаточных ресурсов для распределенных контейнеров. Swarm назначает контейнеры базовым узлам и оптимизирует ресурсы, автоматически планируя рабочие нагрузки контейнера для запуска на наиболее подходящем хосте. Эта система уравновешивает рабочие нагрузки контейнерных приложений, обеспечивая запуск контейнеров в системах с ограниченными ресурсами, поддерживая необходимый уровень производительности.

В Swarm есть три механизма, чтобы определить, на каких узлах должен работать каждый контейнер:

Spread – действует как настройка по умолчанию и распределяет контейнеры по узлам в кластере на основе доступных системных ресурсов. Преимущество этого механизма состоит в том, что если узел выходит из строя, теряется только несколько контейнеров.

BinPack – контейнеры распределяются до полной загруженности каждого узла. Когда узел заполнен, контейнеры назначаются следующему объекту в кластере. Преимущество BinPack заключается в том, что он использует меньшее количество инфраструктуры и оставляет больше места для больших контейнеров на неиспользуемых машинах.

Random – выбор узла происходит случайным образом.

Хотя этот инструмент и заслуживает рассмотрения, он обладает рядом существенных недостатков, среди которых:

- Docker Swarm привязан к контейнерам Docker, хотя Docker – всего лишь одно из возможных решений контейнеризации;
- он не является расширяемым, так как был разработан для достаточно узкой сферы деятельности, что упрощает использование, но делает невозможным произвести тонкую настройку системы;

- в Docker Swarm отсутствует возможность автоматического обновления контейнеров.

Kubernetes

Kubernetes – это платформа с открытым исходным кодом, предназначенная для развертывания, масштабирования и управления кластером контейнеров Linux как единой системой на большом количестве хостов.

Проект был начат компанией Google в 2014 году, в который она вложила полтора десятка лет опыта работы с контейнерами [4]. Сейчас Kubernetes имеет активное сообщество и поддерживается такими компаниями, как Microsoft, RedHat, IBM и Docker.

Начиная с его первого выпуска в 2014 году, K8s быстро развивалась с привлечением всего сообщества Open Source, включая Red Hat, VMware и Canonical.

Kubernetes позволяет разработчикам перейти от хост-ориентированной инфраструктуры к контейнерно-ориентированной, что позволит использовать все преимущества и выгоды, присущие контейнерам.

Прежде чем приступать к описанию компонентов Kubernetes, следует привести значение некоторых связанных терминов.

Pod. Kubernetes предназначен для управления гибкими приложениями, состоящими из множества микросервисов, взаимодействующих друг с другом. Часто эти микросервисы образуют группу контейнеров, которые обычно выполняются вместе на одном сервере. Эта группа – наименьшая единица, которая может быть запланирована для развертывания через Kubernetes, называется pod. Эта группа контейнеров будет совместно использовать место на жестком диске, пространства имен Linux, cgroups, IP-адреса. Они совместно используют ресурсы и всегда планируются вместе. Pod-ы не предназначены для длительного использования. Они создаются, уничтожаются и снова создаются по требованию, в зависимости от состояния

сервера и самой службы.

Сервис. Поскольку pod-ы имеют короткий срок службы, нет точных сведений относительно IP-адреса, на котором они обслуживаются. Это может затруднить связь микросервисов. Поэтому в Kubernetes введена концепция сервиса, которая представляет собой абстракцию, объединяющую наборы некоторого числа pod-ов. K8S поддерживает динамическое наименование и балансировку нагрузки pod-ов помощью абстракций, гарантируя прозрачное подключение к сервисам по имени и отслеживая их текущее состояние. Имеется возможность настроить балансировку нагрузки для многочисленных pod-ов.

На Рисунке 1 можно увидеть высокоуровневую архитектуру Kubernetes. Далее будут рассмотрен функционал каждого компонента.

