УДК 004

Основы архитектуры технологии Kubernetes

Голубь Илья Сергеевич
Приамурский государственный университет имени Шолом-Алейхема
Магистрант

Аннотация
Целью исследования является изучение основ технологии контейнеризации Kubernetes. Методом исследования является изучение теоретической информации. Данная архитектура может избавить от проблем масштабирования, а так от проблем с пространством на жестком диске серверов.
Ключевые слова: Kubernetes, API, RestAPI, LoRaWAN.

Fundamentals of Kubernetes technology architecture

Golub Ilya Sergeevich
Sholom-Aleichem Priamursky State University
Undergraduate

Abstract
The aim of the research is to study the basics of Kubernetes containerization technology. The research method is the study of theoretical information. This architecture can eliminate scalability issues as well as hard disk space issues for servers.
Keywords: Kubernetes, API, RestAPI, LoRaWAN.

Введение

В реалиях современной разработки крупные компании, а так же подрядные организации приходят к тому, что приложения занимают очень много места при разворачивании на отдельном сервере с Linux или Widows Server. Решение данного вопроса сводиться к оптимизации кода, сжатии хранимых бинарных файлов и другим вариантам оптимизации места на жёстком диске. Одним из решением данного вопроса является технология контейнеризации Kubernetes.

На современном этапе актуальность данного вопроса высока. Данная технология решает сразу несколько проблем. Во-первых, это проблема свободного места, каждый отдельный контейнер поднимается со своим собранным образом на технологии Linux, в каждый образ можно закинуть максимальное малое количество дополнительных функций. Во-вторых, продукт, находящийся в контейнере изолирован от остальной среды, соответственно отказоустойчивость его повышается. В-третьих, масштабируемость таких контейнеров сводится к тому, что контейнер просто

копируется и разворачивается на другом сервере. Так же настраиваемый балансировки помогает регулировать трафик между контейнерами и улучшает отказоустойчивость.

Обзор исследований

В работе S Mohanty рассматривается вопрос образования новой парадигмы бессерверных вычислений. Бессерверные вычисления, также известные как функция как услуга, позволяют разработчикам развертывать функции как вычислительные блоки, не беспокоясь о базовой инфраструктуре. Более того, никакие ресурсы не выделяются и не выставляются счета до тех пор, пока функция не будет вызвана. Таким образом, основными преимуществами бессерверных вычислений являются уменьшение беспокойства разработчиков об инфраструктуре, сокращение времени вывода на рынок и более низкая стоимость. В настоящее время бессерверные вычисления обычно доступны через различных поставщиков общедоступных облачных сервисов. Однако на публичных облачных платформах существуют определенные узкие места, такие как привязка к поставщику, ограничения на вычисления и нормативные ограничения. Таким образом, растет интерес к внедрению бессерверных вычислений в частной инфраструктуре. Один из предпочтительных способов реализации бессерверных вычислений - использование контейнеров. Решение на основе контейнеров позволяет использовать функции существующих фреймворков оркестровки, таких как Kubernetes. В этой диссертации обсуждается реализация бессерверных вычислений в Kubernetes. С этой целью мы проводим оценку возможностей четырех фреймворков бессерверных вычислений с открытым исходным кодом, а именно Kubeless, OpenFaaS, Fission и OpenWhisk. На основе заранее определенных критериев мы выбираем Kubeless, Fission и OpenFaaS для дальнейшей оценки. Во-первых, мы опишем опыт разработчика по каждому фреймворку. Далее мы сравниваем три различных режима, в которых выполняются функции OpenFaaS: HTTP, сериализация и потоковая передача. Мы оцениваем время отклика при вызове функции и простоту мониторинга и управления журналами. Так же обнаружено, что режим HTTP является предпочтительным режимом для OpenFaaS. Наконец, оценивается производительность рассмотренных фреймворков при различных рабочих нагрузках. Мы обнаружили, что Kubeless имеет лучшую производительность среди трех фреймворков как с точки зрения времени отклика, так и соотношения успешных ответов[2].

В статье О.В. Сурай рассматривается масштабирование микросервисной архитектуры, построенной на базе Kubernetes. Рассмотрены общие преимущества от масштабирования, как для разрабатываемого сервиса, так и для команды разработки. Также приведены примеры нескольких способов настройки масштабируемости - императивный, декларативный и авто масштабированию по установленному проценту загрузки ЦП [5].

В статье Д.К. Загребаева описывается система управления контейнерами Kubernetes, являющаяся проектом с открытым исходным кодом, предназначенным для управления кластером контейнеров Linux как единой системой. Приводится ее архитектура, рассматривается кластерная вычислительная платформа Apache Spark фреймворк с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop. Рассмотрен мониторинг кластера на основе Kubernetes[4].

В работе J. Barriga предложена архитектура для развертывания решения интеллектуальной парковки на основе датчиков Long-Range Wide Area Network (LoRaWAN), LoRaWAN и кластера Kubernetes. Этот подход обеспечивает открытую архитектуру, способную обмениваться информацией с другими сторонами через интерфейс REST API. Точно так же он содержит мобильное и веб-приложение для взаимодействия с пользователем. Это решение предоставляет административный интерфейс для управления парковками. Пользовательский интерфейс позволяет пользователю находить, просматривать информацию, отображать доступные места и оценивать парковку в режиме реального времени. Это решение можно использовать как приложение в качестве системы служебной парковки. Предлагаемая архитектура полностью переносима и масштабируема за счет использования Kubernetes[1].

