ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ТЕЛЕКОММУНИКАЦИИ
СПбГУТ))) Декабрь 2020 Том 8 № 4 ISSN 2307-1303 http://ijitt.ru/
TELECOM IT
December 2020 Vol. 8 Iss. 4 ISSN 2307-1303 http://ijitt.ru/

ИССЛЕДОВАНИЕ МОДЕЛЬНОЙ СЕТИ ДЛЯ ОРКЕСТРАЦИИ СЕРВИСОВ ИНТЕРНЕТА ВЕЩЕЙ НА ОСНОВЕ KUBERNETES

Д. В. Ермоленко*, К. Х. Киличева, А. А. Хакимов, А. С. А. Мутханна

Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича, Санкт-Петербург, 193232, Российская Федерация

*Адрес для переписки: daniil-ermolenko@mail.ru

Аннотация—**Предмет исследования.** Настоящая работа посвящена реализации концепции микросервисов, которая может быть использована для разработки и реализации распределенных граничных сервисов для приложений Интернета вещей. В основном IoT представляет собой платформу, где интегрированные сервисы связаны общей сетью, поэтому все устройства способны собирать и обмениваться данными друг с другом. Обычно монолитные сервисы мобильности пользователей разрабатываются для основных компонентов эталонной архитектуры унифицированных систем ETSI MEC. ETSI MEC рассматривает микросервисы как инструмент для разделения монолитных приложений на набор слабо связанных распределенных компонентов, т. е. которые предоставляет независимую единицу, реализующую какую-либо функцию. В MEC пользовательское оборудование через шлюзы подключается к микросервисам, работающим на граничном узле. Ожидается, что эта архитектура облегчит динамическую адаптацию во время выполнения приложения. Однако повышенная модульность может увеличить нагрузку и сложность оркестрации и управления системой. **Метод.** В работе распределенный сервис реализован с помощью контейнеров Docker и рассчитан на фактическую адаптацию к окружающим серверам малой емкости. **Основные результаты.** Результаты демонстрируют тот факт, что в реализации такого рода можно достичь низкой задержки для граничных приложений. Интеграция технологии создания программного обеспечения со стандартизированной граничной системой обеспечивает надежную основу для последующей разработки. **Практическая значимость.** В статье рассматривается применение архитектуры граничных вычислений и Kubernetes для оркестровки и управления сетевыми приложениями.

Ключевые слова—Edge computing, Kubernetes, IoT, кластеризация, оркестрация.

Информация о статье
УДК 621.396.93
Язык статьи – русский.
Поступила в редакцию 23.11.2020, принята к печати 23.12.2020.

Ссылка для цитирования: Ермоленко Д. В., Киличева К. Х., Хакимов А. А., Мутханна А. С. А. Исследование модельной сети для оркестрации сервисов Интернета вещей на основе Kubernetes // Информационные технологии и телекоммуникации. 2020. Том 8. № 4. С. 69–82. DOI 10.31854/2307-1303-2020-8-4-69-82.

EXPLORING A MODEL NETWORK FOR ORCHESTATION IOT SERVICES BASED ON KUBERNETES

D. Ermolenko*, K. Kilicheva, A. Khakimov, A. Muthanna

The Bonch-Bruevich Saint-Petersburg State University of Telecommunications,
St. Petersburg, 193232, Russian Federation

*Corresponding author: daniil-ermolenko@mail.ru

Abstract—This paper focuses on the implementation of the concept of microservices, which can be used to design and implement distributed edge services for IoT applications. Basically, IoT is a platform where integrated services are connected to a common network, so all devices are able to collect and exchange data with each other. Typically, monolithic user mobility services are developed for the core components of the ETSI MEC Unified Systems Reference Architecture. The ETSI MEC views microservices as a tool for separating monolithic applications into a set of loosely coupled distributed components, i.e. which provides an independent unit that implements a function. This architecture is expected to facilitate dynamic adaptation at runtime. However, increased modularity can also increase the load on orchestration and system management. In MEC, user equipment is connected through gateways to microservices running at the edge node. In the work, a distributed service is implemented using Docker containers and is designed to actually adapt to surrounding low-capacity servers. The results demonstrate that low latency for edge applications can be achieved with this kind of implementation. The integration of software technology with a standardized boundary system provides a solid foundation for subsequent development. This article discusses the use of edge computing architecture and Kubernetes for orchestration and management of network applications.

Keywords—Edge computing, Kubernetes, IoT, Clustering. Orchestration.

Article info
Article in Russian.
Received 23.11.2020, accepted 23.12.2020.

For citation: Ermolenko D., Kilicheva K., Khakimov A., Muthanna A.: Exploring a Model Network for Orchestation IoT Services Based on Kubernetes // Telecom IT. 2020. Vol. 8. Iss. 4. pp. 69–82 (in Russian). DOI 10.31854/2307-1303-2020-8-4-69-82.

Введение

Сегодня сети связи стали обязательной частью нашей жизни. Раньше сети создавались для соединения людей через вычислительные машины, но теперь они предназначены для соединения самих машин и подобных устройств [1, 2]. Внедрение Интернета вещей (IoT) оказывает значительное влияние на все аспекты человеческой деятельности, превращая предметы повседневной жизни в устройства связи [3]. В основном это происходит из-за способности Интернета вещей предоставлять удобные, эффективные и выполнимые решения для различных приложений, таких как здравоохранение, транспорт, безопасность и финансы. Согласно исследованию департамента Statista, к 2030 году количество подключенных устройств достигнет более 50 миллиардов.

Вышеупомянутые утверждения не позволяют существующей сетевой инфраструктуре поддерживать огромный рост трафика и предоставлять услуги с наилучшими показателями QoS[1,2]. При разработке будущих сетевых систем необходимо адаптировать существующие сетевые архитектуры к возрастающим потребностям, а также спроектировать и разработать новые возможности управления, которые помогут удовлетворить строгие требования будущих сценариев использования[3].

