ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 70860— 2023

Информационные технологии

ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ

Общие технологии и методы

(ISO/IEC TS 23167:2020, NEQ)

Издание официальное

Москва
Российский институт стандартизации
2023

gost.e-ecolog.ru

Предисловие

1 РАЗРАБОТАН Обществом с ограниченной ответственностью «Информационно-аналитический вычислительный центр» (ООО ИАВЦ), Институтом системного программирования им. В.П. Иванникова РАН (ИСП РАН)

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 22 «Информационные технологии»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 22 августа 2023 г. № 700-ст

4 Настоящий стандарт разработан с учетом основных нормативных положений международного документа ISO/IEC TS 23167:2020 «Информационные технологии. Облачные вычисления. Общие технологии и методы» (ISO/IEC TS 23167:2020 «Information technology — Cloud computing — Common technologies and techniques», NEQ)

5 ВВЕДЕН ВПЕРВЫЕ

6 Федеральное агентство по техническому регулированию и метрологии не несет ответственности за патентную чистоту настоящего стандарта

*Правила применения настоящего стандарта установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (*gost.e-ecolog.ru

© Оформление. ФГБУ «Институт стандартизации», 2023

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Федерального агентства по техническому регулированию и метрологии

Содержание

1 Область применения 1
2 Нормативные ссылки 1
3 Термины и определения 2
4 Сокращения и обозначения 3
5 Обзор общих технологий и методов, используемых в сфере облачных вычислений 4
5.1 Общие положения 4
5.2 Технологии 4
5.3 Методы 5
6 Виртуальные машины и гипервизоры 5
6.1 Общие положения 5
6.2 Виртуальные машины и виртуализация систем 6
6.3 Гипервизоры 6
6.4 Безопасность ВМ и гипервизоров 8
6.5 Образы ВМ, метаданные и форматы 8
7 Контейнеры и системы управления контейнерами (CMS) 9
7.1 Общие положения 9
7.2 Контейнеры и виртуализация операционных систем 9
7.3 Образы контейнера и уровни файловой системы 12
7.4 Системы управления контейнерами 15
8 Бессерверные вычисления 17
8.1 Общие положения 17
8.2 Функции как услуга 17
8.3 Бессерверные базы данных 19
9 Архитектура микросервисов 20
9.1 Общие положения 20
9.2 Преимущества и недостатки микросервисов 21
9.3 Спецификация микросервисов 22
9.4 Многоуровневая архитектура 22
9.5 Сервисная сетка 24
9.6 Автоматический размыкатель 26
9.7 API-шлюз 26
10 Автоматизация 26
10.1 Общие положения 26
10.2 Автоматизация жизненного цикла разработки 27
10.3 Инструменты автоматизации 27
11 Архитектура систем PaaS 28
11.1 Общие положения 28
11.2 Отличительные особенности систем PaaS 29
11.3 Архитектура компонентов, работающих под управлением системы PaaS 31
12 Хранение данных как услуга 32
12.1 Общие положения 32
12.2 Общие характеристики сервисов DSaaS 33
12.3 Типы функциональных возможностей DSaaS 36
12.4 Важные дополнительные функциональные возможности DSaaS 37

13 Создание сетевых подключений в облачных вычислениях 37
13.1 Ключевые аспекты создания сетевых подключений 37
13.2 Сетевое подключение для доступа к облаку 38
13.3 Внутриоблачное сетевое подключение 38
13.4 VPN и облачные вычисления 39
14 Масштабируемость облачных вычислений 40
14.1 Подходы к обеспечению масштабируемости 40
14.2 Параллельные экземпляры и балансировка нагрузки 41
14.3 Адаптивность и автоматизация 42
14.4 Масштабирование базы данных 42
15 Безопасность и общие технологии облачных вычислений 43
15.1 Общие положения 43
15.2 Межсетевые экраны 43
15.3 Защита конечных точек 43
15.4 Управление идентификацией и доступом пользователей 44
15.5 Шифрование данных 44
15.6 Управление ключами 44
Библиография 45

Введение

Облачные вычисления на уровне концепций описаны в ГОСТ ISO/IEC 17788, а также в [1].

С ростом объемов использования облачных вычислений увеличился и набор общепринятых технологий, предназначенных для поддержания, упрощения и расширения применения облачных вычислений, а также набор общепринятых методов, позволяющих эффективно использовать функциональные возможности облачных сервисов. Многие из этих технологий и методов предназначены для разработчиков и операторов, которые все чаще работают вместе в рамках унифицированного подхода, получившего название DevOps (см. 10.2). Цель такого подхода заключается в том, чтобы ускорить и упростить создание и использование решений на базе облачных сервисов.

В настоящем стандарте описываются общие технологии и методы, относящиеся к облачным вычислениям, их связь друг с другом и принципы их использования для решения некоторых задач, связанных с облачными вычислениями.

Настоящий стандарт касается развивающихся технологий, которые в будущем потребуют разработки новых стандартов.

Настоящий стандарт предназначен в первую очередь для разработчиков сервисов в категории «Поставщики облачных сервисов» и разработчиков отраслевых стандартов, работающих с организациями по стандартизации.

Настоящий стандарт необходимо применять с учетом требований нормативных правовых актов и стандартов Российской Федерации в области защиты информации.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационные технологии

ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ

Общие технологии и методы

Information technology. Cloud computing. Common technologies and techniques

Дата введения — 2024—01—30

1 Область применения

Настоящий стандарт содержит описание набора общих технологий и методов, используемых в сфере облачных вычислений. К ним относятся:

- виртуальные машины (ВМ) и гипервизоры;
- контейнеры и системы управления контейнерами (CMS);
- бессерверные вычисления;
- архитектура микросервисов;
- инструменты автоматизации;
- системы и архитектура «платформа как услуга»;
- сервисы хранения данных;
- средства обеспечения безопасности, масштабируемости и сетевого подключения в контексте вышеуказанных технологий облачных вычислений.

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты:

ГОСТ ISO/IEC 17788—2016 Информационные технологии. Облачные вычисления. Общие положения и терминология

ГОСТ Р ИСО/МЭК 17203 Информационная технология. Спецификация открытого формата визуализации (OVF)

ГОСТ Р ИСО/МЭК 18384-1 Информационные технологии. Эталонная архитектура для сервис-ориентированной архитектуры (SOA RA). Часть 1. Терминология и концепции SOA

Примечание — При пользовании настоящим стандартом целесообразно проверить действие ссылочных стандартов в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет или по ежегодному информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по выпускам ежемесячного информационного указателя «Национальные стандарты» за текущий год. Если заменен ссылочный стандарт, на который дана недатированная ссылка, то рекомендуется использовать действующую версию этого стандарта с учетом всех внесенных в данную версию изменений. Если заменен ссылочный стандарт, на который дана датированная ссылка, то рекомендуется использовать версию этого стандарта с указанным выше годом утверждения (принятия). Если после утверждения настоящего стандарта в ссылочный стандарт, на который дана датированная ссылка, внесено изменение, затрагивающее положение, на которое дана ссылка, то это положение рекомендуется применять без учета данного изменения. Если ссылочный стандарт отменен без замены, то положение, в котором дана ссылка на него, рекомендуется применять в части, не затрагивающей эту ссылку.

Издание официальное

3 Термины и определения

В настоящем стандарте применены термины по ГОСТ ISO/IEC 17788, а также следующие термины с соответствующими определениями:

3.1

гостевая операционная система (guest operating system): Операционная система, установленная на виртуальной машине.

[ГОСТ Р 59163—2020, пункт 3.2]

3.2

хостовая операционная система (host operating system): Операционная система, в среде которой функционирует гипервизор.

[ГОСТ Р 56938—2016, пункт 3.12]

Примечание — Программное обеспечение для виртуализации может включать в себя гипервизор и ВМ, а также контейнерную службу и контейнеры.

3.3 **бессерверные вычисления** (serverless computing): Категория облачных сервисов, в рамках которой потребитель сервиса облачных вычислений может использовать различные типы функциональных возможностей облачных сред без необходимости предоставления, развертывания и управления аппаратными или программными ресурсами, за исключением предоставления кода приложения или пользовательских данных, обрабатываемых приложением.

Примечания

1 Бессерверные вычисления обеспечивают автоматическое масштабирование с динамическим адаптивным выделением ресурсов поставщиком облачной службы, автоматическое распределение ресурсов по нескольким объектам, а также автоматическое обслуживание и резервное копирование.

2 Функции бессерверных вычислений инициируются одним или несколькими событиями, определенными потребителем сервиса облачных вычислений, и могут выполняться в течение ограниченного периода времени, необходимого для обработки каждого конкретного события.

3 Функции бессерверных вычислений могут быть инициированы прямым вызовом из мобильных и веб-приложений.

3.4 **контейнерная служба** (container daemon): Программный сервис, выполняющийся на операционной системе хоста и отвечающий за создание, запуск и остановку работы контейнеров в этой системе.

3.5 **система управления контейнерами** (container management system): Программное обеспечение, осуществляющее управление и оркестрацию экземпляров контейнеров.

Примечание — Функциональные возможности включают в себя первоначальное создание и размещение, планирование, мониторинг, масштабирование, обновление и параллельное развертывание таких функциональных возможностей, как балансировщики нагрузки, межсетевые экраны, виртуальные сети и средства протоколирования.

3.6 **облачное приложение** (cloud native application): Приложение, специально разработанное для работы в облачных средах и использования всех их функциональных возможностей.

3.7 **функциональная декомпозиция** (functional decomposition): Тип модульной декомпозиции, в рамках которой система разделяется на отдельные компоненты, соответствующие определенным функциям и подфункциям системы.

Пример — Иерархическая декомпозиция, пошаговое улучшение.

3.8 **непрерывное развертывание** (continuous deployment): Подход в программировании, в рамках которого специалисты создают программное обеспечение в короткие циклы таким образом, что оно может быть в любое время опубликовано для промышленного использования, а сам процесс развертывания в производство автоматизирован.

3.9 **непрерывная доставка** (continuous delivery): Непрерывное развертывание, при котором этап развертывания инициируется вручную.

3.10 **оркестрация** (orchestration): Метод организации управления сервисами, в котором специализированный сервис координирует и управляет другими элементами, входящими в состав сервиса.

Примечание — Элемент, который управляет оркестрацией, не является частью самой оркестрации (экземпляром структуры).

3.11

образ виртуальной машины (virtual machine image): Файл (файлы), содержащий(ие) информацию о конфигурации, настройках и состоянии виртуальной машины, а также хранящиеся в ней программы и данные.

[ГОСТ Р 56938—2016, пункт 3.6]

3.12 **метаданные виртуальной машины** (virtual machine metadata): Информация о конфигурации и запуске виртуальной машины.

3.13 **микросервис** (microservice): Независимо развертываемый компонент, предоставляющий сервис для реализации конкретной функциональной части приложения.

3.14 **микросервисная архитектура** (microservices architecture): Подход к проектированию, который разделяет приложение на набор микросервисов.

3.15 **функции как услуга** (functions as a service): Категория облачного сервиса, в которой функциональная возможность, предлагаемая потребителю облачной службы, представляет собой выполнение кода приложения потребителя облачной службы в виде одной или нескольких функций, каждая из которых запускается событием, заданным потребителем облачной службы.

3.16 **бессерверная база данных** (serverless database): Категория облачных сервисов, в которой функциональная возможность, предлагаемая потребителю облачной службы, представляет собой базу данных, полностью управляемую поставщиком облачной службы и доступную через программный интерфейс приложения (API).

3.17 **межсетевой экран** (firewall): Программное и программно-техническое средство, реализующее функции контроля и фильтрации в соответствии с заданными правилами проходящих через него информационных потоков и используемое в целях обеспечения защиты (не криптографическими методами) информации, содержащей сведения, составляющие государственную тайну, иной информации ограниченного доступа.

3.18 **реестр контейнеров** (container registry): Компонент, обеспечивающий возможность хранения и доступа к образам контейнеров.

3.19 **близость ресурсов** (resource affinity): Размещение двух или более ресурсов в непосредственной близости друг от друга.

Примечание — Словосочетание «в непосредственной близости друг от друга» подразумевает такие факторы, как скорость доступа или высокая пропускная способность соединения между ресурсами.

4 Сокращения и обозначения

В настоящем стандарте применены следующие сокращения и обозначения:

ВМ — виртуальная машина;

ОС — операционная система;

ПО — программное обеспечение;

API — программный интерфейс приложения (application programming interface);

CMS — система управления контейнерами (container management system);

CSC — потребитель облачной службы (cloud service customer);

CSP — поставщик облачной службы (cloud service provider);

DDoS — отказ в обслуживании (distributed Denial of Service);

DevOps — методология автоматизации, объединяющая разработку программного обеспечения и промышленное использование с целью сокращения длительности операций в ходе жизненного цикла разработки и сопровождения;

DevSecOps — расширенная методология DevOps, включающая в себя функции обеспечения безопасности в качестве важной и неотъемлемой части процессов разработки и ИТ-операций;

DNS — служба доменных имен (domain name service);

DSaaS — хранение данных как услуга (data storage as a service);

Faas — функция как услуга (functions as a service);

GUI — графический интерфейс пользователя (graphical user interface);

HTTP — протокол передачи гипертекста (hypertext transfer protocol);

IaaS — инфраструктура как услуга (infrastructure as a service);

IP — интернет-протокол (internet protocol);

MAC — контроль доступа к носителям информации (media access control);

OCI — проект Linux Foundation для разработки открытых стандартов для виртуализации на уровне операционной системы, прежде всего контейнеров Linux (open containers initiative);

OVF — открытый стандарт для хранения и распространения виртуальных машин (open virtualization format);

PaaS — платформа как услуга (platform as a service);

SaaS — программное обеспечение как услуга (software as a service);

SOA — сервис-ориентированная архитектура (service-oriented architecture);

VPN — виртуальная частная сеть (virtual private network).

5 Обзор общих технологий и методов, используемых в сфере облачных вычислений

5.1 Общие положения

Настоящий стандарт содержит описание набора общих технологий и методов, используемых в сфере облачных вычислений.

Общая технология — это технология, которая используется для реализации одного или нескольких функциональных компонентов облачных вычислений, описанных в [1]. Общие технологии, как правило, являются частью облачного сервиса или применяются при использовании облачного сервиса.

Общий метод — это метод или подход к выполнению отдельных действий, связанных с облачными вычислениями (см. [1]). Общие методы, как правило, предназначены либо для того, чтобы упростить использование облачных сервисов, либо для того, чтобы обеспечить использование всех функциональных возможностей, предоставляемых облачными сервисами.

Многие из общих технологий и методов используются совместно в процессе разработки и работы облачных приложений.

5.2 Технологии

5.2.1 Общие положения

Общие технологии в основном относятся к виртуализации и контролю, а также управлению виртуализированными ресурсами в процессе разработки и работы облачных приложений. Облачное приложение — это приложение, специально разработанное для работы в облаке, а также для использования функциональных возможностей и среды облачных сервисов. Эти технологии касаются трех основных аппаратных ресурсов — ресурсов для обработки, хранения данных и сетевого подключения, а также платформенных функциональных возможностей облачного сервиса, определение которых приведено в [1], 9.2.4.2. Среди этих технологий имеются следующие:

- для виртуализированной обработки используются ВМ и гипервизоры (см. раздел 6), контейнеры (см. раздел 7), бессерверные вычисления (см. раздел 8);
- для виртуализированного хранения данных используются различные варианты хранения данных как услуги (Data Storage as a Service) (см. раздел 12);
- виртуализированные сети являются одной из основных групп технологий для предоставления и использования функциональных возможностей сетей применительно к облачным сервисам (см. раздел 13);
- категория облачных сервисов «Платформа как услуга» предназначена для более быстрой разработки, тестирования и выпуска облачных приложений (см. раздел 11).

Технологии обеспечения безопасности и масштабируемости применяются в целом ко всем типам облачных сервисов, хотя явное использование этих технологий потребителем сервиса облачных вычислений более характерно для отдельных типов облачных сервисов (см. разделы 14 и 15).

5.2.2 Тип инфраструктурных функциональных возможностей облачных сервисов

В рамках инфраструктурных функциональных возможностей облачных сервисов, как правило, используются следующие технологии:

- ВМ;
- контейнеры;
- виртуализированное хранилище;
- виртуализированные сети;
- системы безопасности.

5.2.3 Платформенные функциональные возможности облачных сервисов

В рамках платформенных функциональных возможностей облачных сервисов обычно используются следующие технологии:

- контейнеры;
- бессерверные вычисления;
- облачные сервисы PaaS;
- виртуализированное хранилище;
- виртуализированные сети;
- системы безопасности.

5.2.4 Прикладные функциональные возможности облачных сервисов

В рамках прикладных функциональных возможностей облачных сервисов обычно используются следующие технологии:

- виртуализированное хранилище;
- виртуализированные сети;
- система безопасности.

5.3 Методы

Общие методы, как правило, применяются ко всем категориям облачных сервисов, хотя отдельные методы могут более эффективно работать с одними категориями облачных сервисов, чем с другими.

Оркестрация и управление виртуализированными ресурсами осуществляются с помощью соответствующего инструментария, включая CMS (см. раздел 10 и 7.4).

Как правило, в облачных вычислениях используют следующие методы:

- различные виды автоматизации, применяемые во всех процессах DevOps (см. раздел 10);
- подходы к обеспечению масштабируемости, такие как параллельные экземпляры (см. раздел 14);
- подход к проектированию приложений и систем на основе микросервисов (см. раздел 9);
- межсетевые экраны, шифрование и методы управления идентификацией и доступом (IAM) для обеспечения информационной безопасности (см. раздел 15).

6 Виртуальные машины и гипервизоры

6.1 Общие положения

ВМ и гипервизоры представляют собой технологии, которые обеспечивают виртуализированную обработку (также известную как виртуализированные вычисления) для облачных сервисов. Эти технологии в первую очередь относятся к облачным сервисам типа инфраструктурных функциональных возможностей и IaaS (инфраструктура как услуга) согласно описанию в ГОСТ ISO/IEC 17788 (см. также [1]).

Одной из ключевых особенностей облачных вычислений является возможность совместного использования ресурсов. Это имеет важное значение не только для сокращения расходов, но и для поддержания на высоком уровне таких рабочих характеристик, как масштабируемость и устойчивость. Совместное использование вычислительных ресурсов требует определенного уровня виртуализации. Виртуализация в целом означает, что определенный ресурс становится доступным для использования в форме, которая физически не существует как таковая, но которая может использоваться с помощью программного обеспечения. Другими словами, виртуализация обеспечивает абстрактное представление базового ресурса, который преобразуется в программно-определяемую форму для использования другими программными объектами. ПО, выполняющее виртуализацию, позволяет нескольким пользователям совместно и одновременно использовать один физический ресурс, не влияя на работу друг друга и, как правило, не зная друг о друге (см. [2], 5.5).

Одним из подходов к виртуализации вычислительных ресурсов является использование ВМ, что предполагает наличие гипервизора, обеспечивающего абстрактное представление аппаратных ресурсов вычислительной системы и позволяющего нескольким ВМ работать в одной физической системе. При этом каждая ВМ имеет собственную гостевую ОС, как показано на рисунке 1. Это позволяет использовать ресурсы одной системы несколькими приложениями, работающими в разных ВМ.