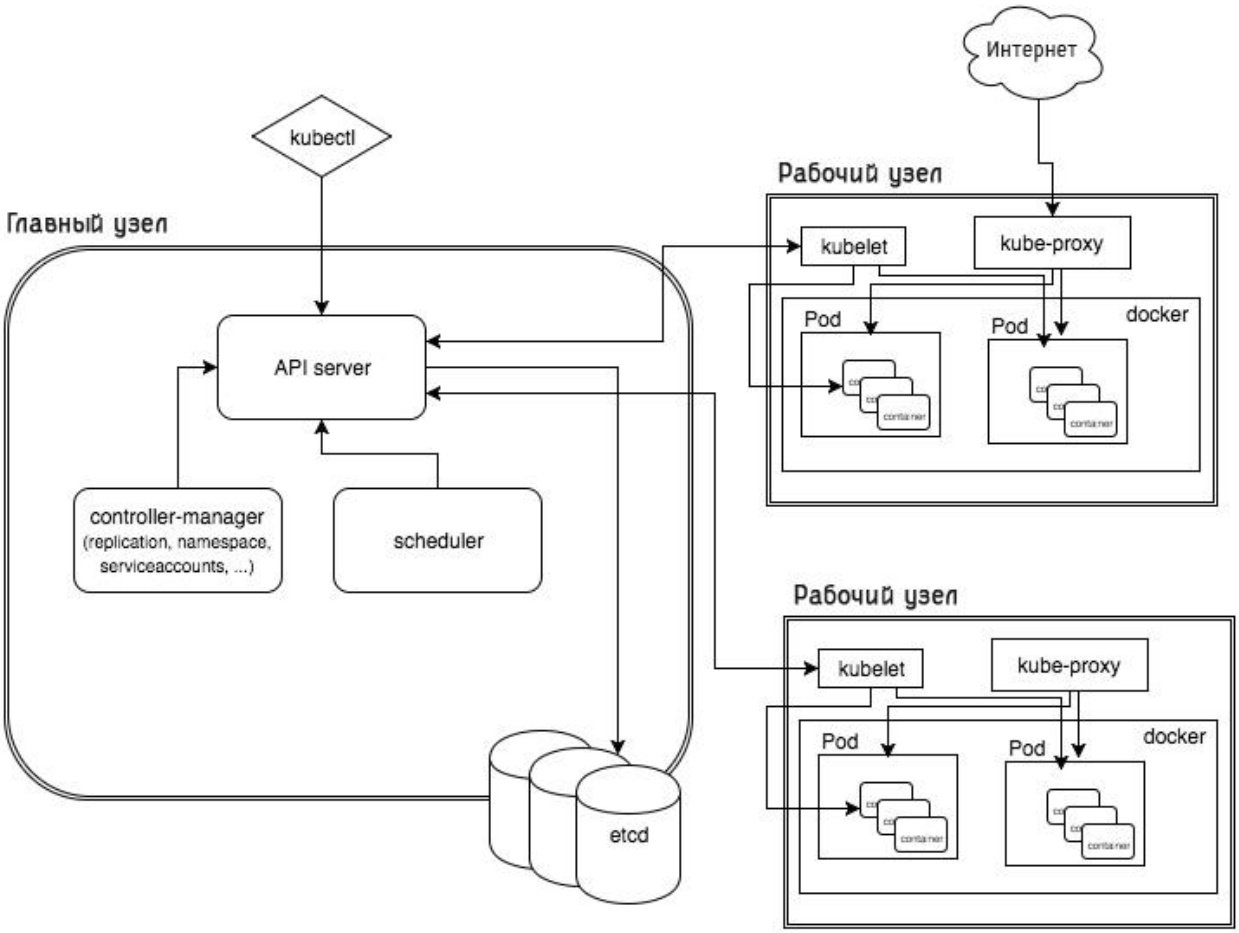

Рисунок 1. Высокоуровневая архитектура Kubernetes

Источник: документация Kubernetes

Главный узел (Master Node). Главный узел отвечает за управление кластером Kubernetes. Это точка входа для всех административных задач. Master Node заботится о том, чтобы организовать рабочие узлы (Worker Node), на которых работают реальные службы.