Именно концепция граничных вычислений способствует развитию гибкой сетевой архитектуры нового поколения сетей. Граничные вычисления – это новая вычислительная парадигма в построении сети, которая приближает ресурсы сервера, как центра обработки данных в меньшем масштабе, к конечным устройствам. Этот подход помогает раскрыть весь потенциал высокопроизводительных сетей доступа для сверхнизкой задержки и высокой скорости передачи, а также повышает устойчивость к проблемам в базовых сетях и центрах обработки данных[4] [4].

Сегодня граничные вычисления с множественным доступом (MEC) [5] – это стандартизированный подход Европейского института телекоммуникационных стандартов (ETSI) для доступа к граничным вычислениям на сетевом уровне. MEC, работающий на уровне базовой сети и доступа, является идеальным решением для большинства случаев использования. Однако есть еще некоторые проблемы, которые необходимо решить: первая связана с уязвимостью доступа к сети, а вторая связана с высокой нагрузкой на сети доступа и серверы MEC [6]. Это серьезное препятствие в крупномасштабных сценариях использования Интернета вещей, когда несколько датчиков могут производить большие объемы данных или, когда критически важные системные функции испытывают проблемы с доступом к сети. Реализация концепции в настоящее время возможна за счет использования технологий виртуализации сетевых функций (NFV) и программно-конфигурируемых сетей (SDN) [7].

Эта статья посвящена изучению возможности переноса некоторых граничных функций на локальный уровень в виде виртуализированных и динамически развертываемых компонентов, использующих локальное оборудование. В статье рассматривается модель многоуровневой сетевой архитектуры с использованием концепции граничных вычислений, виртуализации сетевых функций, управления сетевым взаимодействием с использованием структуры конфигурируемых программным обеспечением сетей и оркестровки приложений. Для исследования был реализован прототип локальной граничной сети на основе виртуализированных микросервисов, которые представлены в виде контейнеров Docker и развернуты с использованием системы оркестрации Kubernetes на рабочих узлах кластера.

[1] Mobile Edge Computing (MEC); Framework and Reference Architecture / 26.03.2020. URL: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf

[2] Recommendation ITU-R M.2083: IMT Vision, "Framework and overall objectives of the future development of IMT for 2020 and beyond," Sep. 2015.

[3] ITU-R, "Minimum requirements related to technical performance for IMT-2020 radio interface(s)," Nov. 2017.

[4] 3GPP TS 28.554, "Management and orchestration; 5G end to end Key Performance Indicators (KPI)", Ver. 2.0.0, release 15, Sep 2018.

Постановка задачи

Архитектура микросервисов – это недавно сформированная концепция в области моделей программного обеспечения, которая включает разделение монолитного приложения на отдельные процессы, реализующие одну функцию. Такой подход дает несколько преимуществ перед традиционными монолитными архитектурами:

1) снижение сложности за счет использования небольших приложений;
2) более легкое развертывание и удаление сервисных компонентов;
3) гибкость в использовании различных структур и инструментов;
4) предсказуемая масштабируемость;
5) повышение отказоустойчивости сервиса.

Архитектура микросервисов важна при реализации сетей граничных вычислений с точки зрения эффективного управления ресурсами и возможности масштабирования одного компонента для повышения производительности и отказоустойчивости.

Масштабируемость можно повысить за счет различных вариантов развертывания небольших компонентов. Таким образом, микросервисы представляют собой многообещающую парадигму для проектирования, внедрения и развертывания распределенных служб Интернета вещей. Проблема оркестрации современных вычислительных систем заключается в неоднородности современных сервисов, сложности управления функциональными блоками, а также в установлении взаимосвязи между ними. Современные стандарты MEC также требуют заблаговременного устранения сбоев и их последствий для обеспечения высокой доступности приложений. Однако эти проблемы могут быть решены путем использования комплексных решений для развертывания и разработки платформ граничных вычислительных сетей с набором инструментов для обеспечения непрерывного цикла разработки и интеграции сервисов, а также организации вычислительных и сетевых ресурсов. В настоящее время микросервисы реализуются с использованием технологии контейнеризации, которая предполагает виртуализацию на уровне операционной системы.

Собственно, Docker стал ключевым решением в развитии технологии контейнеризации. Он стандартизировал упаковку и доставку приложений, а также включил парадигму микросервисов и установил передовой опыт создания образов докеров, где есть только один рабочий процесс, который выполняет одну функцию приложения.

Docker обеспечивает безопасность, ручное управление вычислительными ресурсами и поддерживает сети оверлейного типа, позволяющие создавать контейнерные сети между серверами. Следующим шагом в разработке контейнеров стала стандартизация интерфейсов среды выполнения (интерфейс времени выполнения контейнера) и сетевых интерфейсов (интерфейс сети контейнера), что позволило отделить плоскость выполнения от управления и обеспечить совместимость между различными корпоративными решениями. А система Kubernetes позволяет обеспечить автоматизированное развертывание приложений на рабочих узлах, непрерывное обновление сервисов, администрирование жизненного цикла приложений, а также мониторинг всей вычислительной системы.

Предлагаемая система

На данный момент в телекоммуникационной сфере можно наблюдать постепенное внедрение новых технологий в организации сетевой и вычислительной инфраструктуры, а также новых паттернов проектирования сервисов. Новая концепция сетей NET-2030, развиваемая организацией ITU (*International Telecommunication Union*), еще находится в разработка, но уже становится понятно, что будущее в телекоммуникационной индустрии направлено на размытие границы между сотовой связью и фиксированной сетью связи, разработка общих принципов построения сервисов, обеспечение высокой доступности приложений, как территориально, так и технически, а также обеспечение еще больших скоростей и показателей качества обслуживания по сравнению с существующей инфраструктурой. В качестве технологий сетевых инфраструктурных технологий стоит считать SDN (*Software-defined Networking*) и NFV (*Network Function Virtualization*).