Гипервизор представляет собой, как правило, ПО, которое устанавливается и управляется CSP. Облачный сервис, на базе которого работает ВМ, предоставляет возможность CSU загружать ПО из образа виртуальной машины и запускать его на ВМ в системе CSP. ВМ управляется гипервизором, но CSU этого не видит.

6.2 Виртуальные машины и виртуализация систем

ВМ представляет собой изолированную среду выполнения ПО, которая использует виртуализированные физические вычислительные ресурсы. Другими словами, речь идет о виртуализации системы. ПО в каждой ВМ получает тщательно контролируемый доступ к физическим ресурсам, чтобы обеспечить совместное использование этих ресурсов. ВМ, которые иногда называются системными ВМ, предлагают широкую функциональность, необходимую для выполнения полных стеков ПО, включая целые ОС и код приложений, использующих ОС. Как показано на рисунке 1, это отражается в виде элементов «Гостевая ОС» и «Приложение N» в каждой ВМ.

Цель ВМ — обеспечить одновременную работу нескольких приложений в одной аппаратной системе. При этом эти приложения остаются изолированными друг от друга. ПО, работающее в каждой ВМ, имеет свое собственное системное аппаратное обеспечение, такое как процессор, оперативная память, одно или несколько устройств для хранения данных и сетевое оборудование. Изолированность означает, что ПО, работающее в одной ВМ, отделено от ПО, работающего в других ВМ в той же системе, и это ПО никак не пересекается друг с другом. Также оно отделено от ОС хоста. Виртуализация обычно означает, что часть доступных физических вычислительных ресурсов может быть предоставлена каждой ВМ, например: ограниченное количество процессоров, ограниченный объем оперативной памяти, ограниченное пространство для хранения данных и контролируемый доступ к сети.

Каждая ВМ содержит полный стек ПО, начиная с операционной системы и заканчивая любым другим ПО, необходимым для работы одного или более приложений, выполняемых в ВМ. Программный стек может быть очень простым (например, собственное приложение, написанное на языке программирования C, использующее только функции, предоставляемые самой ОС) или сложным [например, приложение, написанное на языке программирования Java™, для которого требуется среда выполнения и которое активно использует библиотеки и (или) сервисы, которых нет в операционной системе и которые должны предоставляться отдельно].

Каждая ВМ может в принципе содержать любую ОС. Различные ВМ на одной аппаратной системе могут работать под управлением совершенно разных ОС, таких как Linux и Windows. Единственное требование заключается в том, что все ПО, выполняемое в ВМ, должно быть разработано для аппаратной архитектуры именно этой системы, поскольку аппаратное обеспечение виртуализируется, а не эмулируется. Так, например, программный код, созданный для архитектуры ARM, не будет работать в ВМ, запущенной в системе на базе архитектуры Intel x86.

6.3 Гипервизоры

6.3.1 Общие положения

Гипервизоры, которые также иногда называются мониторами ВМ, представляют собой ПО, которое виртуализирует физические ресурсы и позволяет запускать ВМ. Виртуализация означает управление абстрактным представлением физических ресурсов системы. Гипервизоры также управляют работой ВМ. Гипервизор распределяет ресурсы для каждой запущенной ВМ, включая процессор, оперативную память, систему хранения данных и сетевые компоненты, а также пропускную способность.

Гипервизоры бывают двух типов:

- автономные (тип I);
- на основе базовой ОС (тип II).

Гипервизоры типа I могут работать быстрее и эффективнее, поскольку для них не требуется ОС хоста. Гипервизоры типа II могут работать медленнее, но они имеют ряд преимуществ: отличаются простотой настройки и совместимы с большим количеством аппаратного обеспечения, чем гипервизоры типа I. Это связано с тем, что для гипервизоров типа I необходимо учитывать особенности аппаратного

обеспечения, в то время как гипервизоры типа II используют преимущества аппаратной поддержки, встроенной в ОС хоста.

6.3.2 Гипервизоры типа I

Гипервизоры типа I работают непосредственно на основе аппаратного обеспечения системы и напрямую управляют им, так же как и ВМ. Принцип организации системы, использующей гипервизор типа I, показан на рисунке 1.

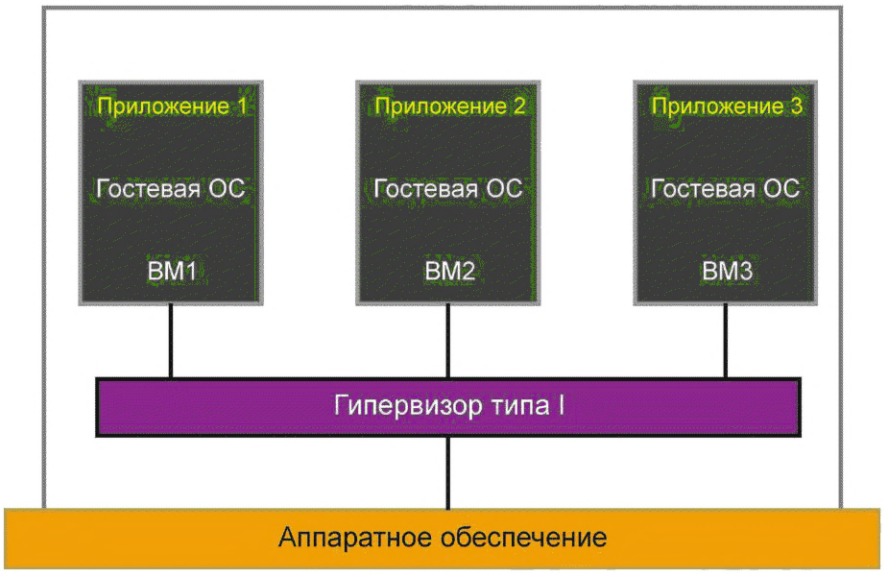

Рисунок 1 — Виртуализация аппаратного обеспечения гипервизором типа I

6.3.3 Гипервизоры типа II

Гипервизоры типа II работают поверх ОС хоста, а точнее, ядра ОС хоста. Именно ОС хоста управляет аппаратным обеспечением системы, а гипервизор использует ее функциональные возможности для работы и управления ВМ. Принцип организации системы, использующей гипервизор типа II, показан на рисунке 2.

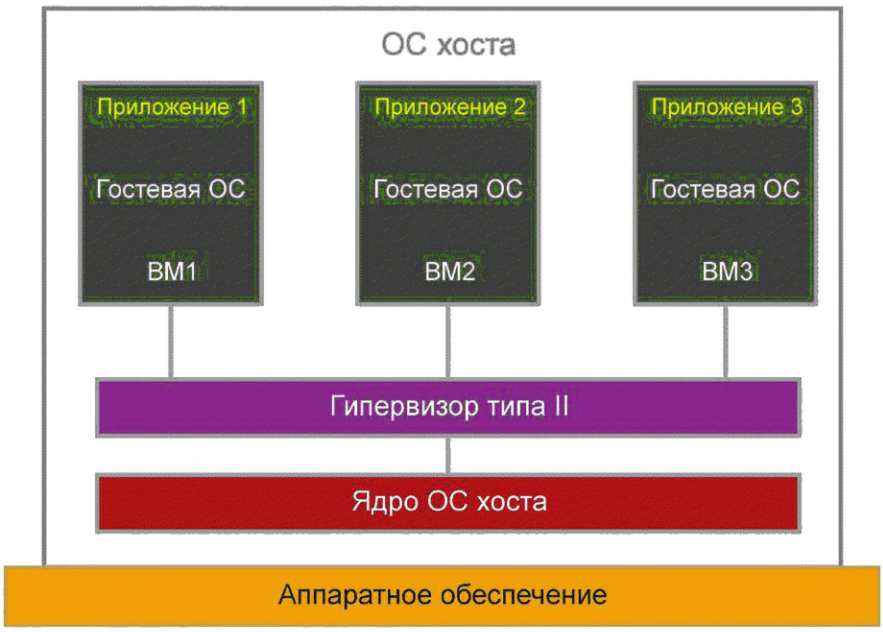

Рисунок 2 — Виртуализация аппаратного обеспечения гипервизором типа II

6.4 Безопасность ВМ и гипервизоров

Для аппаратных систем операционная система имеет самый высокий уровень привилегий, поскольку она должна контролировать доступ ко всем аппаратным ресурсам. Однако в случае с хостом гипервизора последний должен контролировать весь доступ гостевых ВМ к ресурсам процессора и оперативной памяти (обеспечивая виртуализацию процессора и памяти), поэтому он должен работать на более высоком уровне привилегий, чем все ВМ. Для этого гипервизоры устанавливаются на аппаратных системах, которые помогают выполнять виртуализацию. В частности, аппаратная система имеет два состояния процессора: режим root (гипервизор) и режим non-root (гость). Все гостевые ОС работают в режиме non-root, в то время как гипервизор работает в режиме root.

Несмотря на аппаратную поддержку виртуализации, изоляция процессов среды выполнения для ВМ, обеспечиваемая гипервизором, может быть нарушена неавторизованными или скомпрометированными ВМ, получившими доступ к областям оперативной памяти, принадлежащим гипервизору или другим ВМ. Неавторизованные или скомпрометированные ВМ используют определенные уязвимости гипервизоров в отношении определенных программных структур, таких как блок управления ВМ (VMCB) и таблицы страниц виртуальной памяти, которые используются гипервизором для отслеживания состояния выполнения ВМ и отображения памяти с адресов ВМ на адреса памяти хоста соответственно. Об этих уязвимостях гипервизоров было известно уже давно, поэтому многие из них уже устранены или будут устранены в скором будущем. Более поздние версии гипервизоров были обновлены для более надежной работы. CSC и CSP должны убедиться в том, что все используемые гипервизоры обновлены и защищены от известных уязвимостей.

Еще одна проблема безопасности в платформе хоста гипервизора может быть связана с ПО, используемым для виртуализации устройств. В отличие от виртуализации набора команд и памяти виртуализация устройств осуществляется не непосредственно гипервизором, а с помощью вспомогательных программных модулей. Основные источники возникновения уязвимостей включают в себя: а) программный код, который эмулирует физические аппаратные устройства, запущенные в гипервизоре в качестве загружаемого модуля ядра, и б) драйверы устройств с прямым доступом к памяти (DMA), которые могут обращаться к областям памяти, принадлежащим другим ВМ или даже гипервизору.

Одними из возможных негативных последствий получения неавторизованной ВМ контроля над гипервизором являются: установка руткитов или атаки на другие ВМ на том же хосте гипервизора. Перед установкой и использованием в системе, в которой присутствуют гипервизор и ВМ, все ПО для виртуализации устройств должно быть проверено на наличие уязвимостей.

6.5 Образы ВМ, метаданные и форматы

Образ ВМ представляет собой пакет данных, содержащий информацию и исполняемый код, необходимые для запуска экземпляра ВМ. Образ ВМ используется для создания нового экземпляра ВМ в случае необходимости. Образ ВМ может включать в себя полный стек ПО, необходимый для работы приложения, начиная с операционной системы, библиотек, сред выполнения, программного кода самого приложения, конфигурационных файлов и других метаданных, используемых приложением. Образ ВМ может также включать в себя метаданные, связанные с созданием самой ВМ.

Метаданные ВМ содержат информацию о конфигурации и запуске ВМ. Сюда могут входить такие свойства ВМ, как объем оперативной памяти, требования к процессору и так далее. Метаданные ВМ также обычно ссылаются на образы дисков, содержащиеся в образе ВМ, в частности, указывая, как они развернуты в экземпляре ВМ.

Принцип организации образа ВМ заключается в том, что он должен содержать все объекты, необходимые для работы экземпляра ВМ, где образ ВМ используется в качестве входных данных для гипервизора для создания и запуска ВМ. В целом образ ВМ состоит из двух наборов данных: во-первых, метаданных ВМ и, во-вторых, образов дисков. Важно понимать, что существует множество различных форматов как метаданных ВМ, так и образов дисков. Конкретный гипервизор, используемый для создания ВМ, может понимать только определенные форматы метаданных ВМ и образов дисков.

Образы ВМ основаны на данных, хранящихся в файлах. Файлы находятся в файловых системах, которые хранятся в образе ВМ в виде одного или нескольких образов дисков. Эти файлы могут быть файлами операционной системы, приложения и любой другой части программного стека. Существует как минимум один образ диска, но может быть несколько образов диска, если такая организация файлов используется приложением и его программным стеком. Как правило, объем данных, хранящихся в образах дисков, очень большой, и поэтому форматы, используемые для хранения данных, предполагают использование технологии сжатия данных в той или иной форме.

Существует множество используемых форматов образов ВМ и дисков, из которых значительная часть является чьей-то собственностью, а другая часть имеет открытый исходный код. Примеры форматов образов ВМ и дисков, созданных на базе отраслевых стандартов:

- OVF (см. ГОСТ Р ИСО/МЭК 17203 и [3]). Пакет OVF состоит из нескольких файлов, размещенных в одном каталоге. Существует файл дескриптора OVF (с расширением .ovf) с данными в формате XML, описывающими ВМ в пакете, включая метаданные, такие как имя, требования к аппаратному обеспечению и ссылки на другие файлы в пакете. Пакет OVF также содержит один или несколько образов дисков, а также некоторые дополнительные файлы, например файлы сертификатов. Формат образов OVF относительно широко поддерживается как с помощью инструментов импорта/экспорта, так и без их использования;

- формат диска ИСО — формат архива, используемый для данных, записываемых на оптические диски (см. [3] и [4]). Стандартом файловой системы для оптических дисковых носителей является [3], в основном компакт-дисков. В форматах для DVD-дисков и дисков Blu-ray (BD), как правило, применяют [4] (также известный как Universal Disk Format или UDF), идеально подходящий для (пере)записываемых оптических носителей.

Образы дисков обычно сжимаются из-за их большого размера, хотя бывают случаи, когда используются необработанные несжатые образы дисков для обеспечения более высокой скорости при запуске ВМ за счет использования большего пространства.

Форматы образов ВМ и образов дисков, которые поддерживаются конкретным гипервизором, указаны в документации к гипервизору.

7 Контейнеры и системы управления контейнерами (CMS)

7.1 Общие положения

Контейнеры представляют собой технологию, обеспечивающую виртуализированную обработку данных для облачных сервисов. Эта технология относится как к типу инфраструктурных функциональных возможностей, так и к типу платформенных функциональных возможностей облачных сервисов согласно описанию в ГОСТ ISO/IEC 17788 (см. также [1]).

Контейнеры обеспечивают среду выполнения ПО за счет виртуализации ядра операционной системы, установленной в системе. Контейнеры представляют собой еще один подход к предоставлению среды выполнения ПО с помощью виртуализации вычислительных ресурсов. Контейнеры подразумевают виртуализацию ядра ОС, в отличие от виртуализации аппаратного обеспечения, как в случае с ВМ. Цель контейнеров — обеспечить нескольким различным наборам ПО возможность работать одновременно в одной системе, не мешая друг другу, благодаря безопасному разделению ресурсов системы.

Облачный сервис, поддерживающий контейнеры, позволяет CSU загружать ПО из образа контейнера и запускать его в контейнере в системе CSP. Управление контейнером осуществляет либо CSP, либо CSU, в зависимости от типа функциональных возможностей облачного сервиса. В любом случае, как правило, управление осуществляется с помощью CMS (см. 7.4).

7.2 Контейнеры и виртуализация операционных систем

7.2.1 Описание контейнеров

Контейнер представляет собой изолированную среду выполнения для работы ПО, использующую ядро виртуализированной ОС. Контейнеры работают в ОС, которая называется ОС хоста.

Как указано в 6.2, ВМ представляет собой виртуализированную версию аппаратного обеспечения для ПО, находящегося внутри ВМ. Доступ к виртуализированным аппаратным ресурсам контролируется, и ПО внутри ВМ получает возможность видеть и использовать только контролируемую и ограниченную версию этих ресурсов (например, ограниченное количество процессоров, ограниченное количество вычислительных потоков, ограниченный объем оперативной памяти).

Аналогичным образом контейнер представляет собой виртуализированную версию ядра ОС хоста. Доступ к виртуализированным ресурсам ядра ОС контролируется, и ПО внутри контейнера получает возможность видеть и использовать только контролируемые и ограниченные ресурсы ОС.

Изоляция среды выполнения означает, что ПО, работающее в одном контейнере, отделено от ПО, выполняемого в других контейнерах, не пересекается с ним, а также отделено от ОС хоста. Единственное ПО, работающее вне контейнера, которое может получить доступ или повлиять на работу ПО, работающего внутри контейнера, это контейнерная служба.

На рисунке 3 показаны три контейнера, работающие на базе аппаратного обеспечения системы. Физическая система имеет свою собственную ОС хоста. Каждый контейнер содержит собственное прикладное ПО (приложение x) и запускает это ПО в одном или нескольких процессах ОС, используя такие ресурсы, как оперативная память, процессор, система хранения данных и сетевые компоненты, изолированно от других контейнеров, работающих на той же системе, но все они совместно используют ядро ОС хоста.

Ядро ОС хоста используется всеми контейнерами совместно, поэтому ОС, используемая ПО в контейнерах, должна быть совместима с ядром ОС хоста. Благодаря этому различные контейнеры смогут использовать различные варианты ОС Linux, если ОС хоста является, например, одной из версий ОС Linux, но ОС Windows невозможно будет использовать в контейнере, если ОС хоста является одной из версий ОС Linux (и наоборот).

Контейнеры создаются и управляются с помощью контейнерной службы, которая запускается как отдельный процесс в ОС хоста.