В диссертации рассматривается отсутствие поддержки библиотеки и общая низкая доступность реализации сценариев использования согласования настраиваемых состояний на Kubernetes при работе в рамках языка C # и платформы .NET. Решение в виде комплекта разработки программного обеспечения C # под названием KubeSharper было реализовано и оценено путем создания двух типичных приложений. Результаты показывают, что, пока требуются дальнейшие исследования и испытания, поставленное решение значительно снижает сложность обсуждаемых реализаций[3].

Цель исследования

Целью исследования является изучение основ технологии контейнеризации Kubernetes.

Методы исследования

Методами, используемыми в данной работе, будут являться изучение теоретической информации в сети интернет. Изучив теоретический материал, можно будет перейти к способу настройки, а так же развертывания контейнера с базовым образом, который предоставляется в открытом доступе.

Результаты и дискуссия

Kubernetes - это проект с открытым исходным кодом, который предназначается для управления кластером контейнеров Linux как целостной

системой. Kubernetes управляет и запускает контейнеры Docker на большом количестве хостов, и обеспечивает совместное размещение, репликацию большого количества контейнеров. Проект начали Google и в данный момент поддерживается многими компаниями, среди которых Microsoft, RedHat, IBM и Docker.

Компания Google использует контейнеризацию уже более десятка лет. Начали в Google с запуска более 2 млрд. контейнеров в течение недели. С помощью проекта Kubernetes компания делится своим опытом создания открытой платформы, предназначенной для масштабируемого запуска контейнеров.

Проект преследует несколько целей. Если вы пользуетесь контейнерами Docker, то возникает вопрос о том, как масштабировать и запускать контейнеры одновременно на большом количестве хостов Docker, а также как выполнять их балансировку. В проекте предлагается API, определяющее логическое группирование контейнеров, позволяющее определять пулы контейнеров, балансировать нагрузку, а также задавать их размещение.

В Kubernetes используются следующие концепции:

Nodes (node.md) - Нода это машина в кластере Kubernetes.

Pod - это группа контейнеров с общими разделами, которые запускаются как единое целое.

Replication Controllers (replication-controller.md): replication controller – контроллер репликации гарантирует, что определенное количество «реплик» pod'ы будут запущены в любой момент времени.

Services (services.md): Сервис в Kubernetes это абстракция которая определяет логический объединённый набор pod и политику доступа к ним.

Volumes (volumes.md): Volume(раздел) это директория, возможно, с данными в ней, которая доступна в контейнере.

Labels (labels.md): Label'ы это пары ключ/значение которые прикрепляются к объектам, например pod'ам. Label'ы могут быть использованы для создания и выбора наборов объектов.

Kubectl Command Line Interface (kubectl.md): kubectl интерфейс командной строки для управления Kubernetes.