IoT устройства с каждым годом становятся более значимыми в повседневной жизни, и их количество растет. Для поддержания данного явления требуется новые подходы к проектированию сетевых вычислительных инфраструктур и управлению сетевыми сервисами. В последние несколько лет приложения IoT были реализованы как набор небольших и независимых микросервисов. Микросервисная архитектура – относительно новый термин в моделях программного обеспечения. Парадигма микросервисов является расширением традиционной парадигмы сервис-ориентированной архитектуры (SOA), в которой приложение разбивается на набор детализированных сервисов. Каждый микросервис может быть представлен легким контейнером, который может использоваться множеством клиентов. Например, в сценарии умного города ресурсы должны быть распределены внутри сети, чтобы микросервисы, составляющие приложение, располагались рядом с конечным устройством, которое запрашивает приложение IoT. Микросервисы в большей степени определяют новую структуру построения сети. Чтобы обеспечить правильное распределение ресурсов, необходимо учитывать множество факторов, таких как низкая задержка, пропускная способность, энергоэффективность и стоимость.

Именно вышеперечисленные технологии стали ключевыми в развитии граничных вычислений, микросервисы определили подход к развертыванию служб, синергия программно-определяемых сетей и виртуализация сетевых функция заложила базовые принципы в разработке и эксплуатации граничных сервисов. Регулирующей организацией по стандартизации и снабжению спецификациями стал ETSI (*European Telecommunications Standards Institute*). Рабочая группа, в которую входят инженеры известных телекоммуникационных компаний, на данный момент определила технологию граничных вычислений с множественным доступом (MEC). MEC позволяет достичь близость услуги к пользователю, низкую круговую задержку, минимизировать использование полосы пропусканию сети общего пользования, а также самое главное предоставить устройствам вычислительные мощности для обработки информации и хранилища для ее хранения. MEC предполагает использование с разными типами сетей доступа (как мобильные сети, так и фиксированные), что точно определяет MEC как ключевую технологию в построении сетей NET-2030. На рис. 1 показан простой пример прикрепления граничного облака к сети общего пользования.

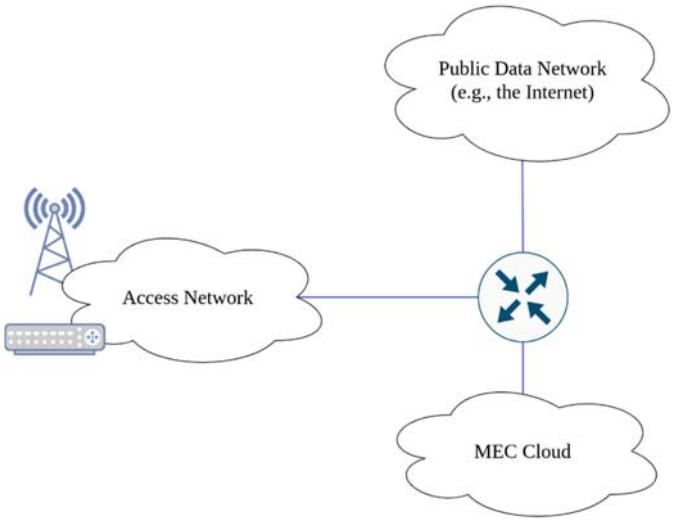

Рис. 1. Концепция MEC

Для управления и оркестрацией приложений Интернета вещей, необходимо применение оркестратора микросервисов для обеспечения стабильной работы и масштабируемости эксплуатируемой вычислительной системы. На сегодняшний день при применении микросервисной архитектуры активно используется технология контейнеризации, которая позволяет обеспечить в рамках виртуализации сетевых функций большую эффективность использования вычислительных ресурсов, предсказуемую масштабируемость и удобство в разработке микросервисных приложений. Для преодоления сложности оркестрации и управления служб при таком подходе к построению граничных вычислительных систем предлагается использование Kubernetes как базисного элемента, а в качестве платформы виртуализации использование OCI-совместимых средств по управлению контейнерами. В данной работе мы концентрируемся на проработке двух основных аспектах технологии multi-access edge computing (MEC): развертывание (обеспечение работы) и эксплуатации (обеспечение масштабируемости и адаптации системы).

Предлагаемая модельная сеть граничного облака для сервисов IoT

На рис. 2 (см. ниже) приведена общая архитектура предлагаемой модели граничного облака для приложений Интернета вещей. Данный фреймворк объединяет архитектуры SDN и NFV MANO (*Management and Orchestration*), что обеспечивает стабильную инфраструктуру для развертывания приложения, включения служб безопасности и обеспечения эксплуатационных возможностей по сопровождению, оркестрации и масштабируемости приложений. Такая инфраструктура сетевых и вычислительных ресурсов позволяет реализовать новые подходы к развертыванию приложений с необходимыми вычислительными ресурсами.

Граничное облако состоит из шлюза для сети доступа, брандмауэра, обеспечивающий первоначальную фильтрацию трафика, для соответствия подключенным сервисами, и кластера Kubernetes, который обеспечивает серверную и вычислительную инфраструктуру облака. Также благодаря гибкости архитектуры Kubernetes, система предоставляет входные точки для обеспечения хранения данных, в том числе возможно применения распределенных файловых систем.

Кластер Kubernetes главным образом состоит из управляющей плоскости и рабочей плоскости. Управляющая плоскость может быть представлена как одним мастер узлом, так и объединенных в кластер компонентов управления для достижения повышенной отказоустойчивости всей системы. Control plane

включает в себя такие компоненты как: etcd, API server, Controller-manager, Scheduler.