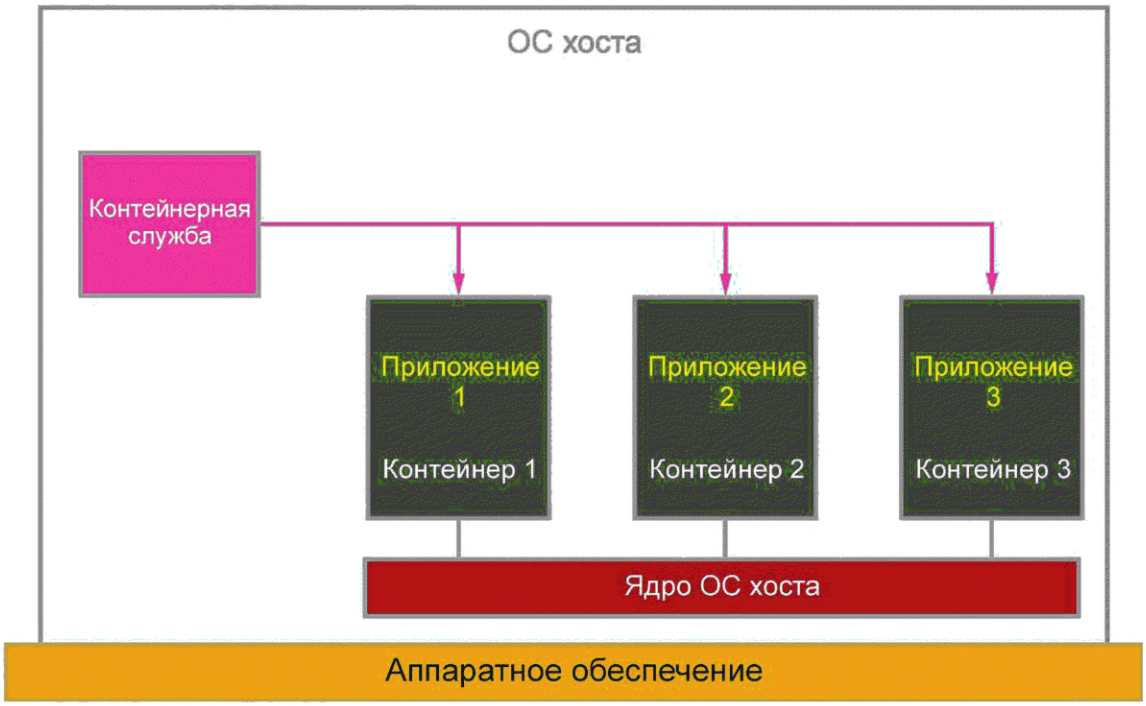

Рисунок 3 — Виртуализация контейнеров

Программный стек, работающий в каждом отдельном контейнере, может быть разным, но обычно он содержит само приложение (приложение x) и любые программные зависимости, которые есть у приложения. В принципе, программный стек может быть достаточно скромным, особенно если программный код приложения зависит только от функций ядра ОС хоста. Однако бывает, что код контейнера может включать в себя элементы ОС за пределами ядра, такие как библиотеки и утилиты, особенно если программный код приложения зависит от конкретных версий этих библиотек и утилит.

ОС хоста, используемая контейнерами, может работать внутри ВМ, а не на базе физического аппаратного обеспечения. ПО, работающее в контейнерах, не знает, запущена ли ОС хоста на ВМ или нет.

7.2.2 Контейнерная служба

Контейнерная служба представляет собой программный сервис, который выполняется на ОС хоста и отвечает за создание контейнеров в этой системе и управление ими. Конкретный контейнер — это исполняемый экземпляр программного стека, который содержится в образе контейнера (см. 7.3 для получения дополнительной информации об образах контейнеров). В образ контейнера входят метаданные и параметры, используемые контейнерные службы. Контейнерная служба использует метаданные контейнера для указания определенных функциональных возможностей контейнера, а также использует параметры сервиса контейнера, чтобы определить, как создается и работает контейнер.

Контейнерная служба предлагает сервисный интерфейс, через который к его функциональным возможностям могут обращаться клиентские приложения. Клиентские приложения могут работать в той же системе, что и контейнерная служба, либо в удаленных системах и обращаться к функциональным возможностям контейнерной службы через сеть.

Контейнерная служба поддерживает следующий набор операций с контейнерами:

- Create: эта операция создает новый контейнер. Эта операция ссылается на используемый образ контейнера, созданный в доступном ОС хоста каталоге файловой системы (называемом bundle). При создании контейнера выделяется набор ресурсов для него и выполняются соответствующие настройки согласно метаданным контейнера, хранящимся в пакете. Контейнеру присваивается уникальный идентификатор, по которому на него впоследствии можно ссылаться;
- Start: эта операция запускает контейнер путем выполнения прикладной программы, заданной для контейнера, с любыми параметрами согласно его метаданным;
- Kill: эта операция останавливает программу(ы), запущенную(ые) в контейнере. Как правило, путем отправки определенного сигнала процессу, запущенному в контейнере;
- Delete: эта операция удаляет ресурсы, выделенные контейнеру, и уничтожает контейнер. Уникальный идентификатор больше не идентифицирует контейнер, хотя этот же идентификатор может быть использован позже для создания нового контейнера.

Контейнерная служба обычно также обеспечивает интерфейс событий, который позволяет ему сообщать о важных событиях, связанных с управляемыми им контейнерами. Интерфейс событий позволяет одному или нескольким компонентам клиентского ПО отслеживать определенные события и реагировать на них.

7.2.3 Ресурсы контейнеров, изоляция и контроль

Контейнер обеспечивает изолированную среду выполнения ПО, которая контролирует ресурсы.

Изоляция означает, что, с точки зрения ПО, работающего внутри контейнера, все ресурсы системы полностью принадлежат ему и единственный выполняемый процесс (или процессы) запущен внутри контейнера. В действительности в той же системе может работать множество других процессов, но ПО внутри контейнера никак с ними не пересекается и не может с ними взаимодействовать.

Контроль ресурсов означает, что набор ресурсов, доступных ПО в контейнере, выделяется контейнеру (контейнерной службе) при его создании, причем эти ресурсы являются ограниченными и отслеживаются. Сюда входит выделение ресурсов процессора, рабочей памяти, сетевых ресурсов, одной или нескольких файловых систем. С точки зрения ПО, существуют только эти ресурсы. Ресурсы распределяются таким образом, что ресурсы, выделенные одному контейнеру, не могут взаимодействовать с ресурсами, выделенными другим контейнерам, работающим на той же системе.

Изоляция и контроль ресурсов обеспечиваются с помощью функциональных возможностей ОС хоста, которые используются и управляются с помощью контейнерной службы. Конкретные функциональные возможности, доступные для контейнеров и используемые ими, зависят от ОС. Функциональные возможности, доступные в ОС Linux, описаны в настоящем стандарте в качестве примера. Дополнительная информация об аналогичных функциональных возможностях других систем приведена в соответствующей документации по этим системам.

Для доступа к блочному вводу/выводу контейнер по умолчанию имеет доступ к файловой системе, состоящей из образа контейнера, использованного для создания контейнера в режиме «только для чтения», плюс уровень контейнера для чтения/записи. Файловая система по умолчанию является временной, и уровень контейнера удаляется при удалении контейнера. Дополнительные, обычно постоянные, ресурсы для хранения данных могут стать доступны ПО внутри контейнера за счет настройки конфигурации контейнера контейнерной службой — либо в виде дополнений файловой системы внутри контейнера, внесенных из другого места вне контейнера, либо посредством предоставления одного или нескольких сервисов хранения (как правило, облачных). Такие дополнительные средства хранения необходимы, если ПО в контейнере должно иметь возможность обращаться к объектам хранения, имеющим длительный жизненный цикл. Во всех случаях видимое расположение файлов и объектов хранения внутри контейнера сопоставляется с реальным расположением за пределами контейнера.

Аналогичным образом для доступа к сетевым функциональным возможностям ресурсы, доступные для программного кода, работающего в контейнере, контролируются с помощью контейнерной службы и конфигурации, применяемой к контейнеру, т. е. сетевые функциональные возможности могут включаться/выключаться и настраиваться. Можно не предоставлять контейнеру никаких сетевых функциональных возможностей (т. е. никаких открытых портов, никаких доступных сетевых устройств,

в связи с чем пути к конечным сетевым устройствам тоже отсутствуют). Также можно детально контролировать доступ к контейнеру и доступ из контейнера.

Порты, открытые контейнером, можно контролировать и сопоставлять с сетевыми адресами и портами, видимыми извне. Например, контейнер может открыть порт 80 для HTTP-трафика, но он может быть сопоставлен с портом 8080 для внешнего доступа (например, интернета). В целом можно контролировать и сопоставлять IP-адреса, порты, имена хостов, MAC-адреса, службы маршрутизации и службы DNS.

Одна из важных форм сетевого взаимодействия, которая используется с контейнерами, — это когда сеть объединяет исключительно набор связанных контейнеров, например контейнеры, которые представляют собой единое приложение, реализованное с использованием архитектуры микросервисов, причем различные компоненты приложения выполняются в разных контейнерах. Это форма виртуальной сети, когда только назначенные контейнеры могут общаться друг с другом (кроме любых конкретных внешних конечных устройств), как если бы они были единственными объектами в сети. В сеть могут входить различные системы, а также различные другие сети, обеспечивая большую гибкость в месте расположения каждого контейнера, т. е. обеспечивается прозрачность расположения контейнера. При этом сохраняется возможность контроля и ограничения обмена данными в целях безопасности.

В ОС Linux контроль над ресурсами осуществляется с помощью функциональной возможности, называемой контрольными группами или cgroups. Она обеспечивает контроль над ресурсами, доступными наборам процессов, включая процессор, память, ввод/вывод на блочные устройства (т. е. файловые системы), доступ к устройствам, сетевое подключение.

В ОС Linux изоляция реализуется с помощью пространств имен. Таким образом, любые ресурсы, к которым обращается один контейнер, являются частью одного пространства имен, в то время как ресурсы, к которым обращаются другие контейнеры, относятся к другим пространствам имен. Пространства имен работают таким образом, что ПО, работающее в процессе, который был запущен в одном пространстве имен, может «видеть» только ресурсы в этом пространстве имен.

В ОС Linux существуют следующие виды пространств имен (начиная с ядра Linux 4.10):

- Interprocess Communication (ipc): относится к межпроцессному взаимодействию. Только процессы в одном и том же пространстве имен могут обмениваться данными (например, выделять общую память);
- Mount (mnt): относится к точкам монтирования, т. е. местам, куда монтируются (дополнительные) файловые системы. Начальный набор для монтирования доступен при создании контейнера контейнерной службой, но после этого любые новые монтирования будут видны только внутри контейнера;
- Network (net): содержит ресурсы, связанные с сетью, такие как интерфейсы, IP-адреса, таблица маршрутизации, список сокетов, таблица отслеживания соединений и т. д.;
- Process ID (pid): содержит набор идентификаторов процессов; первый процесс в пространстве имен имеет идентификатор 1, и этот процесс имеет специальный режим, аналогичный процессу init в базовой ОС;
- User ID (user): предоставляет идентификаторы пользователей, позволяющие как идентифицировать пользователя, так и контролировать привилегии: пространство имен пользователей сопоставляет идентификаторы пользователей в пространстве имен с идентификаторами пользователей в базовой системе, что позволяет тщательно контролировать привилегии и обеспечить более высокие привилегии в пространстве имен, которые не предоставляются для каких-либо ресурсов вне этого пространства имен;
- UTS: дает возможность различным процессам иметь разные имена хостов и доменов.

Совместное использование cgroups и пространств имен обеспечивает контроль ресурсов и изоляцию, необходимую для контейнеров.

7.3 Образы контейнера и уровни файловой системы

7.3.1 Назначение и содержание образа

Образ контейнера представляет собой исполняемый пакет, который содержит все необходимое для работы ПО, например приложение или микросервис. Это может быть программный код самого приложения, среда выполнения, библиотеки, переменные среды, файлы конфигурации и другие метаданные, используемые приложением. Цель состоит в том, чтобы образ контейнера описывался самостоятельно и был инкапсулированным, в результате чего у контейнерной службы появилась бы возможность заимствовать образ контейнера и создать из него контейнер без дополнительных зависимостей

от базовой системы (независимо от инфраструктуры) и независимо от содержимого образа контейнера (независимо от содержимого).

Образ контейнера содержит наборы файлов, которые содержат программный код приложения, его зависимости и другие файлы и метаданные, используемые приложением. Образ контейнера также содержит структурированные метаданные о содержимом образа контейнера и о том, как преобразовать это содержимое в контейнер. Метаданные контейнера могут варьироваться в зависимости от конкретного формата образа контейнера. Например, они могут включать в себя следующее:

- указатель образа. Метаданные «верхнего уровня», которые предназначены для образов контейнеров, поддерживающих несколько различных платформ (иногда их называют fat-манифестом). Если поддерживается несколько платформ, для каждой платформы существует свой собственный образ, содержащий программные компоненты, которые необходимо использовать при работе контейнера на этой платформе. По сути, указатель образа ссылается на один или несколько манифестов образов;
- манифест образа. Содержит информацию одного образа контейнера для конкретной архитектуры процессора и ОС, состоящего из конфигурации и набора уровней;
- расположение образа. Конкретное расположение каталогов и файлов внутри образа с метаданными об уровнях файловой системы;
- уровни файловой системы. Одна или несколько упорядоченных файловых систем (т. е. структура каталогов и файлов) и изменения в файловой системе (удаленные или обновленные файлы). Уровни накладываются друг на друга для создания полной файловой системы в работающем контейнере (см. 7.3.2 для получения дополнительной информации об уровнях файловой системы).

Функциональные возможности указателя образа и fat-манифестов позволяют одному образу контейнера обеспечить поддержку образов, характерных для конкретной платформы. Платформа может содержать тип процессора (архитектура ПК), тип ОС и ее уровень. Таким образом, один образ контейнера может быть структурирован так, чтобы обеспечить доставку и развертывание одного и того же приложения на нескольких различных целевых системах.

Одно из характерных свойств метаданных контейнера, содержащихся в образах контейнеров, заключается в том, что они предоставляют расширенные функции обеспечения безопасности, призванные предотвратить подделку содержимого образа контейнера с момента его создания. Записывается длина данных, а также их дайджесты (по сути, это устойчивый к конфликтам криптографический хеш байтов данных). Соответствующими данными могут быть содержимое уровней файловой системы или элементы метаданных. Дайджест может также служить уникальным идентификатором содержимого, который также может использоваться для доступа к данным с возможностью адресации содержимого. Отдельная безопасная передача дайджеста пользователю образа контейнера позволяет проверить содержимое образа контейнера, даже если оно получено из ненадежного источника.

7.3.2 Многоуровневая файловая система

Образы контейнеров и контейнеры, созданные на их основе, используют технологию многоуровневой файловой системы при работе с файлами, которые они содержат.

Многоуровневая файловая система представляет собой подход к созданию содержимого файловой системы, используемой контейнером. Принцип заключается в том, что содержимое файловой системы строится как набор уровней, каждый из которых содержит определенный набор каталогов и файлов, и все из них имеют общий корневой каталог. Каталоги и файлы передаются с каждого уровня по очереди, начиная с базового нижнего уровня и продвигаясь к верхним. Каждый последующий уровень может не только добавлять новые файлы, но также может заменять файл нижнего уровня файлом другой версии или даже удалять («уничтожать») файл нижнего уровня.

Наличие нескольких уровней в файловой системе позволяет эффективно работать с файлами в образах контейнеров. Такой принцип организации файлов также является хорошим решением для создания образов контейнеров, учитывая, что приложения, как правило, в качестве зависимостей используют программные стеки, которые позволяют их запускать.

В качестве примера рассмотрим приложение node.js. Программный код приложения node.js может находиться в файле сценария app.js, а также в других файлах сценария, файлах данных и файлах конфигурации. Приложение node.js имеет зависимость от среды выполнения node.js и ряда (внешних) пакетов. В свою очередь, среда выполнения node.js зависит от различных библиотек ОС.

В образе контейнера для приложения node.js это может быть отображено с помощью трех уровней (начиная с самого верхнего):

- сценарий app.js и связанные с ним программные компоненты составляют самый верхний уровень;

- среда выполнения node.js и связанные с ней пакеты образуют следующий уровень;
- библиотеки ОС образуют нижний уровень.

Следует отметить, что ядро ОС не обязательно должно присутствовать в образе контейнера. Ядро ОС предоставляется ОС хоста, на которой работают контейнеры.

Разделение на уровни позволяет организовать эффективный процесс построения (создания) образов контейнеров. Можно создать образ контейнера с одним уровнем, содержащим все необходимые файлы, но гораздо эффективнее будет разделить программный стек, используемый приложением, на отдельные уровни. Это связано с тем, что приложение и все его зависимости обычно представляют собой отдельные независимые наборы файлов, что описывается в примере на основе приложения node.js.

Образ контейнера может быть создан с использованием другого образа контейнера в качестве основы (или родительского объекта). Поэтому, если снова обратиться к примеру с приложением node.js, первым созданным образом контейнера может быть образ для ОС. Затем можно создать второй образ контейнера для среды выполнения node.js и связанных с ней пакетов, используя образ ОС в качестве родительского объекта. Наконец, третий образ контейнера может быть создан для приложения app.js с использованием в качестве родительского объекта среды выполнения node.js. Родительские образы создают нижние уровни для нового образа, построенного поверх них. Поэтому в этом примере библиотеки ОС становятся самым нижним уровнем, среда выполнения node.js — средним уровнем, а приложение app.js — самым верхним уровнем.

Такая модель организации позволяет каждому образу решать только свои собственные задачи. Например, если среде выполнения node.js не нужны все файлы из библиотек ОС, она может удалить ненужные файлы. Таким образом, при создании образа контейнера основной вопрос заключается в том, какой (какие) базовый(ые) образ(ы) использовать.

Многоуровневая организация файловой системы также применяется и к контейнерам, но с одним отличием. Когда контейнер создается из образа контейнера, формируются те же уровни файловой системы, что и в образе контейнера, но они рассматриваются как доступные только для чтения. Эти уровни называются уровнями образа. Приложение, работающее в контейнере, не может изменять файлы на уровнях образа. Однако поверх уровней, присутствующих в образе контейнера, добавляется дополнительный записываемый уровень — он называется уровнем контейнера (в отдельных контейнерных средах он называется уровнем «песочницы»). Все изменения в файлах, внесенные работающим в контейнере приложением, записываются на уровне контейнера, будь то создание новых файлов, изменение существующих или удаление ненужных файлов. Это подразумевает принцип копирования при записи для файлов в контейнере.

В результате использования уровней образа, предназначенных только для чтения, и принципа копирования при записи уровни образа могут совместно использоваться различными контейнерами. Это позволяет экономить на хранении данных и памяти во время выполнения и сократить время запуска контейнеров.

7.3.3 Репозитории и реестры образов контейнеров

Возможность хранения и получения доступа к образам контейнеров является ключевой особенностью экосистемы контейнеров. Отдельные образы контейнеров обычно используются в нескольких различных системах, например, для поддержания достаточной масштабируемости и обеспечения резервирования для повышения доступности ресурсов. В рекомендуемом процессе создания образов особое внимание уделяется возможности доступа к существующим образам, которые являются родительскими объектами для создаваемого образа.

В экосистеме контейнеров рекомендуется повторное использование ресурсов. Образы контейнеров для общих элементов программного(ых) стека(ов), используемого(ых) приложениями, применяются в качестве родительских образов для многих приложений. Типичными примерами таких образов являются образы для библиотек ОС и образы для промежуточного ПО и сред выполнения. Весьма вероятно, что образы контейнеров для этих программных пакетов будут (повторно) использоваться снова и снова в образах контейнеров для приложений, которые применяют эти программные пакеты в своих программных стеках.

Как правило, лучше и дешевле повторно использовать образ контейнера, созданный экспертом в области данного программного пакета, чем создавать новый образ для этого программного пакета. Кроме того, такие образы обычно обновляются по мере внесения исправлений в ПО.

За обеспечение возможности хранения и получения доступа к образам контейнеров отвечает реестр контейнеров. Реестры контейнеров могут предоставляться как общедоступные облачные сервисы

или как частные облачные сервисы. Примером общедоступного облачного сервиса для предоставления реестра контейнеров является Docker Hub, доступный по адресу: gost.e-ecolog.ru . Реестры контейнеров имеют сервисные интерфейсы, которые обеспечивают по крайней мере возможность выполнения операций push и pull. Операция push закачивает один или несколько образов в реестр, а операция pull скачивает один или несколько образов из реестра.