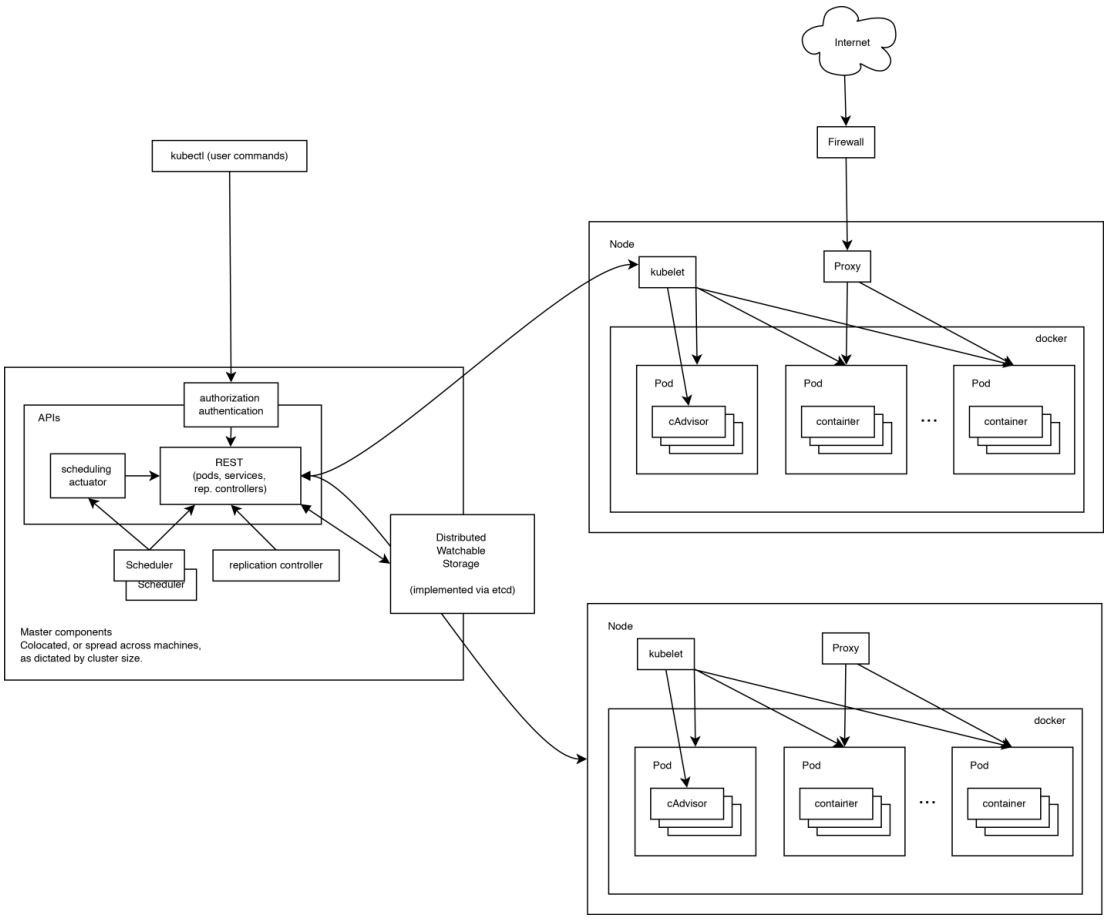

Рис. 1 Архитектура Kubernetes

Архитектура Kubernetes включает в себя агента, запущенного на нодах (kubelet) и некоторые компоненты мастера (APIs, scheduler, etc), поверх решения с распределённым хранилищем. Данная схема, показанная на рис. 1 изображает желаемое, в конечном итоге, состояние, хотя все ещё ведётся работа над некоторыми частями данной архитектуры. Например:необходимо сделать так, чтобы kubelet самостоятельно запускался в контейнере, что сделает планировщик на 100% подключаемым.

Смотря на архитектуру системы, мы можем разбить его на сервисы, работающие на каждой ноде,а так же на сервисы уровня управления кластера. На каждой отдельной ноде Kubernetes запускаются сервисы, необходимые для управления данной нодой со стороны мастера и для запуска приложений. На каждой ноде запускается Docker, обеспечивающий загрузку образов и запуск контейнеров.

Задача Kubelet заключается в управление pod'ами, их контейнерами, образами, разделами, etc.

Соответственно на каждой ноде запускается балансировщик, а так же настраивается в Kubernetes API. Называется данный балансировщик - Kube-Proxy. Он может выполнять простейшее перенаправление потоков TCP и UDP между набором бэкендов.

Компоненты управления Kubernetes разделены на несколько компонентов. В текущий момент все они запускаются на мастер-ноде, но в

ближайшее времени это будет изменено для повышения отказоустойсвости кластера. Эти компоненты работают вместе, чтобы обеспечить единое представление кластера.

Состояние мастера хранится в экземпляре etcd. Это необходимо для обеспечивания надёжного хранения конфигурационных данных и своевременное оповещение прочих компонентов об изменении состояния.

Kubernetes API - обеспечивает работу api-сервера. Данный RestAPI предназначен для того, чтобы дать доступ к CRUD операциям сервера со встроенной бизнес-логикой, реализованной в отдельных компонентах. Он, в основном, обрабатывает операции, проверяя их и обновляя соответствующие объекты в etcd.

Scheduler - планировщик привязывает незапущенные pod'ы к нодам через вызов binding API. Планировщик подключаем; планируется поддержка множественных планировщиков и использование пользовательских планировщиков.

Все остальные функции уровня кластера представлены в Controller Manager. Например, обнаруживаются, управляются и контролируются ноды средствами node controller. Данная сущность в итоге может быть разделена на отдельные компоненты, для реализации независимого подключения.

ReplicationController — это механизм, основывающийся на pod API. В конечном счете, планируется перевести её на общий механизм plug-in, когда он будет реализован.

Выводы

Подводя итог видно, что использование Kubernetes для масштабирования проекта, а так же для балансировки нагрузки на проект является одним из вариантов увеличения отказоустойчивости сервиса. Так же данная технология поможет ускорить сам процесс масштабирования, если использовать его в совокупность с CICD технологиями

Библиографический список

1. Barriga J. J. et al. A Smart Parking Solution Architecture Based on LoRaWAN and Kubernetes //Applied Sciences. 2020. Т. 10. №. 13. С. 4674.
2. Mohanty S. et al. Evaluation of Serverless Computing Frameworks Based on Kubernetes. 2018.
3. Orlovský V. An SDK for Building Kubernetes Operators in C#.
4. Загребаев Д. К. Мониторинг кластера анализа больших данных Apache Spark на основе Kubernetes // Достижения науки и образования. 2019. №5 (46). URL: https://cyberleninka.ru/article/n/monitoring-klastera-analiza-bolshih-dannyh-apache-spark-na-osnove-kubernetes (дата обращения: 13.12.2020).
5. Сурай О. В. Масштабирование микросервисной архитектуры на базе Kubernetes //Актуальные научные исследования в современном мире. 2018. №. 4-10. С. 118-122.