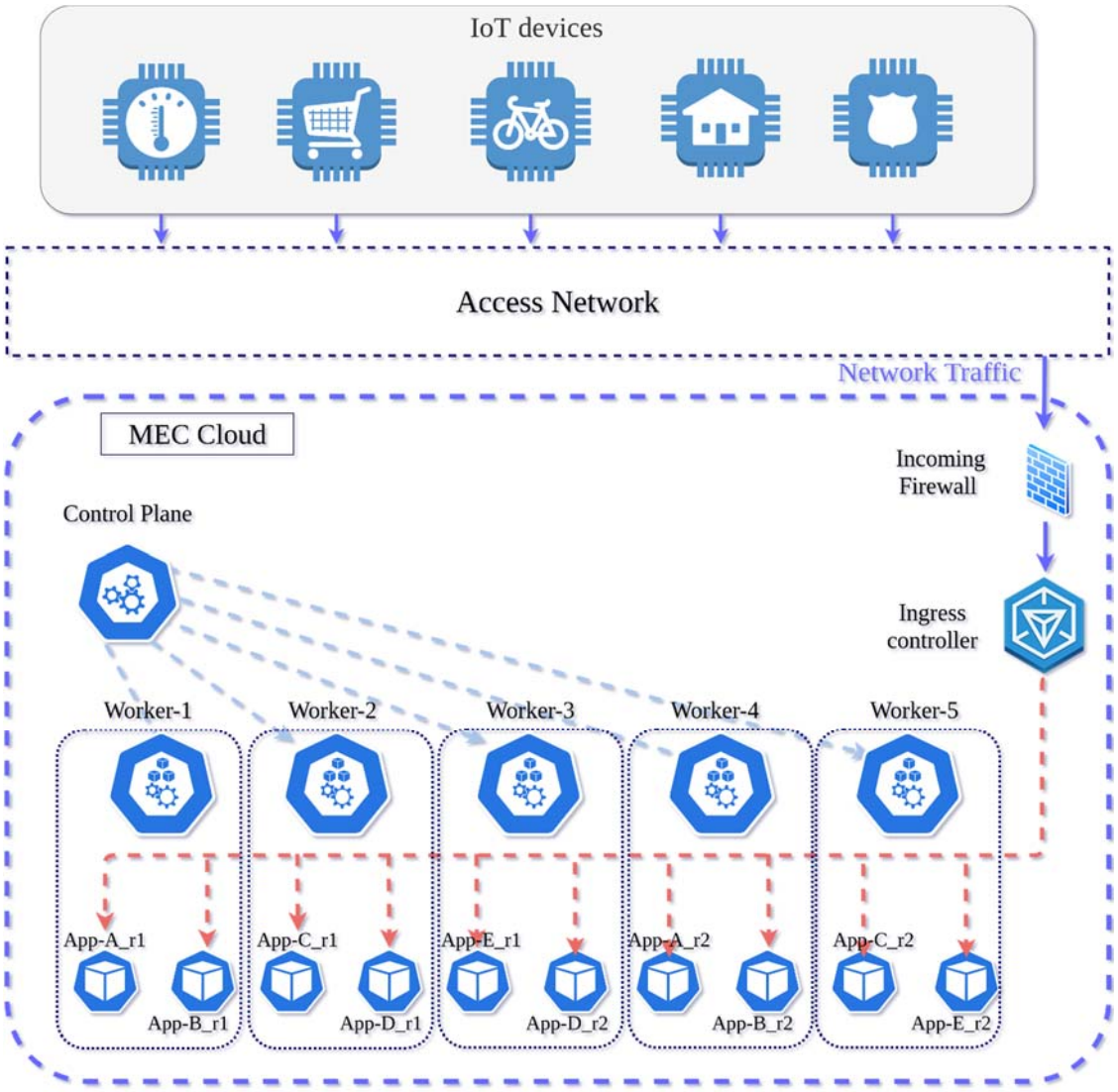

Рис. 2. предлагаемая реализация граничного облака

Etcd – это хранилище кластера типа ключ-значение, содержащее всю информацию о кластере, все настройки, описание объектов и их секретные ключи. API сервер является центральной частью плоскости управления. Данный компонент обрабатывает запросы от остальных компонентов кластера, работая по REST протоколу с использованием механизмов шифрования соединения. Controller-manager состоит из нескольких контроллеров процессов управления в кластере. Опишем три основных контроллера. Node controller отвечает за уведомление о стоянии узла и реагировании при его недоступности. Replication controller поддерживает корректное количество pod'ов для каждого объекта ReplicaSet в системе. Endpoint controller закрепляет объект Endpoint, который соединяет kube service и pod. Scheduler занимается назначением pod на узел и распределением ресурсов, он учитывает показатель QoS, Affinity/anti-affinity, запрошенные ресурсы.

На рабочих узлах разворачиваются непосредственно сами экземпляры приложения, и ключевыми компонентами системы являются kubelet и kube-proxy, которые работают на всех машинах кластера, в том числе управляющих. Kubelet обеспечивает взаимодействие и контроль с инфраструктурой виртуализации, а kube-proxy занимается реализацией сущности сервиса в кластере, о данной абстракции мы поговорим.

Элементы связности системы

Для рассмотрения организации сети в kubernetes необходимо рассмотреть объекты связанные с обеспечением прохождения сетевого трафика к приложению.

Service

Сервисы позволяют связать приложение или его экземпляр за определенным IP-адресом и DNS-именем. Они необходимы для публикации части приложения, отвечающего за прием/передачау трафика, за пределы кластера. Типы сервисов:

- ClusterIP;
- NodePort;
- LoadBalancer;
- ExternalName;
- Headless.