Репозиторий представляет собой набор связанных образов контейнеров. Примером набора связанных образов контейнеров является набор образов контейнеров для библиотек конкретной ОС, где каждый образ предназначен для определенной версии этой ОС.

Например, репозиторий контейнеров может содержать набор из четырех образов контейнеров с именами some_os_libs:16.01, some_os_libs:16.02, some_os_libs:16.03, some_os_libs:latest. В этом случае в репозитории имеются четыре записи для разных версий ОС, каждая из которых имеет метку номера версии. Версия с меткой «latest» (последняя) указывает на тот же образ контейнера, что и версия с меткой «16.03».

Это объясняется тем, что, если другим образам контейнеров необходимо всегда использовать последнюю версию образа контейнера ОС в качестве родительского элемента, они могут использовать метку «latest» при получении образа, что приведет к автоматическому обновлению, когда новые образы контейнеров более поздних версий ОС будут загружены в реестр контейнеров. Другой областью применения меток, используемых в репозиториях образов, является создание образов для конкретных целевых сред. В качестве альтернативного метода можно использовать образы fat-манифестов.

7.4 Системы управления контейнерами

7.4.1 Общие положения

Согласно 7.2.2, контейнерная служба и связанные с ним инструменты позволяют управлять жизненным циклом только одного контейнера. Однако развертывание типичных приложений для облачных вычислений, как правило, включает в себя развертывание и работу множества контейнеров, часто на нескольких хост-системах. Приложение может содержать несколько экземпляров определенного контейнера, работающих параллельно, как для обеспечения резервирования на случай отказа одного экземпляра, так и для обеспечения достаточной масштабируемости, чтобы справиться с нагрузкой, создаваемой входящими запросами. Приложение также может содержать множество различных компонентов, каждый из которых работает в собственном экземпляре контейнера(ов) благодаря использованию архитектуры микросервисов или разделению функциональных возможностей на нескольких уровнях; пример — веб-приложение, использующее базу данных. CMS осуществляет оркестрацию и управление заданными наборами контейнеров.

CMS может абстрагировать базовую инфраструктуру, рассматривая набор контейнеров как единую цель развертывания и в то же время обеспечивая соблюдение политик развертывания, таких как разделение параллельных экземпляров контейнеров для целей резервирования и перехода на другой ресурс при сбое.

7.4.2 Общие функциональные возможности CMS

Общие функциональные возможности CMS включают:

а) оркестрацию

CMS обеспечивают оркестрацию экземпляров контейнеров, включая первоначальное создание и размещение, планирование, мониторинг, масштабирование, обновление и параллельное развертывание таких функциональных возможностей, как балансировщики нагрузки, межсетевые экраны, виртуальные сети и средства протоколирования.

Инструменты оркестрации CMS могут абстрагировать базовую инфраструктуру, рассматривая набор контейнеров как единую цель развертывания и в то же время обеспечивая соблюдение политик развертывания, таких как разделение параллельных экземпляров контейнеров для целей резервирования и перехода на другой ресурс при сбое.

Оркестрация — это ключевой компонент CMS, необходимый для поддержания масштабирования, поскольку масштабирование требует наличия эффективных инструментов автоматизации;

б) планирование

Планировщик обеспечивает постоянное наличие достаточного объема ресурсов, необходимых инфраструктуре. Планировщик выбирает узел на основе своей оценки доступности ресурсов, а затем отслеживает уровень использования ресурсов для гарантии того, что компонент не превышает объем выделенных ему ресурсов. Он поддерживает и отслеживает требования к ресурсам, доступность ресурсов и множество других ограничений, определенных пользователем, а также требования политик;

в) мониторинг и проверку состояния

Автоматизация — это важный элемент систем облачных вычислений, особенно когда речь идет об обеспечении отказоустойчивости и быстрой масштабируемости. Автоматизация может быть обеспечена для производственных систем только с помощью CMS, которая постоянно отслеживает распределенный набор контейнеров приложения и оценивает их состояние.

Автоматизация позволяет обнаруживать сбои и отказы и предпринимать действия для поддержания необходимой конфигурации приложения, заданной в декларативной конфигурации. Для этих целей может выполняться удаление отказавших экземпляров и запуск новых;

г) автоматическое масштабирование

С помощью мониторинга может выполняться динамическое масштабирование ресурсов, доступных приложению, для соответствия рабочей нагрузке. Это делается для того, чтобы объем ресурсов, необходимых для обработки нагрузки, был минимальным, поскольку облачные вычисления часто оплачиваются на основе использованных ресурсов;

д) управление ресурсами

Ресурс в CMS — это логическая конструкция, которую может создавать инструмент оркестрации и управлять ею. В качестве примера можно привести развертывание сервиса или приложения;

е) виртуальную сеть

Типичное приложение состоит из множества отдельных компонентов, которые работают вместе для поддержания функциональности приложения. Отдельные компоненты обычно должны обмениваться данными друг с другом с помощью сети, поскольку они часто размещаются в разных местах. CMS отвечает за настройку сетевых соединений, необходимых для обмена данными компонентами. Сеть часто является виртуальной, что устраняет необходимость понимания компонентами базовой сетевой инфраструктуры, а также повышает безопасность, ограничивая обмен данными только теми компонентами, которые относятся к приложению;

ж) обнаружение сервисов

Обнаружение является ключевым элементом, связанным с развертыванием контейнеров, поскольку приложения состоят из множества контейнеров и связанных с ними компонентов, работающих в широко распределенной инфраструктуре. В результате отдельные компоненты должны обнаружить другие связанные компоненты.

Например, балансировщику нагрузки необходимо определить все экземпляры компонентов, которые он использует, для распределения входящих запросов. CMS предоставляет функциональные возможности, которые помогают при решении подобных задач;

и) обновление и модернизацию

CMS управляют процессом обновления и модернизации компонентов приложения. Эти изменения могут быть результатом появления новых функций или исправлений самого программного кода приложения или могут быть связаны с обновлением программного стека, используемого программным кодом приложения, включая среды выполнения. CMS управляют обновлением для предотвращения простоев в работе путем поэтапного внедрения экземпляров с обновленным кодом и удаления экземпляров со старым кодом;

к) декларативную конфигурацию

Обычно CMS предоставляют команде специалистов DevOps средства для декларативной настройки конфигурации оркестрации приложения с использованием заданной схемы, написанной на таких языках программирования как YAML (gost.e-ecolog.ru 1 JSON (gost.e-ecolog.ru publications-and-standards/standards/ecma-404/). Декларативная конфигурация обычно также содержит информацию о репозиториях контейнеров, сетевой конфигурации, средствах хранения и функциях обеспечения безопасности, которые поддерживает приложение. Декларативная конфигурация необходима для того, чтобы CMS могли автоматизировать процесс управления приложением и его компонентами. По сути, декларативная конфигурация указывает CMS нужную конфигурацию. CMS, в свою очередь, стремится создавать и поддерживать эту конфигурацию. Для этого она принимает решения в отношении действий, необходимых для достижения поставленной задачи, используя знания о целевых системах и внутренних стратегиях развертывания.

8 Бессерверные вычисления

8.1 Общие положения

Бессерверные вычисления представляют собой категорию облачных сервисов, в рамках которой CSC может использовать различные типы облачных функциональных возможностей. При этом CSC не нужно предоставлять, развертывать и управлять аппаратными или программными ресурсами, за исключением предоставления кода приложения или данных CSC. Бессерверные вычисления обеспечивают автоматическое масштабирование с динамическим адаптивным выделением ресурсов поставщиком облачной службы, автоматическое распределение ресурсов по нескольким объектам, а также автоматическое обслуживание и резервное копирование. Функциональные возможности бессерверных вычислений запускаются одним или несколькими событиями, определенными потребителем облачной службы, и работают в течение ограниченного периода времени, необходимого для каждого конкретного события. Функциональные возможности бессерверных вычислений также могут быть запущены с помощью прямого вызова из мобильных и веб-приложений.

Базовая концепция бессерверных вычислений основывается на предоставлении функций как услуги. Идея заключается в том, чтобы поставщик облачной службы динамически и по требованию выделял ресурсы среды выполнения, необходимые для кода приложений потребителя облачной службы, и при этом потребителю не нужно было бы предварительно выделять компьютеры или управлять конкретными компьютерами, ВМ или контейнерами и любым связанным с ними программным стеком. Существуют и другие виды облачных сервисов, которые поддерживают модель бессерверных вычислений, в частности бессерверные базы данных.

Бессерверные вычисления можно также представить как одну из категорий платформенных облачных сервисов (или PaaS), поскольку потребитель облачной службы предоставляет только сам код приложения и (или) данные, а все остальные ресурсы и функциональные возможности, необходимые для работы приложения, предоставляются и управляются поставщиком облачной службы.

Как правило, облачные сервисы категории бессерверных вычислений масштабируются автоматически для обработки входящих запросов. Облачные сервисы категории бессерверных вычислений часто располагают функциями обеспечения устойчивости к сбоям, например возможностью размещения кода приложения потребителя облачной службы в нескольких местах с автоматическим переключением при возникновении сбоя.

Для работы бессерверных вычислений все равно требуются серверы, поэтому в этом смысле их название не соответствует действительности. Что действительно не требуется, так это выделение и управление ресурсами сервера со стороны потребителя облачной службы.

В бессерверных вычислениях часто используют модели оплаты за объем работы, выполненный облачным сервисом, а не за выделенные ресурсы (например, ВМ или контейнер). Поэтому оплата может производиться из расчета за один вызов API или, например, за один HTTP-запрос. Такую модель оплаты можно рассматривать как самый экстремальный вариант оплаты по факту потребления.

Среди преимуществ бессерверных вычислений для потребителя облачной службы могут быть следующие:

- снижение эксплуатационных расходов, в частности снижение расходов, связанных с масштабированием ресурсов для соответствия меняющейся рабочей нагрузке, поскольку оплата напрямую связана с объемом работы, выполняемой облачным сервисом, а не с выделенными ресурсами. Кроме того, необходимые программные стеки не нуждаются в обработке, как это обычно бывает при использовании ВМ и контейнеров;
- снижение затрат на разработку и сокращение времени разработки, поскольку разработчикам не нужно учитывать различные вопросы управления ресурсами, включая развертывание и масштабирование, так как код приложения, например, можно просто загрузить и выполнить;
- меньшая сложность упаковки и развертывания. В частности, не нужно учитывать различные вопросы, за исключением кода приложения или данных потребителя облачной службы. Любые связанные программные стеки поставляются поставщиком облачной службы как часть облачного сервиса, а не упаковываются и не развертываются потребителем облачной службы.

8.2 Функции как услуга

8.2.1 Обзор

Распространенной формой бессерверных вычислений является форма «функции как услуга» (FaaS). FaaS представляет собой форму бессерверных вычислений, в которой функциональные воз-

можности, используемые потребителем облачной службы, заключаются в выполнении кода приложения потребителя в виде одной или нескольких функций, каждая из которых запускается по выбранному потребителем событию. FaaS также является разновидностью формы «вычисления как услуга» (CompaaS) согласно определению по ГОСТ ISO/IEC 17788.

FaaS может выполнять код клиентского приложения, написанный на одном или нескольких языках программирования. Все облачные сервисы FaaS поддерживают приложения, написанные на одном или нескольких языках программирования, включая, помимо прочего, C#, Go, Java™, JavaScript, PHP, Python, Ruby, Swift. FaaS поддерживает функциональные возможности платформы как услуги в отношении предоставления программного стека среды выполнения, необходимого для выполнения кода приложения. Таким образом, потребителю облачной службы не нужно развертывать и поддерживать программный стек среды выполнения. Как правило, FaaS не требует написания кода приложения потребителя облачной службы для использования какой-либо конкретной среды разработки приложений. Кроме того, FaaS организует работу приложения и программного стека среды выполнения по требованию для обслуживания любых событий, которые запускают приложение.

Цель FaaS — сделать инфраструктуру невидимой для разработчиков приложений и сервисов. При использовании FaaS базовые серверы, ВМ и (или) контейнеры невидимы для пользователя сервиса. Разработчик не только не имеет к ним доступа, но и не может получить его, поскольку они автоматически управляются поставщиком облачной службы как часть облачного сервиса.

Важным аспектом FaaS является то, что ресурсы потребляются только во время выполнения определенной функции. Обычно, когда приложение использует ВМ или контейнер в вычислительном облачном сервисе, каждый запущенный экземпляр ВМ или контейнера потребляет ресурсы непрерывно и независимо от того, выполняет ли он входящие запросы или нет. В случае с FaaS ресурсы не потребляются, когда обрабатываемые события отсутствуют. Это может снизить затраты потребителя облачной службы, особенно в том, что касается редко используемых функций. FaaS выделяет необходимые ресурсы (например, базовый контейнер), когда в отношении данной функции возникает событие.

Автоматизированное управление, обеспечиваемое FaaS, является основной отличительной особенностью, т. к. высокая масштабируемость обеспечивается в автоматическом режиме. Если частота возникновения событий для данной FaaS растет, то объем ресурсов, выделяемых соответствующей облачной службе для обработки этих событий, автоматически увеличивается, а когда частота возникновения событий снижается, они удаляются.

8.2.2 Функции FaaS

Использование FaaS подразумевает написание одной или нескольких функций, где каждая функция — это часть программного кода, предназначенная для выполнения одной конкретной задачи. Именно из-за этой особенности программирования бессерверной среды выполнения эти облачные сервисы получили такое название — «функция как услуга» (FaaS). По сути, это серьезное изменение способа разработки приложений и сервисов, включая добавление отдельных принципов архитектуры микросервисов (см. раздел 9). Существуют определенные принципы, которые действуют в отношении функций и способа их выполнения.

Функции не имеют состояния, что означает, что все функции не сохраняют состояния, возникающие между их последовательными вызовами. Фактически это означает, что функции сами не хранят никаких данных. Если функции необходимо хранить данные и получать к ним доступ и время жизни этих данных превышает время одного вызова функции, то тогда функция интегрируется с одним или несколькими облачными сервисами хранения данных (см. раздел 12).

Каждая функция выполняется при наступлении некоторого события, где уведомление о событии является результатом некоторого изменения состояния, т. е. модель FaaS имеет соответствующую архитектуру, управляемую событиями, а это предполагает асинхронное поведение, связанное с отправкой и получением событий. Такой подход обеспечивает гораздо более высокую масштабируемость и устойчивость приложений, особенно если приложения реализованы на распределенных системах.

Функции ограничены по времени, т. е. они не могут выполняться дольше заданного времени, определяемого поставщиком облачной службы. Лимит времени зависит от конкретного предложения FaaS, но часто он не превышает нескольких минут. Таким образом, какие-либо длительные задачи не подходят для обработки в качестве функции.

С ограничением функций по времени связан и вопрос задержки запуска функций. Речь идет о количестве времени, которое необходимо FaaS, чтобы при возникновении события создать работающий экземпляр функции. Это может быть либо холодный запуск, когда новый экземпляр должен быть запу-

щен с нуля, либо горячий запуск, когда FaaS может повторно использовать экземпляр из предыдущего события. Первый вариант имеет гораздо большую задержку, чем второй. Холодный запуск гораздо более вероятен для функций, которые используются нечасто, поскольку FaaS обычно удаляет экземпляр, который не использовался на протяжении определенного (обычно короткого) периода времени.

Все функции доступны через API и могут быть вызваны либо клиентом, находящимся полностью вне облачной системы (например, приложением конечного пользователя, запущенным на клиентском устройстве), либо клиентом, который является частью общего приложения. Каждая функция может рассматриваться как микросервис, и, в свою очередь, каждая функция может зависеть от использования других микросервисов для реализации своих функциональных возможностей.

То, как события описываются в структурах данных, имеет важное значение для функций в рамках бессерверных сред выполнения. Поэтому были созданы спецификации, помогающие четко и согласованно описывать события.

Поскольку за один раз можно развернуть только одну функцию, бессерверный подход обеспечивает значительную гибкость в этом плане. Приложения могут создаваться по одной функции за раз, каждая из которых развертывается и масштабируется отдельно. Это также позволяет ускорить разработку и развертывание (нет необходимости ждать сборки приложения или сервиса, содержащего несколько функциональных возможностей). Каждая функциональная возможность может быть создана, протестирована и развернута отдельно.

8.2.3 Бессерверные фреймворки

Бессерверный фреймворк представляет собой инструмент, призванный оказать помощь при создании и развертывании функций для облачных сервисов FaaS. В частности, он поддерживает развертывание функций в рамках разных предложений FaaS различных поставщиков облачных служб.

Как правило, сервисы FaaS являются собственностью поставщика облачной службы. Однако существуют и отдельные реализации FaaS на базе открытого исходного кода.

Для решения проблемы разработки функций для развертывания на любом из множества предложений FaaS поставщиков облачных служб были разработаны бессерверные фреймворки, которые позволяют разрабатывать функции, ориентированные на различные FaaS по требованию, учитывая различия между ними (особенно в отношении процессов загрузки и развертывания).

8.2.4 Связь FaaS с микросервисами и контейнерами

FaaS подразумевает использование облачной архитектуры микросервисов для приложений. FaaS предусматривает использование «облачного» подхода к приложениям, который существенно отличается от «монолитных» приложений, где все функции содержатся в одном пакете.

Таким образом, использование FaaS и функций — это один из способов реализации архитектуры приложений на основе микросервисов, который требует использования контейнеров и связанных с ними CMS.

Однако существует возможность совместного использования FaaS с микросервисами, реализованными с помощью контейнеров (или с использованием ВМ). При этом функции будут вызывать микросервисы на основе контейнеров, а микросервисы на основе контейнеров будут вызывать функции FaaS по мере необходимости.

8.3 Бессерверные базы данных

Бессерверная база данных представляет собой форму бессерверных вычислений, в которой функциональная возможность, используемая потребителем облачной службы, представляет собой базу данных. При этом база данных предоставляется, управляется и эксплуатируется поставщиком облачной службы, а ее функции предлагаются через API.

Что касается бессерверных вычислений, то распределением ресурсов хранения управляет поставщик облачной службы. Объем хранилища автоматически и динамически масштабируется в соответствии с объемом данных потребителя облачной службы, которые помещаются в базу данных. Управление репликацией и резервным копированием осуществляет поставщик облачной службы. Эти операции включают в себя размещение данных в местах, подходящих для их использования, а также поддержание нескольких реплик в соответствии друг с другом. Вычислительные ресурсы, необходимые для обслуживания запросов и обновлений базы данных, также управляются и масштабируются поставщиком облачной службы.

9 Архитектура микросервисов

9.1 Общие положения

Архитектура микросервисов представляет собой подход к построению облачного приложения. Облачное приложение специально создается для работы в облаке, а также для использования функциональных возможностей и среды облачных сервисов. Архитектура микросервисов является стилем архитектуры, предполагающим разбиение приложения на микросервисы, которые могут быстро и независимо друг от друга быть развернуты на любом инфраструктурном ресурсе по мере необходимости. В архитектуре микросервисов приложение разделяется на ряд отдельных процессов, называемых микросервисами, которые развертываются независимо и связываются друг с другом с помощью сервисных интерфейсов. Идея заключается в том, что микросервисы в рамках приложения предназначены для выполнения определенной области функций, например конкретного бизнес-процесса или определенной технической функциональной возможности. Эта архитектура позволяет использовать и обновлять каждый микросервис отдельно друг от друга. Таким образом, архитектура микросервисов является зонтичным методом, в которой микросервисы выступают в качестве основных компонентов.