API-сервер. API-сервер является точкой входа для всех команд REST (англ. Representational State Transfer), используемых для управления кластером. Он обрабатывает rest-запросы, проверяет их и выполняет связанную бизнес-логику. Состояние результата должно где-то сохраняться, и это приводит к следующему компоненту главного узла.

Хранилище etcd (etcd storage). Etcd – это простое, распределенное, согласованное хранилище ключей и значений. Оно используется для совместной настройки и сервисов. Он предоставляет собой REST-API для операций CRUD (акроним, сокр. от англ. create, read, update, delete — «создать, прочесть, обновить, удалить»), а также интерфейс для регистрации наблюдателей на определенных узлах, что дает возможность надежно уведомить остальную часть кластера об изменениях в конфигурации.

Планировщик (sheduler). Развертывание сконфигурированных пакетов и сервисов на узлах происходит с помощью такого компонента Kubernetes, как планировщик. Планировщик обладает информацией о ресурсах, доступных для компьютеров кластера, а также о тех, которые необходимы для запуска настроенного сервиса, и, следовательно, он может решить, где развернуть конкретную службу.

Менеджер контроллера (controller-manager). При желании можно запускать различные типы контроллеров внутри мастер узла. Controller manager является службой, позволяющей делать это. Контроллер использует apiserver для наблюдения за общим состоянием кластера и внесения корректирующих изменений в текущее состояние в соответствии с желаемым. Примером такого контроллера является Replication Contoller,

который заботится о количестве pod-ов в системе. Коэффициент репликации настраивается пользователем, и обязанностью контроллера является запускать или удалять pod-ы, если их количество вдруг станет меньше или больше требуемого.

Рабочий узел (Worker Node). В рабочих узлах выполняются pod-ы, и поэтому они содержат все необходимые службы для управления сетью между контейнерами, связи с главным узлом и назначения запланированных ресурсов контейнерам.

Docker. Докер запускается на каждом из рабочих узлов и запускает сконфигурированные пакеты. Он заботится о загрузке образов и запуске контейнеров.

Kubelet. Kubelet получает конфигурацию модуля из apiserver и гарантирует, что нужные контейнеры запущены и работают. Это рабочая служба, которая отвечает за связь с главным узлом. Kubelet также связывается с etcd, чтобы получать информацию об услугах и записывать информацию о вновь созданных.

Kube-proxy. Kube-proxy действует как proxy-балансировщик нагрузки для сервиса на одном рабочем узле. Он отвечает за маршрутизацию сети для пакетов TCP и UDP.

Kubectl. Kubectl – инструмент командной строки для связи с сервисом API и отправки команд на главный узел.

Kubernetes позволяет быстро и эффективно развертывать приложения, масштабировать их на лету, во время выполнения, встраивать новые функции и использовать только нужные ресурсы, что позволяет оптимизировать использование оборудования. Это дает возможность быстро реагировать на запросы пользователя и поддерживать желаемое состояние.

Также Kubernetes является:

- переносимым – он легко запускается как публичная, частная или гибридная система, а также в облаке;

- расширяемым – встроенные модули и компоненты при желании могут быть заменены альтернативными;
- самовосстанавливающимся – в нем присутствуют такие функции, как автоматические перезагрузка, репликация, размещение, масштабирование и балансировка нагрузки контейнеров.

Литература:

1. A.A. Semnanian, J. Pham, B. Englert, X. Wu. Virtualization Technology and its Impact on Computer Hardware Architecture. In Eighth International Conference on Information Technology: New Generations, ITNG 2011, Las Vegas, Nevada, USA, 2011.

2. K. Sandstrom, A. Vulgarakis, M. Lindgren, T. Nolte. Virtualization Technologies in Embedded Real-Time Systems. In Proceedings of 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation, ETFA 2013, Cagliari, Italy, 2013.

3. J. Daniels. Server virtualization architecture and implementation. Crossroads, vol. 16, no. 1, 2009.

4. Kubernetes Production-Grade Container Orchestration, 2017. URL: https://kubernetes.io