Ingress

Ingress используется для получения запросов из интернета, описывает правила доступа к приложению по протоколу http. В абстракции указывается имя хоста, например, my.host.foo, которое соответствует имени сервера, а так на какой порт сервиса пересылать трафик. При получении http запроса на hostname, кластер видит, что есть такой ingress с запрашиваемым именем хоста, по описанию принимается решение о том, что трафик нужно отсылать на соответствующий сервис с корректным портом.

Сам Kubernetes не обеспечивает работу Ingress, для этого необходим специальный модуль, называемый Ingress conrtoller.

Ingress controller базируется на прокси сервере, в котором сопоставляются сервисы и конечные точки pod’ов, что обеспечивает прохождения трафика напрямую. Конфигурация прокси сервера составляется из описания ingress’ов. На данный момент ingress контроллеры могут распределять udp и tcp трафик, то есть работать на 4 уровне выполняя роль универсального арбитра трафика.

kube-proxy это компонент кластера, который общается с API Server, стоит на всех узлах, управляет сетевыми правилами на узлах и фактически реализует Service. Реализуется на базе iptables или ipvs. Сетевые правила определяют передачу трафика от сервиса к поду.

Реализация сервиса через iptables:

```
-A KUBE-SERVICES
  -d 1.1.1.1/32
  -p tcp
  -m comment --comment “mynamespace/myservice:http cluster IP”
  -m tcp --dport 80
```

```
-j KUBE-SVC-UT8A43GJFBEDGO3V

-A KUBE-SVC-UT8A43GJFBEDGO3V
  -m comment --comment "mynamespace/myservice:http"
  -m statistic
    --mode random --probability 0.500000000
  -j KUBE-SEP-MMYWB6DZJI483RW

-A KUBE-SVC-UT8A43GJFBEDGO3V
  -m comment --comment "mynamespace/myservice:http"
-j KUBE-SEP-J33WXLDEJI483RW

-A KUBE-SEP-MMYWB6DZJI483RW
  -p tcp
  -m comment --comment "mynamespace/myservice:http"
  -m tcp
-j DNAT
  --to-destination 10.102.3.49:80

-A KUBE-SEP-J33WXLDEJI483RW
-p tcp
  -m comment --comment "mynamespace/myservice:http"
  -m tcp
-j DNAT
  --to-destination 10.102.0.93:80
```

Анализируя цепочку правил можно сделать вывод о том, что kube-proxy реализует round robin алгоритм для сервиса, что обеспечивает равномерную нагрузку между экземплярами, привязанными к объекту сервиса.

CNI (Container Network Interface)

Интерфейс сети контейнеров – это проект Cloud Native Computing Foundation, состоящий из спецификаций и библиотек для написания плагинов, которые занимаются настройкой сетевых интерфейсов в контейнерах Linux. Kubernetes реализует интерфейс CNI для создания оверлей сети, которая обеспечивает маршрутизацию трафика между подами разных узлов. Некоторые плагина для Kubernetes:

- Flannel предоставляет простой и легкий способ настройки сетевой инфраструктуры третьего уровня. Имеет поддержку jumbo frames, способен работать в режиме VXLAN.
- Calico – это виртуальная сеть третьего уровня, в которой есть возможность применять политики. Поддержка IP-IP тунеля и VXLAN.
- Cilium – применяет в качестве управления сетевым трафиком механизмы ядра Linux BPF и XDP.
- ovn-kubernetes – предоставляет оверлейную сеть, построенную на Open vSwitch (OVS) и Open Virtual Networking (OVN) с поддержкой как Linux, так и Windows.

Для построения сети я предлагаю использовать CNI плагин flannel в режиме host-gateway. Данные режим является эффективным с точки зрения задержки и пропускной способности, но требует объединение сети в один L2-сегмент. Flannel обеспечивает маршрутизацию между узлами и подами кластера за счет использования правил маршрутизации и контролированию виртуальных сетевых интерфейсов. Контейнер flanneld работая в пространстве kube-system обеспечивает выдачу ip адресов и предотвращение сетевых конфликтов.

На рис. 3 представлена модель сети, которая иллюстрирует сетевое взаимодействие в граничном облаке.