Примечание — Термины «сервис» (услуга) и «сервисный интерфейс» используются в соответствии с определениями, приведенными в ГОСТ Р ИСО/МЭК 18384-1. Стандарт также содержит понятное объяснение сервис-ориентированной архитектуры, конкретным примером которой является архитектура микросервисов. Сервис или микросервис не следует путать с облачным сервисом. Может возникнуть ситуация, когда микросервис реализуется как облачный сервис, но это зависит исключительно от выбора разработчика приложения, и никаких определенных требований здесь нет.

Простой пример приложения, основанного на микросервисах, приведен на рисунке 4. В данном примере приложение состоит из ядра и двух микросервисов, один из которых работает с учетными записями пользователей, а второй занимается отображением и управлением видеорядом. В примере также показывается, что приложения микросервисов могут также использовать другие сервисы, как правило облачные, предоставляющие функциональные возможности, необходимые приложению. Поэтому в этом примере микросервис учетной записи пользователя использует сервис базы данных для хранения и получения информации об учетной записи; микросервис отображения видео использует сервис хранения видео, который хранит видеоряд; основное приложение использует сервис электронной почты, сервис Twitter-ленты и сервис аналитики.

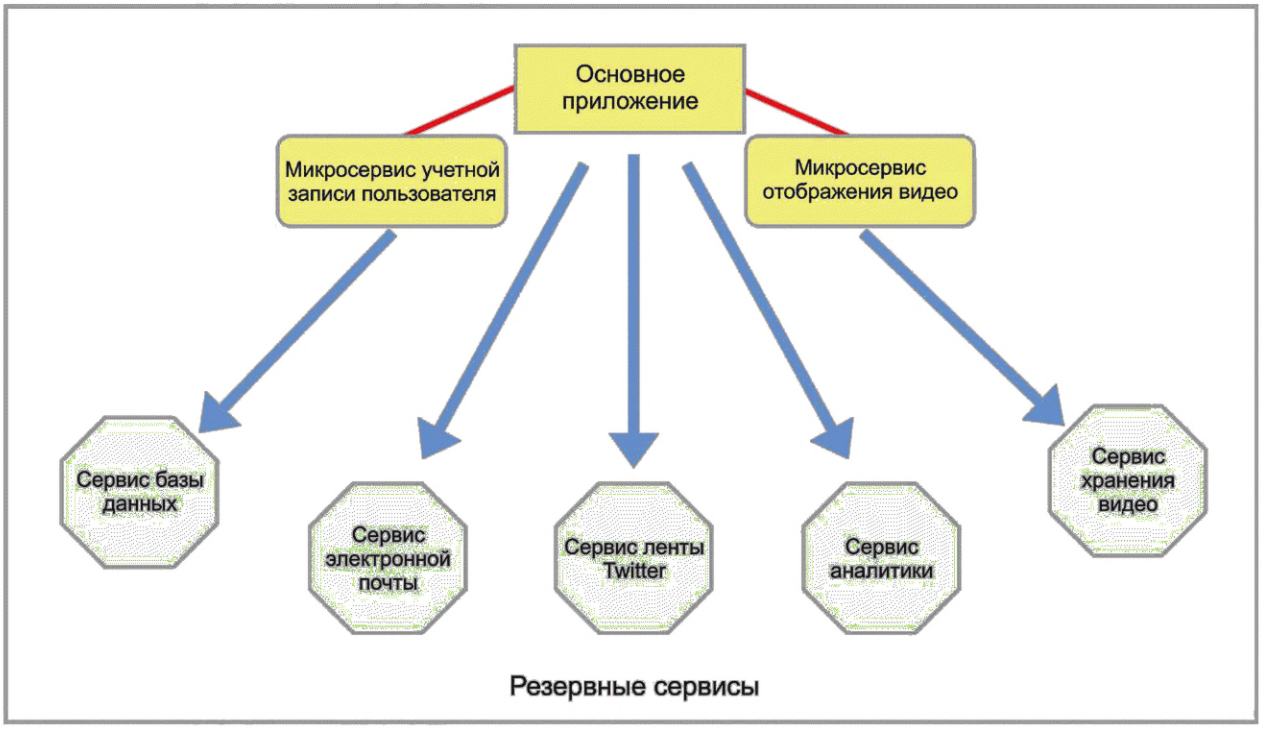

Рисунок 4 — Пример приложения, структурированного с использованием архитектуры микросервисов

Важно понимать, что архитектура микросервисов — это метод, а также, что микросервисы являются основной частью архитектуры микросервисов. В этом и заключается отличие от технологий, которые могут быть использованы для реализации микросервисов. Микросервисы могут быть реализованы с помощью контейнеров, ВМ или бессерверных вычислений и подключены с помощью виртуализированных сетей, но этот подход отличается от метода, используемого для создания приложений.

Часто в контексте предметно-ориентированного проектирования использование функциональной декомпозиции является залогом построения успешной архитектуры микросервисов. Есть мнение, что эта архитектура представляет собой усовершенствованную и упрощенную версию сервис-ориентированной архитектуры (SOA). Архитектура микросервисов имеет следующие особенности:

- каждый архитектурный компонент («сервис») имеет четко определенный и явно обозначенный интерфейс;
- каждый сервис работает полностью автономно;
- изменение реализации сервиса не влияет на другие сервисы, поскольку обмен данными между сервисами происходит только через интерфейсы (обычно REST-интерфейс);
- слабая связанность и высокая связаность между сервисами позволяет создавать несколько сервисов для определения сервисов или приложений более высокого уровня.

Приложения на основе микросервисов отличаются от монолитных приложений, в которых все компоненты приложения создаются и собираются в рамках одного процесса, что характерно для более старых, необлачных корпоративных приложений.

9.2 Преимущества и недостатки микросервисов

Преимуществами микросервисной архитектуры являются:

- более простые кодовые базы для отдельных сервисов;
- возможность обновления и масштабирования каждого сервиса в отдельности;
- возможность написания сервисов на разных языках для удовлетворения потребностей в высокой скорости работы и удобства разработки (это называется «многоязычное программирование»);
- использование различных стеков промежуточного ПО и даже различных уровней данных для различных сервисов (высокая гибкость работы).

Одним из преимуществ использования архитектуры микросервисов является то, что каждый компонент приложения, построенный как микросервис, может быть масштабирован отдельно, чтобы соответствовать нагрузке именно на этот компонент. В подходе, используемом для создания монолитных приложений, такая возможность отсутствует. В архитектуре монолитного приложения все компоненты развертываются и работают как единое целое, а масштабирование возможно только путем увеличения или уменьшения масштаба всего приложения. Это может привести к снижению эффективности использования ресурсов для тех компонентов приложения, для которых высокая нагрузка отсутствует.

Системы PaaS позволяют легко развернуть каждый микросервис по отдельности и связать их вместе для создания полноценного приложения. Каждый микросервис может управляться независимо: масштабироваться, распределяться, обновляться.

Еще одним преимуществом использования архитектуры микросервисов является то, что каждый компонент приложения, построенный как микросервис, может иметь независимый жизненный цикл разработки. Это позволяет создавать более компактные компоненты приложений, которые можно быстрее модифицировать, расширять, тестировать и развертывать.

Кроме преимуществ есть еще и недостатки, которые для реализации всего потенциала преимуществ необходимо устранить. Краткое описание этих недостатков:

а) оптимизация обмена данными. Работа приложения в разных процессах приводит к увеличению объема передаваемых данных в связи с большим количеством вызовов API между сервисами по сравнению с вызовами функций, передаваемыми внутри процесса. Чтобы исправить ситуацию, необходимо определить оптимальный протокол, ожидаемое время отклика, значения тайм-аутов и конструкцию API. Для этого могут использоваться такие программные компоненты, как API-шлюз (см. 9.7), автоматические размыкатели (см. 9.6), балансировщики нагрузки (см. 14.2) и прокси-серверы (см. 14.2);

б) обнаружение сервисов. Обнаружение сервисов означает возможность сервисов согласованно обнаруживать друг друга. Для того чтобы сервисы могли регистрировать и объявлять себя, необходимо внедрить стандартизированный и согласованный рабочий процесс. Сервисы должны быть способны обнаруживать конечные устройства и местоположение других сервисов. Необходимо использовать спецификацию, которая будет определять порядок настройки шлюзов API для информирования о доступности сервисов и сохранения возможности обнаружения;

в) скорость работы. Выполнение одного функционального бизнес-требования может подразумевать оркестрацию сразу нескольких вызовов сервисов. Это может увеличить задержку отклика. Кроме того, данные, которые часто используются одним микросервисом, могут принадлежать другому микросервису. Для того чтобы избежать возникновения большого объема передаваемых данных, связанных с копированием данных при вызове сервисов, необходимо наличие функций совместного использования и синхронизации данных;

г) отказоустойчивость. Отказоустойчивость — это способность системы восстанавливаться после частичного отказа. Разработчикам микросервисов необходимо предложить механизмы для восстановления или остановки распространения сбоя на другие части системы. Кроме того, некоторые сервисы выполняются в нескольких копиях для обеспечения высокой масштабируемости и доступности. Ключевыми факторами для обеспечения отказоустойчивости являются количество копий, согласованность версий между копиями, механизм балансировки нагрузки и расположение сети;

д) безопасность. Критически важным решением является формирование доверительных отношений между микросервисами на основе различных способов, доступных сервисам для обмена данными друг с другом. Сервис может использовать синхронный или асинхронный протокол при вызове другого сервиса. Все эти факторы необходимо учитывать при назначении цепочек авторизации в токенах доступа. Схемы обмена данными между сервисами должны иметь специальные и эффективные механизмы аутентификации и авторизации, созданные на базе политик безопасности на базе управления рисками. Увеличение объема данных, передаваемых между компонентами [см. перечисление а)], требует использования защищенных коммуникационных протоколов, отвечающих требованиям приложения;

е) трассировка и протоколирование. При разбиении монолитных приложений на отдельные микросервисы возникает необходимость в дополнительных методах и решениях для отладки и профилирования систем. Один из таких методов называется «распределенная трассировка». Она отслеживает цепочку вызовов сервисов для поиска отдельной бизнес-транзакции или отдельного пользовательского запроса. Для получения целостного представления о работе системы обычно требуется центральная система протоколирования, которая поддерживает функцию агрегации для объединения информации из журналов отдельных микросервисов;

ж) развертывание. Распространение сервисных процессов требует наличия автоматизированных механизмов развертывания. Масштабируемость и целостность системы являются основными вопросами, которые необходимо учитывать при развертывании микросервисов. Контейнеры являются ключевым механизмом, используемым для развертывания микросервисов, а использование CMS (см. 7.4) (которая назначает ресурсы и реализует топологию соединений) решает проблемы развертывания. При этом отдельные предположения и требования моделей развертывания могут плохо сочетаться с функциональными требованиями отдельных приложений на базе микросервисов. В качестве примера можно предположить, что контейнер, в котором размещается микросервис, не имеет состояния. При этом общие требования к системе или приложению требуют наличия микросервиса, который имеет состояние;

и) функциональная декомпозиция. При декомпозиции монолитного приложения необходимо решить следующие вопросы:

1) правильное разграничение различных сервисов;

2) разделение сервиса на отдельные части, если он слишком объемный.

9.3 Спецификация микросервисов

Проектирование архитектуры микросервисов требует использования диаграмм описания и платформенно нейтральных языков описания из-за неоднородности при проектировании микросервисов компонентов. Хотя язык моделирования UML преимущественно используется для диаграмм описания, обычно применяются следующие языки:

- стандартные языки моделирования, такие как RAML и YAML;
- стандартные языки спецификаций, такие как JavaScript (Node.js), JSON и Ruby;
- псевдокод для алгоритмов;
- язык спецификации интерфейса, нейтральный к реализации, например спецификация Open API (gost.e-ecolog.ru).

9.4 Многоуровневая архитектура

В программировании часто используются предметно-ориентированное проектирование и соответствующая многоуровневая архитектура (см. gost.e-ecolog.ru

gost.e-ecolog.ru).

Благодаря функциональному разделению приложений на отдельные уровни многоуровневая архитектура обеспечивает следующие преимущества:

- эффективную совместную работу: каждый уровень разрабатывается специалистом, специализирующимся именно на данном уровне, например: графический интерфейс пользователя на основе веб-браузера разрабатывается веб-дизайнерами, а логика предметной области — программистами Java™. Специалисты могут сосредоточиться на своих собственных разработках при минимуме посторонних проблем;
- простое обслуживание: программный код каждого уровня логически независим от программного кода других уровней. Если программисты не нарушают интерфейсы на других уровнях, можно легко менять программный код;
- возможность повторного использования: в многоуровневой архитектуре одно приложение разбивается на небольшие компоненты. Мелкие программные компоненты гораздо лучше подходят для повторного использования, чем крупные.

Использование многоуровневой архитектуры оказывается эффективным в приложениях на базе микросервисов. Многоуровневая архитектура используется в приложениях, разработанных с использованием микросервисов, и отдельные методы, связанные с использованием многоуровневой архитектуры, достаточно подробно описываются в литературе. Хотя для микросервисов не существует стандартизированной многоуровневой архитектуры, может быть использована многоуровневая архитектура, предложенная в предметно-ориентированном проектировании, суть которой можно выразить с помощью четырех уровней компонентов (см. рисунок 5):

- пользовательский интерфейс. Программный компонент принимает запросы от пользователей и предоставляет ответы;
- приложение. Программный компонент определяет границу приложения. Он представляет собой конечную точку взаимодействия с клиентами и осуществляет обработку запросов и ответов, вызов логики предметной области и управление контекстами транзакций;
- предметная область. Программный компонент реализует бизнес-логику;
- инфраструктура. Программный компонент инкапсулирует физические ресурсы, включая данные, и предоставляет уровню предметной области абстрактный интерфейс для доступа к данным.

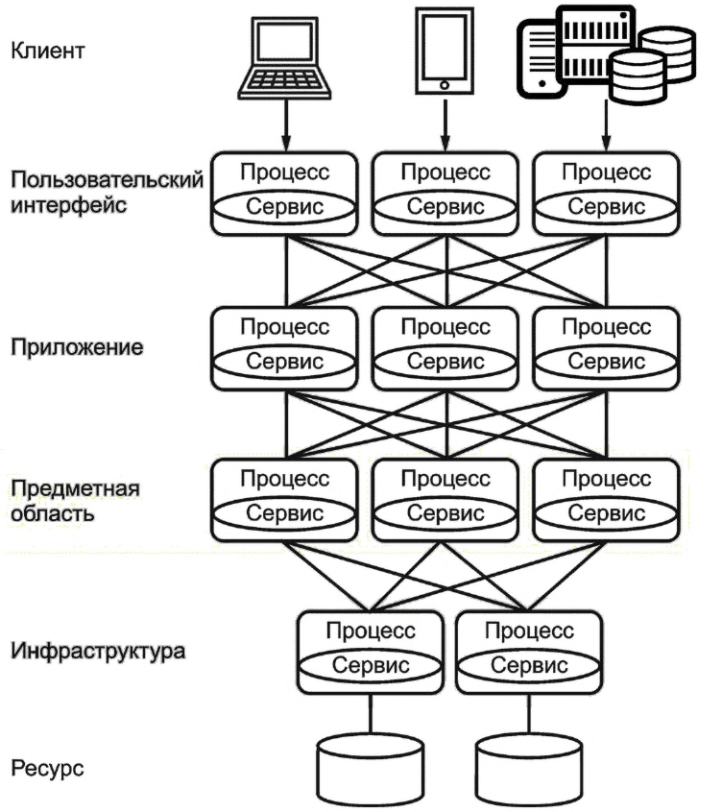

Рисунок 5 — Многоуровневая архитектура, используемая с микросервисами

Каждый компонент, показанный на рисунке 5, представляет собой отдельный микросервис, работающий в своем собственном процессе и вызывающий другие микросервисы по мере необходимости.

Монолитные веб-приложения ранее разрабатывались на основе многоуровневой архитектуры, известной в виде шаблона «модель — представление — контроллер» (см. рисунок 6). Однако существуют некоторые различия в реализации многоуровневой архитектуры между приложением, разработанным с использованием микросервисов, и приложением с монолитной конструкцией. Они связаны с упаковкой программных компонентов и средами выполнения приложения.

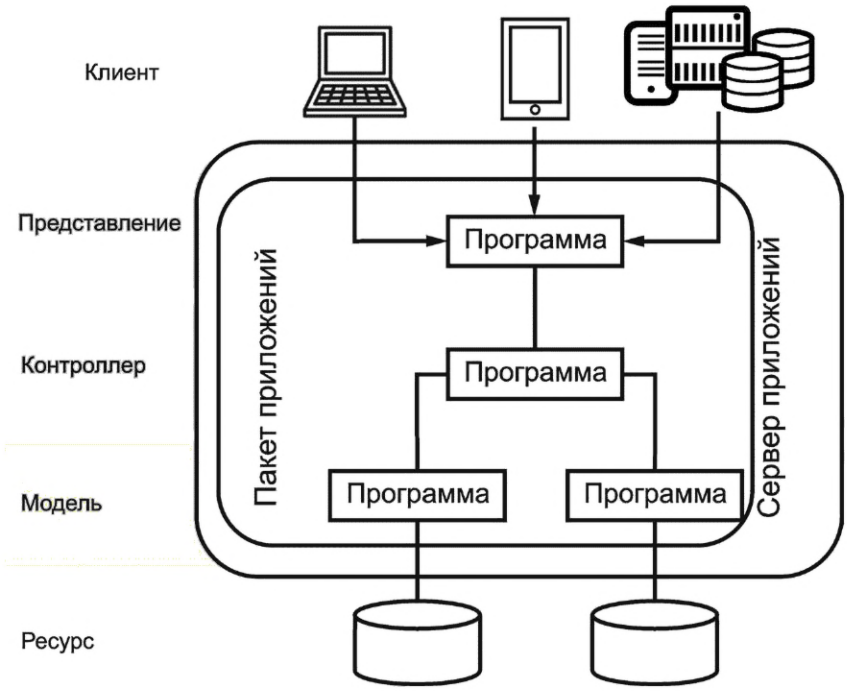

Рисунок 6 — Модель монолитного веб-приложения

При проектировании монолитных приложений, хотя приложение разрабатывается и реализуется на основе многоуровневой архитектуры, все программные компоненты собираются в один программный пакет и развертываются в одной среде выполнения приложения. При условии, если веб-дизайнер добавляет небольшое обновление в графический интерфейс, необходимо собрать и протестировать весь программный пакет, причем серверная среда выполнения должна быть остановлена для развертывания нового программного пакета. Это может занять достаточно много времени, даже в случае небольшого изменения приложения.

С другой стороны, при проектировании приложения микросервисов каждый программный компонент на каждом уровне упаковывается как отдельный микросервис и независимо развертывается в отдельном процессе. Каждый процесс может быть реализован с помощью ВМ, контейнера или в виде бессерверных функций, каждая из которых может быть запущена и остановлена отдельно. Если каждый микросервис спроектирован правильно и имеет слабосвязанную архитектуру, разработчик может обновить один компонент без необходимости собирать, тестировать или повторно развертывать другие микросервисы. Архитектура микросервисов позволяет легко вносить изменения в приложение.

9.5 Сервисная сетка

В архитектуре микросервисов количество связанных с приложением микросервисов может быть достаточно большим. Обычно в таких случаях каждый сервис работает с несколькими экземплярами с кластерной конфигурацией — каждый экземпляр с отдельным процессом, реализованным в виде ВМ или контейнера. Количество процессов может во много раз превышать количество сервисов. Таким образом, общая топология может представлять собой сложную сеть, называемую сервисной сеткой (см. рисунок 7).

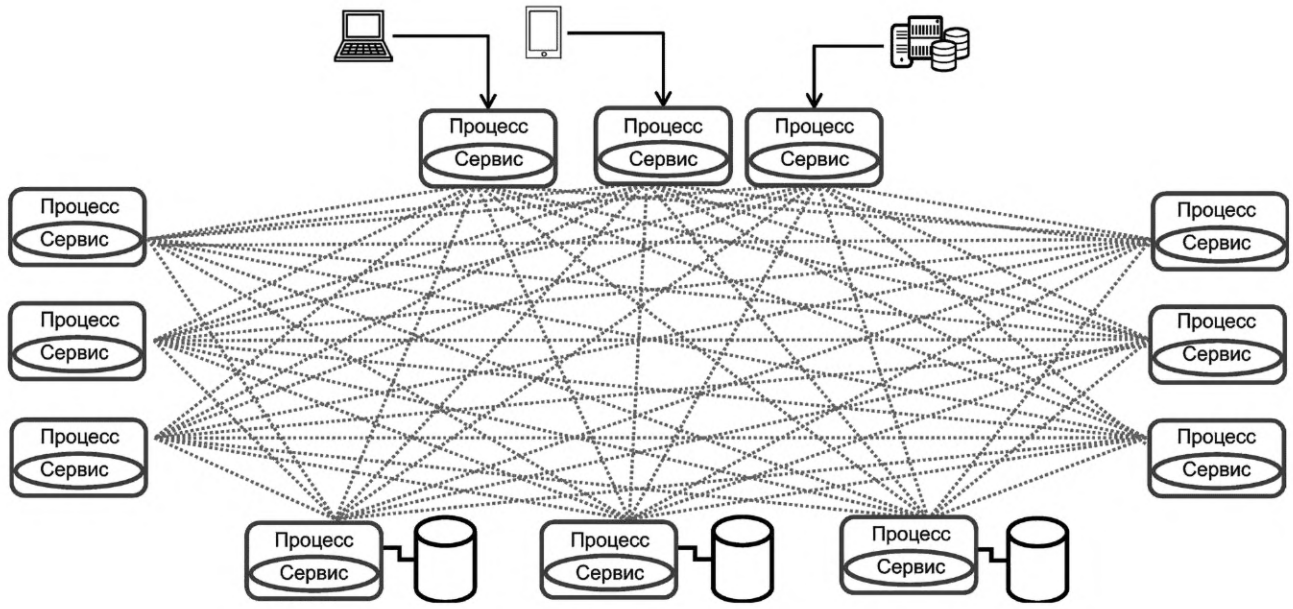

Рисунок 7 — Сервисная сетка для приложения, основанного на микросервисах

Для того чтобы приложение на основе микросервисов работало, и можно было пользоваться всеми его преимуществами, необходимо решить вопросы, связанные с сервисной сеткой:

а) управление трафиком:

1) точная балансировка нагрузки для конкретной версии микросервиса;

2) использование «синего»/«зеленого» методов развертывания для обновления микросервиса без остановки работы приложения;

3) ограниченный релиз;

4) автоматический размыкатель (см. 9.7);

б) обнаружение сервисов:

1) регистрация сервисов;

2) поиск сервисов;

в) тестирование:

1) внесение неисправностей;

г) безопасность:

1) аутентификация;

2) авторизация;

3) шифрование;

д) телеметрия:

1) интеграция функций протоколирования и трассировки;

2) интеграция метрик;

3) информационная панель.

В отношении управления сервисной сеткой существуют и другие подходы:

а) API;

б) среда сервисной сетки.

При подходе с использованием API разработчики приложений используют в своих программах определенный API для управления сервисной сеткой. Однако для этого разработчикам приходится тратить силы на реализацию нефункциональных требований в той же степени, как и функциональных, и в результате в код приложения включаются нефункциональные элементы реализации, что противоречит принципу разделения обязанностей в программировании и делает код более сложным и трудно модифицируемым. Примером API сервисной сетки является Eclipse Foundation. API MicroProfile.

Среда сервисной сетки — это решение для инфраструктуры приложений, расположенное под уровнем приложений и над уровнем оркестрации, которое обрабатывает весь трафик между микросервисами. Эта среда управляет сервисной сеткой, контролируя проходящий трафик. Благодаря этому прикладная программа освобождается от реализации функциональных возможностей, необходимых

для управления сервисной сеткой. Примеры реализаций среды сервисной сетки доступны по адресам:
gost.e-ecolog.ru gost.e-ecolog.ru

9.6 Автоматический размыкатель

Автоматический размыкатель — это шаблон проектирования, а также программный компонент, основанный на этом шаблоне.

Автоматический размыкатель применяется, когда один программный компонент вызывает другой программный компонент (например, микросервис) через API. Эти программные компоненты работают в разных процессах, а вызов API обычно происходит по сети. Такие удаленные вызовы API могут не сработать или остаться без ответа. Если целевым программным компонентом является часто используемый сервис, это может привести к целой серии отказов во всем приложении или системе.

Принцип работы автоматического размыкателя заключается в том, что любой такой удаленный вызов API скрывается компонентом автоматического размыкателя, который фактически является частью клиентского программного компонента. Когда клиент выполняет вызов API, он обрабатывается компонентом автоматического размыкателя, который следит за возможными сбоями. При обнаружении сбоя автоматический размыкатель отправляет сообщение об ошибке на API. В таких ситуациях автоматический размыкатель может также создавать предупреждения в целях мониторинга. Автоматический размыкатель может продолжать контролировать API и целевой компонент на предмет доступности и автоматически выключаться после устранения проблемы.

Определение ошибки может выполняться по-разному для разных автоматических размыкателей. Кроме того, они могут иметь настраиваемые параметры, которые регулируют их работу (например, порог ошибки, порог тайм-аута).

Автоматический размыкатель не избавляет клиентский компонент от необходимости решения проблемы со сбоем вызова API, но облегчает разработку соответствующих механизмов для устранения неполадок.

9.7 API-шлюз

API-шлюз представляет собой программный компонент, который может использоваться для создания единого интегрированного API для набора микросервисов, которые совместно используются определенным клиентским компонентом.

Каждый микросервис имеет свой собственный API, основанный на его функциональных возможностях. Для решения своих рабочих задач конкретный клиент может использовать целую серию микросервисов. У клиентского ПО могут возникать сложности с обработкой различных вызовов API, которые необходимо отправлять на различные микросервисы. Для решения этой проблемы был разработан API-шлюз. Он предлагает клиентскому ПО более простой и согласованный API и вызывает API микросервисов по мере необходимости. Таким образом, API-шлюз представляет собой компонент, ориентированный на клиента. Для удовлетворения требований различных клиентов может потребоваться несколько различных API-шлюзов.

10 Автоматизация

10.1 Общие положения

Автоматизация является ключевой особенностью как в отношении предоставления, так и использования облачных сервисов. Автоматизация применяется на протяжении всего жизненного цикла: при проектировании, разработке, тестировании, развертывании, работе и выводе из эксплуатации. Автоматизация необходима для обеспечения высокой продуктивности работы специалистов, а также для снижения требований к уровню профессиональной квалификации специалистов и уменьшения объема работ, необходимых для предоставления и использования облачных сервисов.

Одной из целей автоматизации является снижение объема работ при развертывании приложений и данных в облачных сервисах. Это связано с тем, что такого рода операции выполняются довольно часто и предназначены либо для устранения неполадок, либо для расширения функциональности. Автоматизация подразумевает внедрение целого ряда методов программирования, которые, хотя и не являются узкоспециализированными решениями, предназначенными исключительно для облачных вычислений, стали важными элементами успешного развертывания облачных вычислений.

10.2 Автоматизация жизненного цикла разработки

Одним из важных элементов автоматизации является внедрение либо непрерывного развертывания, либо непрерывной доставки. Непрерывное развертывание — это подход в программировании, в рамках которого специалисты создают ПО в короткие циклы таким образом, что оно может быть выпущено в производство в любое время, а сам процесс развертывания в производство автоматизирован. Непрерывная доставка выполняется так же, как и непрерывное развертывание. Однако при непрерывной доставке этап развертывания инициируется вручную (т. е. решение о развертывании принимает человек, а не автоматизированная система, а сам процесс развертывания обычно автоматизирован). Как правило, использование методов непрерывного развертывания или непрерывной доставки также связано с использованием в организации концепции DevOps. DevOps включает в себя методику, которая объединяет разработку программного обеспечения и ИТ-операции для сокращения жизненного цикла разработки. Такой подход позволяет специалистам часто добавлять исправления в программные продукты и расширять их функциональность в соответствии с бизнес-целями.

Непрерывное развертывание и непрерывная доставка подразумевают инкрементальную (поэтапную) разработку. При этом во время и после этапов сборки и развертывания тестирование выполняется в автоматическом режиме. Поэтапный принцип работы объясняется использованием микросервисов при разработке приложений (каждый микросервис реализует небольшую отдельную часть общей функциональности всего решения) и отдельных (облачных) сервисов в рамках более общей функциональности (например, функциональные возможности базы данных или обмена сообщениями).

Непрерывная интеграция является неотъемлемой частью процесса непрерывного развертывания и непрерывной доставки, когда обновления базы исходного кода производятся часто, а сама база исходного кода собирается и тестируется регулярно (зачастую по много раз в день). Непрерывная интеграция основывается на разработке посредством тестирования с целью автоматического выполнения модульных тестов и интеграционных тестов с целью убедиться, что обновления базы исходного кода не нарушили программный код. Кроме того, такая схема организации работы призвана обеспечить быструю обратную связь с разработчиками в случае возникновения неполадок.

Автоматизированное управление является ключевой особенностью работы облачных сервисов. Оно используется для решения таких задач, как восстановление отказавших экземпляров ПО, увеличение и уменьшение объема доступных ресурсов, особенно параллельных экземпляров компонентов приложений, репликация данных и резервное копирование данных. При использовании облачных сервисов все эти задачи необходимо автоматизировать, чтобы не создавать избыточную нагрузку на операционный персонал.

Одним из важных элементов концепции DevOps является DevSecOps. В рамках DevSecOps особое внимание уделяется функциям обеспечения безопасности, которые рассматриваются как важная и неотъемлемая часть процессов разработки и эксплуатации. Идея заключается в том, чтобы автоматизировать задачи безопасности параллельно с автоматизацией задач разработки и эксплуатации, которые являются центральным элементом концепции DevOps. При увеличении скорости выполнения задач по разработке и эксплуатации (DevOps) повышается и скорость выполнения задач, связанных с безопасностью, на протяжении всего жизненного цикла приложения (DevSecOps).

10.3 Инструменты автоматизации

Инструменты используются на всех этапах процесса разработки.

Как правило, использование инструментов начинается с системы управления версиями файлов исходного кода (SCM), в которой размещен исходный код и которая предоставляет контролируемые процессы для выполнения обновлений программного кода, включая отслеживание всех изменений. Система SCM создает основу, на базе которой работают инструменты для сборки, тестирования, доставки и развертывания. Существуют различные системы SCM, однако наиболее широкое распространение получила система с открытым исходным кодом: gost.e-ecolog.ru , которая предлагает большое количество вспомогательных инструментов, включая функции хост-сервера.

Сервер автоматизации представляет собой инструмент, используемый для автоматизации этапов непрерывной интеграции, непрерывной доставки и непрерывного развертывания. Этот инструмент оказывается особенно полезным для сборки кода из системы SCM и проведения тестирования (модульные тесты, интеграционные тесты) на основе собранного кода. Существует множество инструментов на основе сервера автоматизации, однако наиболее популярным является инструмент с открытым исходным кодом Jenkins, доступный по адресу: gost.e-ecolog.ru .

Автоматизация задач обеспечения безопасности в рамках DevSecOps может включать инструменты, которые проверяют программный код на наличие уязвимостей на этапе регистрации кода в системе SCM, а также проверяют уязвимости с помощью тестирования во время сборки и на этапе непрерывной интеграции. Подобного рода операции выполняются также и для безопасного использования баз исходного кода, соответствующих требованиям зависимостей приложения, например библиотек промежуточного ПО, образов контейнеров и резервных сервисов. Такие зависимости должны быть связаны с политиками безопасности, определяющими, какие зависимости подходят для использования, и поддерживаться соответствующим тестированием и системой управления, которая реагирует на уведомления об уязвимостях и необходимости перехода на более позднюю исправленную версию программного продукта.

ПО для управления конфигурацией используется для автоматизации предоставления ПО, управления конфигурацией и развертывания приложений. Архитектура, используемая для облачных приложений, увеличивает потребность в ПО для управления конфигурацией. Это объясняется тем, что различные облачные сервисы имеют множество компонентов, и все они должны быть оркестрованы для обеспечения правильной работы приложения. Существует целый ряд программных средств управления конфигурацией, например: Ansible: gost.e-ecolog.ru , Chef: gost.e-ecolog.ru

gost.e-ecolog.ru

Программные средства управления конфигурацией различаются по своей архитектуре. В основе Ansible — архитектура без агентов, в то время как другие инструменты имеют архитектуру с агентами (т. е. они требуют наличия программного системного процесса, работающего в целевых узлах или на связанном с ними сервере).

Ключевым элементом развертывания приложений в облачной среде является оркестрация. Это связано с тем, что обычно приложения состоят из большого количества отдельных компонентов, которые необходимо одновременно развернуть, настроить и обеспечить их работу. Для автоматизации задач оркестрации используются такие инструменты, как CMS (см. 7.4).

Ключевым элементом задач автоматизации является предоставление функциональных возможностей облачных сервисов через API (интерфейсы программирования приложений). API позволяют различным инструментам настраивать, развертывать, контролировать и отслеживать каждый облачный сервис. Это распространяется и на использование других инструментов через API, в частности Kubernetes для развертывания контейнеров.

Приложения, развернутые и работающие в облачных сервисах, должны контролироваться и управляться для обеспечения высокой скорости работы и непрерывной доступности. Контроль и управление обычно осуществляются через API, предлагаемые поставщиком облачной службы. Инструменты для управления перезапуском отказавших экземпляров, как и инструменты для увеличения и уменьшения количества экземпляров определенного программного компонента в соответствии с изменениями рабочей нагрузки, зависят от функций контроля и управления. Некоторые из этих функциональных возможностей предоставляются в виде облачных сервисов (например, функция автоматического масштабирования), но в других случаях они поставляются в виде отдельных инструментов, которые необходимо установить и настроить.

11 Архитектура систем PaaS

11.1 Общие положения

PaaS представляет собой категорию облачных сервисов для предоставления платформенных функциональных возможностей, которые ГОСТ ISO/IEC 17788 рассматривает в качестве инструментов, с помощью которых потребитель облачной службы может развертывать, контролировать и использовать приложения, созданные или приобретенные потребителем, используя один или несколько языков программирования и одну или несколько сред выполнения, поддерживаемых поставщиком облачной службы. Система PaaS обычно включает в себя согласованный набор облачных сервисов PaaS, предназначенных для совместной работы.

Системы PaaS, в первую очередь, предназначены для разработки, развертывания и эксплуатации клиентских приложений. Часто в таких системах предусмотрены и другие вспомогательные функциональные возможности, такие как использование приложений, ресурсов для обработки, хранения и сетевой передачи данных. Системы PaaS обычно включают в себя различные функциональные возможности инфраструктуры прикладного ПО (промежуточное ПО), включая платформы приложений,

интеграционные платформы, платформы бизнес-аналитики, сервисы потоковой обработки событий и мобильные backend-сервисы, а также наборы инструментов, поддерживающих процесс разработки. Кроме того, системы PaaS часто поддерживают набор операционных функциональных возможностей, таких как мониторинг, управление, развертывание и другие сопутствующие функции.

Они поддерживают концепцию DevOps и ориентированы на разработчиков приложений, а также на операционный персонал.

Систему PaaS можно описать как облачный сервис по созданию инфраструктуры приложений, которая включает в себя такие компоненты, как серверы приложений, системы управления базами данных, брокеры интеграции, системы управления бизнес-процессами, обработчики правил и системы комплексной обработки событий. Такая инфраструктура приложений помогает разработчику приложений в написании бизнес-приложений, сокращая объем создаваемого кода и одновременно расширяя функциональные возможности приложений. Суть системы PaaS заключается в том, что поставщик облачной службы занимается установкой, настройкой и эксплуатацией среды выполнения приложений (включая все ВМ, ОС, контейнеры, среды выполнения, библиотеки), предоставляя потребителям сервиса облачных вычислений и разработчикам только сам код приложения. Таким образом, основное различие между системами IaaS и PaaS заключается в том, что в случае с IaaS заказчик должен создать образ ВМ или контейнера для выполнения кода приложения, тогда как PaaS предоставляет все необходимое для загрузки и непосредственного выполнения кода приложения.

Системы PaaS также часто расширяют платформенные функциональные возможности промежуточного ПО, предлагая разработчикам приложений разнообразный и постоянно пополняемый набор сервисов и API с определенными функциями, которые отличаются возможностью управления и постоянной доступностью. Таким образом, разработчики избавлены от необходимости заниматься промежуточным ПО, что сразу повышает продуктивность их работы. Некоторые системы PaaS также сочетают в себе функции облачных сервисов IaaS и SaaS, обеспечивая, с одной стороны, функции контроля распределения основных ресурсов, а с другой — полностью готовые к использованию программные функциональные возможности.

Системы PaaS, как правило, предоставляют свои функциональные возможности таким образом, чтобы приложения, разработанные на их основе, могли совместить в себе преимущества всех отличительных особенностей облачных сервисов. При этом разработчику приложения даже не нужно добавлять специальный программный код в само приложение. Это дает возможность создавать облачные приложения без необходимости в специальных профессиональных навыках.

11.2 Отличительные особенности систем PaaS

Системы PaaS имеют следующие основные отличительные особенности:

- поддержка пользовательских приложений: поддержка разработки, развертывания и эксплуатации пользовательских приложений. Системы PaaS обычно поддерживают облачные приложения, которые могут в полной мере использовать преимущества масштабируемых, адаптивных и распределенных функциональных возможностей облачной инфраструктуры. При этом для их использования разработчикам приложений даже не нужно создавать специальный программный код;

- предоставление сред выполнения: системы PaaS обычно предлагают среды выполнения для приложений, где каждая среда выполнения поддерживает либо один, либо небольшой набор языков программирования и фреймворков, например среды выполнения Node.js, Ruby и PHP. Отличительной особенностью многих систем PaaS является поддержка широкого ряда сред выполнения. Это позволяет разработчикам выбирать наиболее подходящую технологию для решения поставленной задачи. Такие среды иногда называют многоязычными.