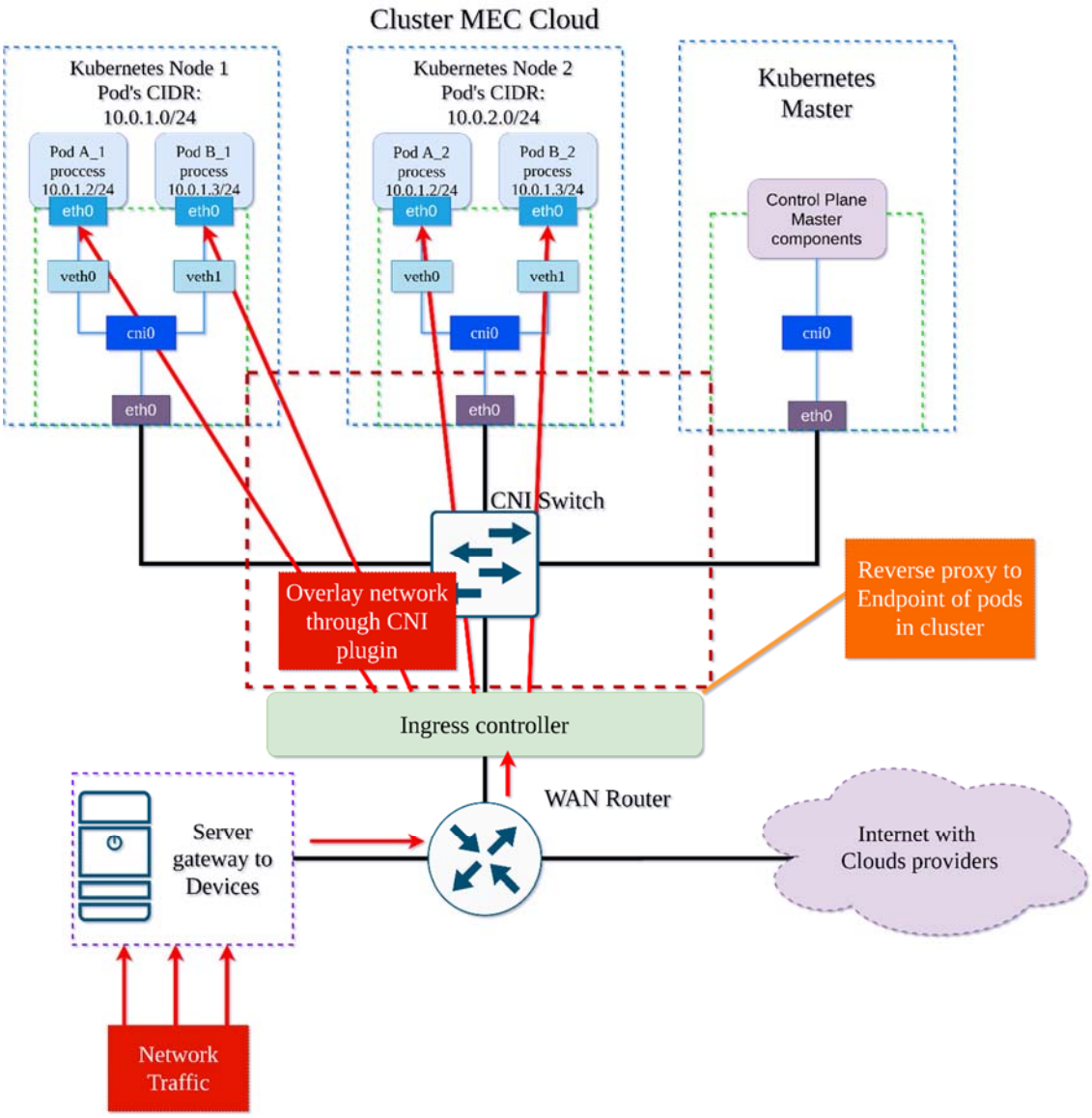

Рис. 3 Модель потока трафика

Целью экспериментальной части настоящей статьи является исследование работы протокола IoT устройств в имитационной реализации граничного локального облака. В качестве протокола IoT устройств был выбран http. Выбрали его, потому что для него существует множество методик тестирования, является базовым протоколом для построения приложений Интернет-вещей, а также методы протокола используется в паттерне проектирования REST, который ориентирован

на работу с ресурсами. Для исследования работоспособности модельной сети проведем два сравнительных эксперимента, где будем замерять время получения ответа от сервера.

Для тестирования был написан сценарий на языке python. Сценарий описывает отправку get-запросы с созданием сессии на сервер, для этого использовалась библиотеке aiohttp и asyncio, чтобы обеспечить многопоточное формирование запросов, тем самым увеличивается производительность запросов в секунду по сравнению с классической библиотекой urllib. Запросы генерируются в пределах от 10 до 50 запросов в секунду с шагом 10. Также на время работы генератора трафика запущен инструмент iperf3 на обоих концах соединения (*client-server*) чтобы нагрузить tcp соединениями канал связи. Клиент стартовал со следующими параметрами: 100 параллельных запросов и длина буфера чтения/записи 100 kB. Такие параметры отражают реальную нагрузку от устройств Интернет-вещей. Длительность каждого шага 10 мин, что минимизирует погрешность измерений.

Эксперимент 1. Сервер Apache развернутый на виртуальной машине

Клиент – это ноутбук, который создает нагрузку tcp-соединений через инструмент iperf3 и генерирует http get-запросы к серверу.

Логически клиент и сервер находятся в одном сегменте локальной сети, который реализуется с помощью мостового соединения на интерфейсе ноутбука. На рис. 4 показана схема экспериментального участка локальной сети.

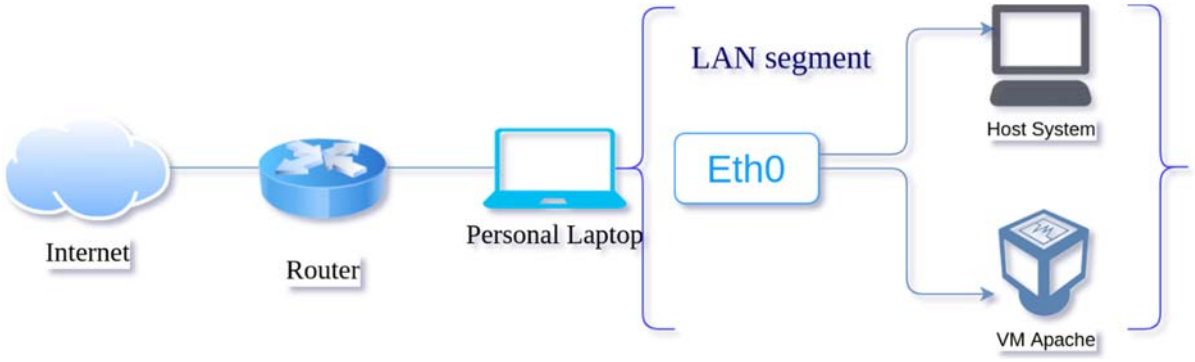

Рис. 4. Эксперимент 1

Эксперимент 2. Сервер Apache развернутый в кластере Kubernetes

Реализация локального участка сети аналогична первому эксперименту. Кластер представляет собой три узла работающие как виртуальные машины Virtualbox. Управляющий узел и два рабочих связаны CNI-плагином Flannel, который работает в режиме host-gw. Для доступа к веб-серверу используется Ingress. Трафик равномерно распределяется между двумя репликами Apache, которые распределены на каждый рабочий узел кластера. Рис. 5 иллюстрирует схему локальной сети для второго эксперимента.