Среды выполнения могут использовать контейнеры (см. раздел 7) и бессерверные вычисления (см. раздел 8);

- механизмы быстрого развертывания: многие системы PaaS предоставляют разработчикам и операционному персоналу автоматизированный механизм push and run для развертывания и работы приложений. Он обеспечивает динамическое выделение ресурсов, когда код приложения передается облачному сервису PaaS через API. По умолчанию требования к конфигурации являются минимальными. При этом предусмотрена возможность для контроля конфигурации, если это необходимо; например, возможность контроля количества параллельно работающих экземпляров приложения, чтобы справиться с рабочей нагрузкой или обеспечить отказоустойчивость;

- поддержка ряда функциональных возможностей промежуточного ПО: приложения предъявляют разнообразные требования, для удовлетворения которых предоставляется широкий спектр инфраструктуры приложений (промежуточное ПО) с поддержкой ряда функциональных возможностей. Одним из примеров является управление базами данных. Для этих целей предоставляются технологии баз данных как SQL, так и NoSQL. Другие функциональные возможности включают в себя сервисы интеграции, управление бизнес-процессами, сервисы бизнес-аналитики, обработчики правил, сервисы обработки событий и мобильные backend-сервисы;

- предоставление сервисов: системы PaaS зачастую предоставляют некоторые функциональные возможности в виде ряда отдельных сервисов, которые обычно вызываются через API того или иного типа. Сервисы устанавливаются и запускаются поставщиком услуг, поэтому потребители сервисов облачных вычислений этим не занимаются. Например, если используются сервисы базы данных, поставщик услуг обеспечивает доступность и надежность, наличие реплик и резервных копий данных базы данных, защиту данных и т. п. Услуги, предоставляемые поставщиком, позволяют уменьшить объем работы и сложность при создании программных систем. Вместо того чтобы устанавливать потенциально сложный набор ПО и управлять им, клиенты могут получить готовые функциональные возможности от поставщика;

- предварительно сконфигурированные функциональные возможности: многие системы PaaS поддерживают функциональные возможности, которые предварительно настраиваются поставщиком, при этом разработчикам и операционному персоналу заказчика доступен минимум настроек. В результате процесс упрощается, повышается продуктивность работы и уменьшается вероятность возникновения непредвиденных проблем. При этом такие функциональные возможности проще в управлении и в отладке. Некоторые системы могут автоматически настраивать конфигурацию на основе моделей использования и рабочих нагрузок. Это дополнительно сокращает количество специалистов и времени, необходимых для обеспечения максимально эффективной работы приложений;

- функции управления API: бизнес-приложения часто реализуют отдельные функциональные возможности через API. Это может быть связано с характером пользовательского интерфейса приложения. Мобильным приложениям обычно требуется API, чтобы, работая независимо от бизнес-приложения, они могли получать доступ к данным и транзакциям по мере необходимости. В других случаях предприятие может испытывать необходимость предоставления другим сторонам (партнерам, клиентам, поставщикам) возможностей для интеграции своих собственных приложений с приложениями компании. Такая интеграция осуществляется через API. Предоставление API требует определенного уровня контроля, чтобы только авторизованные пользователи могли получить доступ к API и каждый пользователь мог получить доступ только к индивидуально разрешенным функциональным возможностям. Для этого необходимо наличие определенных функций управления API, которые предлагаются многими системами PaaS;

- функции обеспечения безопасности: безопасность — один из самых важных аспектов любого решения. Системы PaaS обычно оснащены предварительно интегрированными средствами обеспечения безопасности, что позволяет снизить нагрузку на разработчиков и операционный персонал. Функциональные возможности включают в себя межсетевой экран, управление конечными точками, защищенную обработку протоколов, доступ и авторизацию, шифрование данных при перемещении и хранении, проверку целостности, а также механизмы обеспечения отказоустойчивости, такие как резервные копии данных и автоматическое резервное копирование. Системы PaaS могут предложить эти функциональные возможности, оказывая минимальное или нулевое влияние на код приложения, упрощая задачи разработчика. Кроме того, поскольку базовая среда выполнения является частью платформы, поставщик облачной службы осуществляет установку исправлений системы безопасности ОС, обнаружение и удаление вредоносных программ и другие важные задачи по поддержанию безопасности. Таким образом, пользователям не нужно заниматься поиском уязвимостей системы безопасности в собственном коде;

- инструменты разработчика: многие системы PaaS стремятся унифицировать и упростить процессы разработки и эксплуатации, т. е. поддерживают концепцию DevOps, объединяя задачи разработки и эксплуатации. Предоставляемые инструменты разработки включают в себя редакторы кода, репозитории кода, инструменты сборки, инструменты развертывания, инструменты и сервисы тестирования, и инструменты обеспечения безопасности. Часто предлагаются сервисы мониторинга и анализа приложений, которые поддерживают такие функциональные возможности, как протоколирование, анализ журналов, анализ использования приложений и информационные панели;

- операционные функциональные возможности: системы PaaS помогают операционному персоналу, предлагая операционные функциональные возможности как для развернутых приложений, так и для самой системы PaaS. Они реализуются через информационные панели, а также с помощью API и позволяют клиентам подключать собственные операционные инструменты. Например, часто предлагаются функциональные возможности увеличения или уменьшения количества запущенных экземпляров приложения (для решения проблемы меняющейся нагрузки на приложение). В некоторых случаях такие задачи решаются с помощью автоматизированных сервисов, которые меняют количество экземпляров на основе набора правил, установленного эксплуатационным персоналом потребителя облачной службы;

- поддержка переноса существующих приложений: у многих потребителей облачных служб имеются приложения, которые могут быть перенесены в среду PaaS. Некоторые PaaS-системы предлагают прикладные среды, цель которых — соответствовать средам, доступным в существующих необлачных стеках промежуточного ПО, а также соответствующие инструменты, которые помогают в процессе переноса;

- поддержка приложений, использующих архитектуру микросервисов: системы PaaS, как правило, предлагают поддержку приложений, построенных с использованием архитектуры микросервисов (см. раздел 9). Сюда входит поддержка сред выполнения, используемых для самих микросервисов, поддержка базовых сервисов, используемых микросервисами, и поддержка сервисной сетки, которая объединяет все компоненты;

- сетевые функциональные возможности: поскольку поставщик облачной службы полностью контролирует используемые стеки сетевых протоколов, системы PaaS могут быть более глубоко интегрированы с сетевыми функциональными возможностями хоста поставщика облачной службы. Это позволяет относительно легко (как для поставщика облачной службы, так и для разработчика потребителя облачной службы) интегрировать PaaS с технологиями виртуализации сети, балансировки нагрузки сети, технологиями обеспечения отказоустойчивости, оптимизации сети, кеширования, передачи сообщений и работы с очередями и другими сетевыми технологиями.

11.3 Архитектура компонентов, работающих под управлением системы PaaS

Объединение нескольких элементов стандартной системы PaaS приводит к созданию схематической архитектуры компонентов стандартного приложения, созданного и развернутого с помощью системы PaaS (см. рисунок 8).

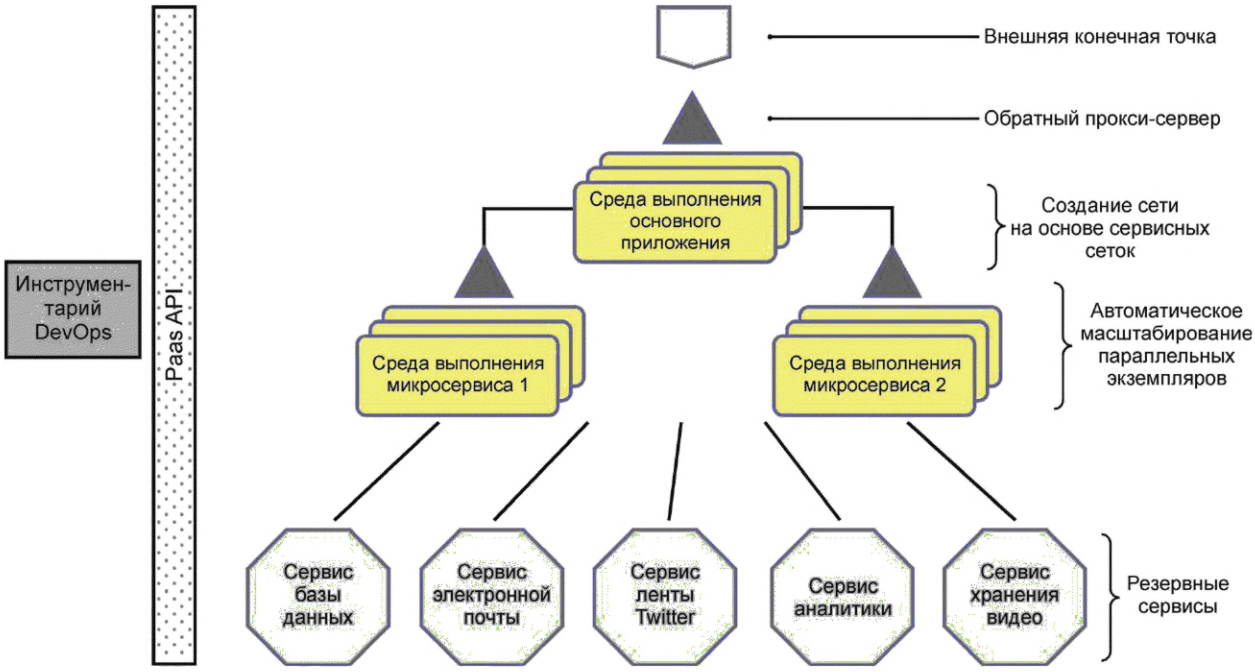

Рисунок 8 — Схематическая архитектура компонентов, работающих под управлением системы PaaS

Внешняя конечная точка: обеспечивает конечную точку с внешним доступом (например, через интернет) и соответствующей системой обеспечения безопасности (например, поддержка протокола https, управление сертификатами, защита от DDoS-атак, управление идентификацией и доступом).

Обратный прокси-сервер: для каждого компонента приложения, который масштабируется с помощью параллельных экземпляров (основное приложение и микросервисы), необходим обратный прокси-сервер и функция балансировки нагрузки для равномерного распределения входящих запросов между всеми запущенными экземплярами.

Сервисная сетка: внутренним соединениям между компонентами приложения и соединениям с сервисами необходимы функциональные возможности для обеспечения эффективного подключения.

Автоматическое масштабирование параллельных экземпляров: обычный подход к масштабированию компонентов приложения заключается в параллельном запуске нескольких экземпляров каждого компонента и распределении входящих запросов по этим экземплярам (см. раздел 14). Количество экземпляров, работающих в каждый момент времени, можно увеличивать и уменьшать в соответствии с объемом запросов. Как правило, для этого системе PaaS необходимо осуществлять мониторинг экземпляров для определения степени их загруженности. Эта функциональная возможность иногда может быть связана с функцией автоматической балансировки загрузки сети PaaS, чтобы уровни трафика для конкретных экземпляров динамически соотносились с их текущей емкостью и доступностью.

Резервные сервисы: как правило, многие функциональные возможности, необходимые компонентам приложения, предоставляются набором облачных сервисов, к которым компоненты подключаются по мере необходимости. Такие сервисы могут быть весьма разнообразными, но наиболее распространенные примеры поддерживают такие функциональные возможности, как сервисы баз данных или другие сервисы хранения данных (см. раздел 12).

PaaS API: функциональные возможности системы PaaS и отдельных облачных сервисов, составляющих эту систему, предоставляются различным инструментам DevOps для использования с помощью одного или нескольких PaaS API. Например, такой API может передавать код компонента приложения в сервис среды выполнения для обработки.

Инструментарий DevOps: разработчики и операционный персонал, в идеальном случае объединенные в группу специалистов в рамках концепции DevOps, используют различные инструменты DevOps в своей работе. Инструменты разработки и тестирования используются во время создания и тестирования приложения и его микросервисов, а инструменты мониторинга и управления используются для наблюдения и контроля компонентов приложения в производстве.

12 Хранение данных как услуга

12.1 Общие положения

Смысл облачных вычислений заключается в предоставлении облачных сервисов, которые в итоге основаны на инфраструктурных ресурсах трех типов: вычислительных ресурсах, ресурсах хранения данных и сетевых ресурсах. Здесь рассматриваются облачные сервисы, которые предлагают ресурсы хранения данных.

Технологии ВМ и контейнеров, описанные в разделах 6 и 7, предлагают вычислительные ресурсы, т. е. средства выполнения ПО в виртуализированной среде определенного типа. Некоторые функциональные возможности хранения данных связаны как с ВМ, так и с контейнерами. Существуют связанные с ними файловые системы, которые содержат файлы, представляющие ПО и непосредственно связанные с ним конфигурацию и метаданные, и эти файловые системы также, как правило, требуются для поддержки выполнения ПО.

Однако необходимо понимать, что файловые системы, связанные как с ВМ, так и с контейнерами, по сути, имеют быстротечный характер, поскольку они появляются в момент создания ВМ или контейнера и исчезают, когда та же ВМ или контейнер останавливаются и удаляются. Это означает, что такие файловые системы не могут быть использованы для долгосрочного хранения информации. Они также не могут быть использованы для информации, которая должна быть доступна нескольким разным ВМ или контейнерам, поскольку файловые системы внутри ВМ или контейнера являются изолированными.

Долгосрочное хранение информации обеспечивается с помощью модели DSaaS. Сервисы DSaaS предлагают возможности хранения данных различных типов и эти функциональные возможности могут быть предложены как системам потребителя облачной службы, так и другим облачным сервисам. Сервисы DSaaS основаны на ресурсах хранения данных поставщика облачной службы, доступ к которым

обычно осуществляется по сети через API. В некоторых случаях сервис DSaaS рекомендуется разместить вместе с вычислительным сервисом, чтобы устранить задержки в работе сети.

12.2 Общие характеристики сервисов DSaaS

DSaaS позволяет пользователям хранить и извлекать информацию в любом месте и в любое время при наличии подключения к сервису DSaaS. Сервисы DSaaS поддерживают масштабируемость с точки зрения объема хранимой информации и обладают надежностью с точки зрения доступа к информации из любого типа приложений независимо от систем или устройств, на которых эти приложения работают.

Приложения и системы используют DSaaS для доступа к облачному хранилищу через соответствующие протоколы. Эти протоколы могут поддерживать географически удаленные ресурсы хранения и поддерживать виртуализацию используемых мест хранения, так что при необходимости резервное или реплицированное хранилище становится доступным для обеспечения устойчивости к точечным сбоям.

Сервисы хранения данных имеют следующие общие отличительные особенности:

а) долговечность. Данные хранятся в одном или нескольких местах, контролируемых поставщиком облачной службы. Сервисы DSaaS должны обеспечивать хранение данных без потери информации в связи со стихийными бедствиями, человеческим фактором или техническими ошибками. Для этого можно сделать несколько реплик или резервных копий данных, которые могут предлагаться как часть сервиса или могут быть реализованы потребителем облачной службы с использованием функциональных возможностей сервиса DsaaS;

б) доступность. Сервисы DSaaS обеспечивают хранение и извлечение данных по запросу для удовлетворения потребностей приложений и систем потребителя облачной службы;

в) безопасность. Сервисы DSaaS должны надежно хранить информацию. В частности, несанкционированный доступ к данным потребителей сервиса облачных вычислений должен быть невозможен. Желательно обеспечить шифрование данных, хотя эту задачу может выполнить потребитель облачной службы, поскольку шифрование связано с затратами и снижением скорости работы;

г) фиксированные затраты. При использовании DSaaS заказчик обычно платит только за фактически используемые ресурсы для хранения данных. Для некоторых сервисов DSaaS информация, которая используется реже, может храниться в более дешевых системах хранения данных, которые, в связи с невысокой стоимостью, могут обеспечить не такую высокую скорость доступа к данным;

д) широкие возможности для управления. DSaaS поставщика облачной службы располагает политиками и процессами управления жизненным циклом хранения данных, которые позволяют пользователям и разработчикам сосредоточиться на решении прикладных задач и не беспокоиться об управлении информацией.

DSaaS, как показано в таблицах 1 и 2, можно разделить по типу хранения и по категории сервисов. Каждый сервис хранения имеет один или несколько сервисных интерфейсов, таких как драйвер блочного устройства, интерфейс файловой системы или API хранилища объектов, которые используются клиентским ПО. Эти API работают через сеть, поскольку в большинстве случаев предполагается, что функциональные возможности для хранения данных существуют в системе, отличной от системы, на которой запущено клиентское ПО.

Таблица 1 — DSaaS по типу хранения данных

Сервис	Описание
Сервис хранения файлов	Сервисы хранения файлов предлагают хранение с использованием обычной модели файловой системы, при которой файлы содержатся в каталогах внутри томов. Сервис хранения обычно предоставляется для клиентского ПО с использованием протокола NFS (NFS 4.2 — IETF 7862), а соответствующие тома монтируются в клиентскую среду с помощью драйвера клиента NFS. В частности, сервисы хранения файлов можно монтировать в ВМ и контейнеры, чтобы обеспечить долговременные возможности хранения данных в этих средах. Сервисы хранения файлов обычно работают по сети [как устройства сетевого хранения (NAS)] и могут использоваться несколькими клиентами одновременно. Поскольку хранилище виртуализируется сервисом хранения файлов, оно поддерживает широкие возможности для масштабирования и отличается исключительной долговечностью.