Конфигурация Лэптопа: Ноутбук Asus x507ma, Процессор Intel Pentium N5000, 8 ГБ DDR4-2400 одноранговой одноканальной оперативной памяти, 480 ГБ SSD на ячейках памяти MLC. Операционная система Fedora Linux 32, ядро версии 5.8.4, VirtualBox 6.1.

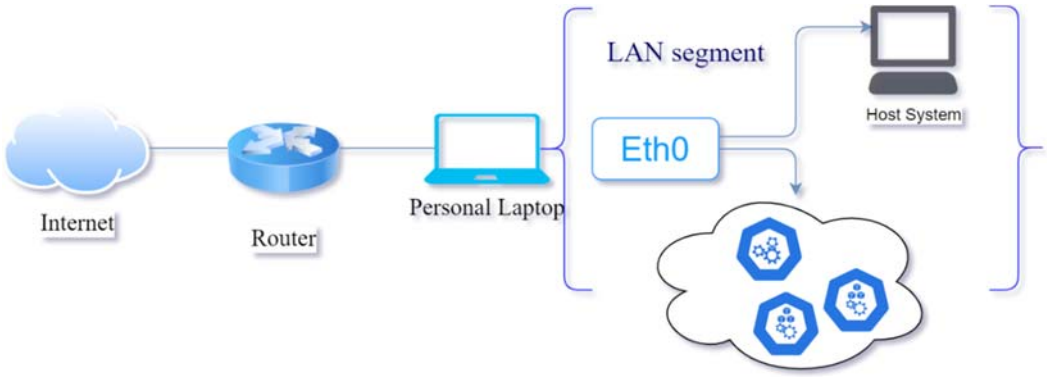

Рис. 5. Эксперимент 2

Результаты

Как уже упоминалось выше, целью эксперимента было определить время отклика от сервера. На каждом этапе тестирования, пакеты захватывались с помощью утилиты tcpdump. Далее в Wireshark определялось время с момента запроса для каждого ответа, а затем вычисляли среднее значение полученного параметра на протяжении всего этапа тестирования. На рис. 6 представлена сравнительная диаграмма среднего времени отклика сервера для двух экспериментов.

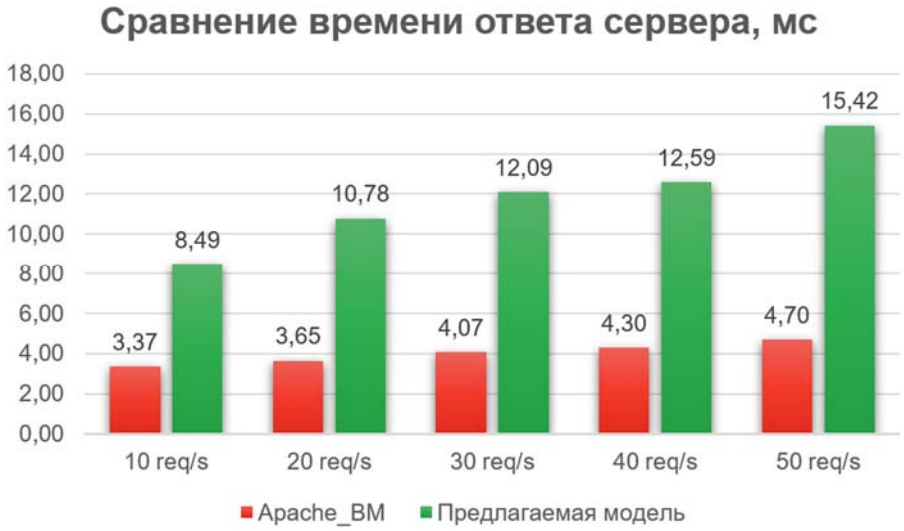

Рис. 6. Результаты тестирования, первый вариант развертывания

Анализ результатов двух экспериментов показывает, что приложение Apache, развернутое в кластере Kubernetes, уступает классической версии на одном сервере. Но, как видите, результаты не растут линейно, это связано с тем, что iperf3 требует большого количества процессорного времени для создания параллельных запросов, поэтому тестирование не отражает тот результат, который мы хотели бы видеть. Чтобы снизить нагрузку на систему и получить больше производительности от экспериментального клиента и сервера, можно использовать minikube. Minikube предоставляет более компактное, но менее гибкое решение для развертывания локального кластера Kubernetes из коробки.

Проведем второй сценарий тестирования предложенного фреймворка (рис. 7), конфигурация кластера останется такой же, как и в первом сценарии. Виртуальная машина кластера развертывается с использованием гипервизора kvm, 4 vcpu, 3 ГБ оперативной памяти. Эта конфигурация требует меньше vCPU и объема RAM.

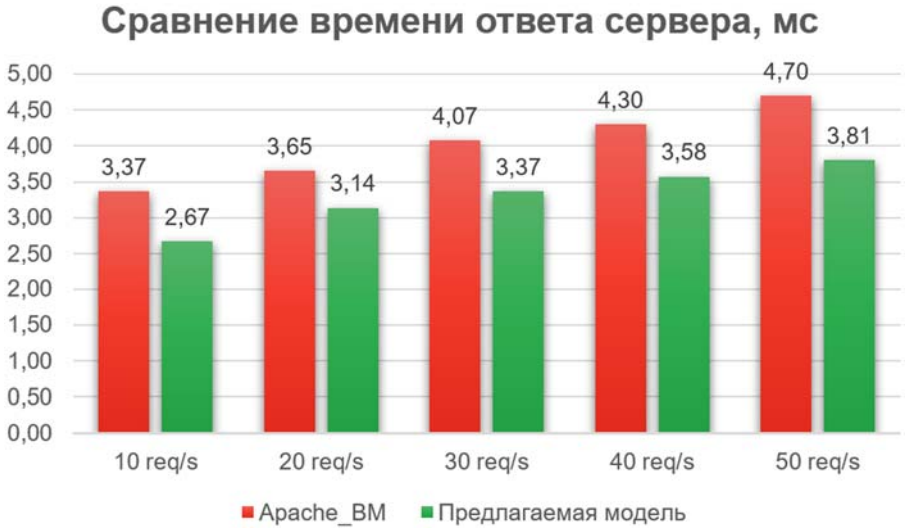

Рис. 7 Второй вариант для тестирования предлагаемой модели

Анализируя полученные результаты, можно сделать вывод, что балансировка нагрузки дает положительный результат по сокращению времени отклика, а с помощью оркестаратора Kubernetes можно легко масштабировать и изменять параметры кластерных приложений. Например, значение числа реплик задается в файле манифеста объекта развертывания, который является плоскостью управления и принимает в качестве декларативного описания работу приложения. Кроме того, в конфигурации http-сервера использовался механизм configmap для передачи конфигурации внутри контейнера с приложением.

Заключение

В этой статье мы предложили основу для развертывания современных облачных приложений Интернета вещей на базе оркестратора и менеджера контейнеров Kubernetes. В ходе работы и тестирования модель показала удовлетворительные результаты с точки зрения производительности. При развертывании граничного облака можно применять легкое масштабирование в зависимости от задач, дополнительные абстракции объектов для настройки и подключать хранилище данных. Граничные вычисления имеют колоссальное значения для масштабного развертывания глобального Интернета вещей, и именно разработка и оптимизации фреймворка граничного облако позволит достичь максимально эффективную вычислительную систему, способную к адаптации в зависимости от видов приложений, нагрузок и условий работы оборудования. Подход к объединению разнесённого приложения в единый вычислительный кластер и обеспечение надежного канала передачи данных приносит концепцию всеобъемлющего бесшовного вычислительного центра, объединяя граничные сервера между собой, а также граничное облако и через шлюз сервисы сети общего пользования. Будущая работа будет заключаться в том, чтобы предоставить фреймворк

который способен оптимизировать ресурсы и эффективно распределять гетерогенный трафик для различных приложений, которые требуют высокий отклик сервера.

Литература

1. Бородин А. С., Кучерявый А. Е. Сети связи пятого поколения как основа цифровой экономики // Электросвязь. 2017. № 5. С. 45–49.
2. Кучерявый А. Е., Парамонов А. И., Кучерявый Е. А. Сети связи общего пользования. Тенденции развития и методы расчёта. – М.: ФГУП ЦНИИС, 2008. – 296 с.
3. Кучерявый А. Е. Интернет Вещей // Электросвязь. 2013. № 1. С. 21–24
4. Ateya A. A., Muthanna A., Koucheryavy A. 5G framework based on multi-level edge computing with D2D enabled communication // 20th International Conference on Advanced Communication Technology (ICACT) conference proceedings. 2018. С. 507–512.
5. Мутханна А.С. Интеллектуальная распределенная архитектура сети связи для поддержки беспилотных автомобилей // Электросвязь. 2020. № 7. С. 29–34.
6. Vladyko, A.; Khakimov, A.; A. Muthanna et al. Distributed Edge Computing to Assist UltraLow-Latency VANET Applications // Future Internet. 2019. Vol. 11, Issue 6. P. 128.
7. Владыко А. Г., Мутханна А. С., Киричек Р. В., Волков А. Н., Маколкина М. А., Парамонов А. И. Программируемые сети SDN. – СПб.: Издательство «Лигр», 2019. – 120 с. ISBN 978-5-907207-41-7.

References

1. Borodin A. S., Koucheryavy A. E. Fifth generation networks as a base to the digital economy // Electrosvyaz'. 2017. No 5. pp. 45–49.
2. Kucheryavy`j A. E., Paramonov A. I., Kucheryavy`j E. A. Seti svyazi obshhego pol`zovaniya. Ten-dencii razvitiya i metody` raschyota. – M.: FGUP CzNIIS, 2008. – 296 s.
3. Kucheryavyj A. E. Internet Veshchej // Electrosvyaz'. 2013. No 1. PP. 21–24.
4. Ateya A.A., Muthanna A., Koucheryavy A. 5G framework based on multi-level edge computing with D2D enabled communication. В сборнике: 20th International Conference on Advanced Communication Technology (ICACT) conference proceedings. 2018. С. 507-512.
5. Muthanna A. S. Distributed intelligent communication network architecture for unmanned vehicles // Electrosvyaz'. 2020. No 7. pp. 29–34.
6. Vladyko, A.; Khakimov, A.; A. Muthanna et al. Distributed Edge Computing to Assist UltraLow-Latency VANET Applications // Future Internet. 2019. Vol. 11, Issue 6. P. 128.
7. Vlady`ko A. G., Mutxanna A. S., Kirichek R. V., Volkov A. N., Makolkina M. A., Paramonov A. I. Programmiruemy`e seti SDN. – SPb.: Izdatel`stvo «Ligr», 2019. – 120 s. ISBN 978-5-907207-41-7.

Ермоленко Даниил Владимирович – магистрант Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича, daniil-ermolenko@mail.ru
Ermolenko Daniil – Undergraduate student, The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, daniil-ermolenko@mail.ru

Киличева Клавдия Хайруллоевна – магистрант Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича, bambosik@yandex.ru
Kilicheva Claudia – Undergraduate student, The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, bambosik@yandex.ru

Хакимов Абдукодир Абдукоримович – аспирант Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича, Khakimov-aa@rudn.ru
Khakimov Abdukodir – Postgraduate student, The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Khakimov-aa@rudn.ru

Мутханна Аммар Салех Али – кандидат технических наук, доцент кафедры Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича, ammarexpress@gmail.com
Muthanna Ammar – Candidate of Engineering Sciences, Associate Professor, The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, ammarexpress@gmail.com