Окончание таблицы 1

Сервис	Описание
Сервис хранения файлов	Дублированные копии файлов хранятся в разных, физически разделенных, местах. Хранящиеся данные могут быть зашифрованы (см. [5]). Шифрование данных должно осуществляться с применением защищенных протоколов, в которых используются криптографические механизмы, определяемые национальными документами по стандартизации Российской Федерации, таких как TLS 1.2/TLS 1.3 (см. [6], [7]) или IPSec (см. [8], [9]). Многим приложениям необходим доступ к общим файлам и файловой системе. Такой тип сервисов хранения данных поддерживается в основном сетевыми хранилищами (NAS). Этот тип сервисов идеально подходит для таких вариантов использования, как крупномасштабные репозитории контента, среды разработки, медиахранилища или домашние каталоги пользователей
Сервис хранения объектов	Сервисы хранения объектов хранят данные в виде объектов данных в плоском, неиерархическом пространстве имен (пул хранения, корзина или контейнер), где каждый объект имеет уникальный идентификатор или ключ. По сути, сервис хранения объектов работает на основе модели «ключ — значение», где в качестве объекта выступает значение. Кроме того, каждый объект данных может иметь произвольный объем пользовательских метаданных, связанных с ним. Потенциально это гораздо больший объем, чем тот, который доступен в стандартных файловых системах. Сервисы хранения объектов обладают высокой масштабируемостью, поскольку они не привязаны к конкретному аппаратному обеспечению для хранения и могут включать несколько устройств хранения. Отдельные объекты также могут иметь очень большой объем. Сервисы хранения объектов могут одновременно использоваться для хранения и могут включать несколько устройств хранения.
Сервис хранения объектов	Отдельные объекты также могут иметь очень большой объем. Сервисы хранения объектов могут одновременно использоваться несколькими клиентскими приложениями. Сервисы хранения объектов обычно предлагаются через REST API, который доступен клиентам удаленно через сетевое подключение. REST API (например, Amazon S3 API) не похож на интерфейс обычной файловой системы, поэтому клиентские приложения должны быть специально разработаны и написаны для использования сервисов хранения объектов. Сервисы хранения объектов особенно удобны для использования с неструктурированными данными (например, изображениями), которые обновляются относительно редко (обновление происходит путем замены всего объекта новой версией). Сервисы хранения объектов также обычно работают медленнее, чем сервисы хранения файлов
Сервис блочного хранения данных	Сервисы блочного хранения данных обеспечивают широкополосный доступ с низкой задержкой к устройствам хранения данных на уровне блоков. Эти сервисы предоставляют клиентской системе аналог систем Direct Attached Storage (DAS) и Storage Area Network (SAN). Эта низкоуровневая форма доступа к ресурсам хранения предлагает заказчикам более развитые средства контроля и потенциально более высокую скорость работы, чем другие виды сервисов DSaaS. Таким образом, использование сервиса блочного хранения данных можно сравнить с подключением к клиентской системе дополнительных аппаратных устройств хранения данных. Сервисы блочного хранения обычно предназначены для работы в пределах одного центра обработки данных, поскольку при доступе к сервисам через удаленные сети существенно возрастает задержка.
Сервис блочного хранения данных	Обычно для доставки сервисов блочного хранилища по сети используются специализированные протоколы SAN, такие как iSCSI (IETF 7143). Удаленные устройства хранения данных представляются клиентскому ПО как смонтированный том, точно так же, как если бы это был диск, локально подключенный к системе, на которой запущено клиентское ПО. Сервисы блочного хранения могут иметь файловые системы, построенные поверх них клиентским ПО, или же клиентское ПО может использовать блочный интерфейс напрямую. Второй вариант используется, когда клиентское ПО является, например, ПО базы данных (например, база данных SQL) или ПО потоковой обработки (например, Apache Kafka)

Существуют категории облачных сервисов, которые изначально основаны на функциях хранения данных, но их функциональные возможности в этой области не ограничиваются просто хранением данных, а их интерфейсы сервисов, доступные клиентам, обычно соответствуют конкретным требованиям

клиентского ПО. Эти категории облачных сервисов, описанные в таблице 2, используют один или несколько типов хранилищ, описанных в таблице 1, но функция хранения не является основной функциональной возможностью, предоставляемой клиентам.

Таблица 2 — Преимущества для потребителей облачных служб

Сервисы хранения данных	Отличительные особенности
Сервисы баз данных NoSQL	Сервисы баз данных NoSQL предлагают возможности хранения и извлечения различных форм неструктурированных данных, таких как документы, изображения, фильмы и большие двоичные данные. В этой категории имеется широкий спектр базовых технологий, которые можно классифицировать различными способами, как, например: - кеш типа «ключ — значение»; - хранилище типа «ключ — значение»; - хранилище типа «ключ — значение» (согласованное в конечном счете); - хранилище типа «ключ — значение» (упорядоченное); - сервер типа «данные — структуры»; - хранилище кортежей; - база данных объектов; - хранилище документов; - база данных с широким значением столбца (Wide Column Store); - нативная многомодельная база данных; - графовая база данных
Сервисы баз данных SQL	SQL (или реляционные) сервисы баз данных хранят структурированные данные в табличном формате, что позволяет создавать потенциально сложные запросы для извлечения данных в соответствии с требованиями клиента и выполнять динамическое обновление содержимого базы данных. SQL — это интерактивный язык программирования, который создан на базе отраслевых стандартов и предназначен для отправки запросов, обновления и управления данными и наборами данных в системе управления базами данных
Сервисы баз данных SQL	Стандарт SQL (см. [10]). Современные базы данных SQL поддерживают обнаружение столбцов в широком диапазоне наборов данных: не только в реляционных таблицах/представлениях, но и в XML, JSON, пространственных объектах, объектах в виде изображений (двоичных объектах большого размера и символьных объектах большого размера) и в семантических объектах. Существует множество различных технологий баз данных SQL, которые предлагаются в виде облачных сервисов
Сервис хранения очередей сообщений	Функциональные возможности очередей сообщений доступны в виде облачных сервисов и обычно связаны с распределенной асинхронной формой обработки и системной архитектурой, которая получает все большее распространение. Многие системы очередей сообщений имеют возможность сохранять сообщения в хранилище. Сообщения могут сохраняться не только для того, чтобы не потерять их и извлечь при необходимости, но также для содействия обработке потоков, которая требует крупномасштабного анализа множества событий для извлечения полезной практической информации. Сохранение сообщений может создавать большую нагрузку на системы хранения данных из-за большого объема и высокой скорости доставки сообщений
Сервисы на основе технологии блокчейн и распределенных реестров	Облачные сервисы на основе технологии распределенных реестров (DLT), такие как облачные сервисы блокчейн, поддерживают предоставление и использование распределенных реестров, которые представляют собой одну из форм базы данных транзакций. Облачные сервисы DLT обычно предоставляют
Сервисы на основе технологии блокчейн и распределенных реестров	функциональные возможности для запуска узла DLT, включая экземпляр ПО платформы DLT и возможности хранения реплик самого распределенного реестра

Окончание таблицы 2

Сервисы хранения данных	Отличительные особенности
Аналитические сервисы	Функции аналитической обработки основаны на обработке большого количества данных. Такие большие объемы данных и их высокая скорость обработки требуют определенной поддержки со стороны сервисов хранения, в которых размещены данные. Многие поставщики облачных служб предоставляют специализированные сервисы облачного хранения данных, которые поддерживают анализ данных
Сервисы управления файлами	Сервис управления файлами представляет собой тип облачного сервиса с прикладными функциональными возможностями, который обеспечивает автоматическую репликацию файлов между пользовательскими устройствами и облачным хранилищем и позволяет нескольким пользователям совместно использовать и обновлять файлы
Федеративные сервисы хранения данных	В случае федеративных сервисов хранения данных ресурсы хранения могут быть прозрачно объединены между различными облачными хранилищами, включая локальные, частные или публичные облачные среды, независимо от того, предлагаются ли функции хранения файлов, блоков или объектов. Следует иметь в виду, что простые возможности репликации или восстановления после сбоя обычно не означают, что сервис хранения является федеративным
Примечание — Данные сервисы хранения данных могут предоставляться как постоянное хранилище или как хранилище в оперативной памяти.	

12.3 Типы функциональных возможностей DSaaS

В ГОСТ ISO/IEC 17788—2016, подраздел 6.4 определены следующие три различных типа функциональных возможностей:

- инфраструктурный (как в IaaS);
- платформенный (как в PaaS);
- прикладной (как в SaaS).

DSaaS может предлагать один или несколько из этих типов функциональных возможностей, как приведено в таблице 3.

Таблица 3 — Типы функциональных возможностей DSaaS

Тип функциональной возможности	Сервисы
Инфраструктурный	Сервис хранения файлов. Сервис хранения объектов. Сервис блочного хранения данных. Федеративный сервис хранения данных
Платформенный	Программируемое заказчиком хранилище данных, в которое можно загрузить написанный заказчиком код и использовать его для управления хранилищем данных. Аналитический сервис. Сервис базы данных NoSQL. Сервис базы данных SQL. Сервис очередей сообщений. Сервис на основе технологии блокчейн и DLT
Прикладной	Пользовательский интерфейс, ориентированный на человека, для управления хранилищем, например веб-хранилищем документов: - сервис управления файлами

Типы облачных функциональных возможностей, применяемые к облачным сервисам, описанным в таблицах 1 и 2, варьируются (см. таблицу 3). Большинство облачных сервисов, описанных в таблице 1, как правило, предоставляются в виде облачных сервисов инфраструктурного типа. Облачные сервисы, перечисленные в таблице 2, как правило, относятся к платформенному или прикладному типу, за исключением федеративных сервисов хранения данных, которые, как правило, относятся к инфраструктурному типу.

12.4 Важные дополнительные функциональные возможности DSaaS

Помимо ожидаемой функциональной возможности для хранения данных, многие сервисы облачного хранения предлагают другие дополнительные функциональные возможности, которые могут оказаться полезными потребителям облачных служб.

Первый набор таких функциональных возможностей связан с устойчивостью и сопротивляемостью к точечным сбоям в инфраструктуре поставщиков облачных служб. Многие облачные сервисы хранения данных хранят данные в нескольких резервных репликах, чтобы отказ одного устройства хранения или невозможность доступа к одному устройству не привели к недоступности сервиса или, что еще хуже, к потере данных. Характер репликации может быть различным. В некоторых случаях реплики специально размещаются в отделенном на физическом уровне месте (например, в другом центре обработки данных или в другой зоне доступности в пределах одного центра обработки данных) с целью устранения крупных сбоев в рамках всего центра обработки данных. В других случаях, в частности при использовании облачных сервисов с высокоскоростным доступом, места репликации могут, наоборот, располагаться в непосредственной близости друг от друга. Некоторые облачные сервисы хранения данных предоставляют возможность потребителю сервиса облачных вычислений самостоятельно выбирать политику размещения реплик.

Второй набор функциональных возможностей относится к созданию и хранению резервных копий данных. Такое резервное копирование может выполняться автоматически или по запросу потребителя облачной службы. Резервное копирование может осуществляться в объект с подключением к сети интернет (т. е. хранилище, которое доступно в режиме онлайн, но размещено в другом месте) или без подключения к сети интернет. Второй вариант может использоваться для длительного и недорогого хранения данных.

Еще одной функциональной возможностью, предлагаемой отдельными облачными сервисами хранения данных, является близость ресурсов. Речь идет об относительном физическом размещении экземпляров облачных сервисов хранения данных по отношению к другим облачным сервисам, в основном к экземплярам вычислительных сервисов, таким как ВМ и контейнеры. Одной из основных причин размещения экземпляров вычислительных сервисов рядом с экземплярами сервисов хранения является поддержание высокой скорости работы как с точки зрения снижения задержек, так и с точки зрения увеличения пропускной способности для данных, передаваемых между экземплярами вычислительных систем и систем хранения. Некоторые категории сервисов, например сервисы блочного хранения данных, часто предлагаются только в таком виде. Например, сервисы блочного хранения могут предлагаться только для использования экземплярами вычислительных сервисов, работающими на узлах в том же центре обработки данных, где между узлом хранения и вычислительным узлом имеется высокоскоростной канал, например оптоволоконный или Ethernet.

13 Создание сетевых подключений в облачных вычислениях

13.1 Ключевые аспекты создания сетевых подключений

Сетевое подключение является ключевым элементом облачных вычислений. Само определение облачных вычислений основано на том, что доступ к их функциональным возможностям осуществляется через сеть: обеспечение сетевого доступа к масштабируемому и адаптивному пулу совместно используемых физических или виртуальных ресурсов с поддержкой самообслуживания и администрирования по требованию (см. ГОСТ ISO/IEC 17788).

Сети и сетевые функциональные возможности также являются одними из ресурсов, которые часто предоставляются с помощью облачных сервисов.

Таким образом, существуют два общих вопроса, связанных с сетями в облачных вычислениях. Первый — это сетевое подключение, с помощью которого осуществляется доступ к данному облачному сервису и доступ к любым функциональным возможностям внутри облачного сервиса, например к приложению, запущенному в рамках вычислительного сервиса. Оно называется «сетевое подключение для доступа к облаку» или «сетевое подключение для доступа к публичному облаку», если речь идет о публичных облачных сервисах. Второй — это сетевое подключение, используемое для соединения экземпляров облачных сервисов друг с другом. Это сетевое подключение не должно быть доступно за пределами конкретных экземпляров облачных сервисов. Оно называется «внутриоблачное сетевое подключение».

13.2 Сетевое подключение для доступа к облаку

Облачные сервисы имеют доступные извне интерфейсы, которые позволяют использовать их функциональные возможности пользователям облака и системам, действующим от имени пользователей облака. Некоторые облачные сервисы также имеют доступные извне интерфейсы с поддержкой функциональных возможностей потребителя облачной службы, работающих внутри облачного сервиса. Среди примеров можно упомянуть такие интерфейсы, как веб-интерфейсы или API для приложений потребителя облачной службы, работающих внутри экземпляра вычислительного сервиса (например, внутри ВМ или контейнера), а также интерфейсы для хранения данных экземпляра сервиса хранения (например, файлового хранилища).

Как правило, доступные извне интерфейсы видны в интернете, по крайней мере для публичных облачных сервисов. Однако в некоторых случаях доступные извне интерфейсы могут быть намеренно скрыты от посторонних глаз в интернете, даже если доступ пользователей осуществляется через интернет. Частные облачные сервисы могут быть развернуты таким образом, что будут доступны только в пределах частных сетей организации (т. е. такие облачные сервисы недоступны через интернет). При этом может возникнуть ситуация, когда отдельные интерфейсы должны быть доступны извне. Например, когда веб-приложение, развернутое на частном облачном сервисе, должно представлять пользователям общедоступный интерфейс.

Приложениям, работающим в облачном сервисе, могут потребоваться видимые извне интерфейсы, доступные по общедоступному сетевому адресу и номеру порта. Обычно такой общедоступный интерфейс предоставляется в виде «виртуального интерфейса», где внешний интерфейс представлен на сетевом адресе и порте, известным самому облачному сервису, а код, выполняемый внутри облачного сервиса, работает на некотором внутреннем сетевом адресе и порте, на которые сопоставлены внешний адрес и порт. Такая виртуализация конечных точек необходима для предоставления возможности для совместного использования ресурсов, что является основной отличительной особенностью облачных сервисов. Виртуализация конечных точек также поддерживает такие функциональные возможности, как балансировка нагрузки в рамках нескольких экземпляров приложения и функции обеспечения безопасности, включая межсетевые экраны и блокировку DDoS-атак.

Для общедоступных интерфейсов также может потребоваться специальная настройка конфигурации. В частности, организации, приложение которой работает в облачном сервисе, может понадобиться, чтобы адрес, используемый для интерфейса, принадлежал ей, а не поставщику облачной службы. Эта концепция называется «Bring Your Own IP addresses» (BYOIP). Она дает возможность потребителю облачной службы настроить общедоступный интерфейс для приложения, работающего в облачном сервисе, с адресом, принадлежащим потребителю облачной службы. Подобного рода настройка может использоваться как для публичных облачных сервисов, так и для частных облачных сервисов.

13.3 Внутриоблачное сетевое подключение

Для сред облачных сервисов характерно использование сетевого подключения для соединения различных компонентов, составляющих систему. В облачных сервисах вычислительного типа (например, при работе ПО в ВМ или контейнерах) часто существует несколько экземпляров, которые должны обмениваться данными либо друг с другом (например, когда приложение разделено на отдельно работающие компоненты, как в случае с архитектурой микросервисов), либо с другими компонентами решения (например, с балансировщиком нагрузки, где используется горизонтальное масштабирование с несколькими параллельными экземплярами данного компонента). Для сервисов хранения данных также характерно подключение к вычислительным экземплярам, и, как правило, это делается с помощью сетей.

Также может существовать несколько отдельных уровней внутриоблачных сетевых подключений. Уровень приложений, описанный выше, а также уровень управления или контроля, используемый для мониторинга и управления всеми облачными сервисами. Эти уровни намеренно изолированы друг от друга, чтобы не пересекаться.

Обычно внутриоблачные сетевые подключения виртуализируются. Различные облачные сервисы не используют сетевые функциональные возможности напрямую. Вместо этого они используют виртуализированные сетевые функциональные возможности, которые позволяют совместно использовать сетевые ресурсы, а также обеспечивают изоляцию между различными группами экземпляров облачных сервисов как в целях безопасности, так и во избежание нарушений в работе.

Структура и организация виртуализированных сетей также может намеренно не отражать организацию базовых физических сетей. Часто бывает так, что компоненты одного решения распределены по нескольким зонам доступности в одном центре обработки данных или по нескольким центрам обработки данных. В некоторых обстоятельствах крайне нежелательно, чтобы эта физическая организация была видна компонентам решения, поэтому этим компонентам представляется единая унифицированная виртуальная сеть, наложенная на физические сети.

Виртуализированные вычислительные среды — как ВМ, так и контейнеры, — предполагают жесткий контроль и виртуализацию сетевых ресурсов, включая конечные точки, открытые каждой ВМ или контейнером, и целевые сетевые точки, используемые ПО, работающим в этих средах. Возможность запуска нескольких ВМ в одной системе или нескольких контейнеров в одной ОС, безусловно, требует сопоставления каждой из сетевых конечных точек, открытых ПО, с реальными конечными точками в содержащей системе, чтобы обеспечить совместное использование системы без возникновения конфликтов. Развертывание как ВМ, так и контейнеров требует настройки конфигурации для решения таких проблем.

Кроме того, в распределенных средах часто используются как ВМ, так и контейнеры. Несколько экземпляров одного и того же ПО могут работать на разных системах, и эти системы могут находиться в разных физических местах. Приложения также обычно разделяются на несколько независимых компонентов (сервисов, микросервисов), и эти компоненты могут работать в отдельных местах. Эффективным решением является скрытие фактического расположения компонентов от ПО, поскольку компоненты приложения должны беспрепятственно обмениваться данными друг с другом, где бы они ни работали. Еще лучше, если компоненты приложения могут обмениваться данными только друг с другом. Внешний обмен данными должен тщательно контролироваться с помощью специально заданных внешних конечных точек.

Оптимальным решением в отношении сетевого подключения этих программных архитектур будет определение сетевого подключения на уровне приложений. Это виртуальная сеть, которая используется только компонентами приложения и прозрачно охватывает все места, в которых работают компоненты приложения. Подразумевается, что каждое приложение имеет соответствующую виртуальную сеть и каждая виртуальная сеть изолирована от других сетей, как в случае с виртуальными вычислительными средами, такими как контейнеры.

Виртуальные сети могут быть построены с использованием различных методов и технологий, включая программно-определяемые сети (SDN) и виртуализацию сетевых функций (NFV).

13.4 VPN и облачные вычисления

VPN представляют собой одну из технологий, которые могут использоваться при создании решений на основе облачных вычислений. Сети VPN обеспечивают защищенное сетевое соединение систем в тех случаях, когда определенная часть сетевой инфраструктуры использует ненадежные среды. Существует две типовые конфигурации сетей VPN (см. рисунок 9), которые предназначены для разных сценариев использования:

- конфигурация «узел — шлюз» (Host-to-Gateway). При этой конфигурации автономная система, как правило, клиентский компьютер или устройство получает удаленный доступ к защищенной сети. В контексте облачных вычислений типичным сценарием использования является доступ клиентского устройства к облачным сервисам, приложениям и другим ресурсам, работающим в них;
- конфигурация «шлюз — шлюз» (Gateway-to-Gateway). При этой конфигурации защищенный сетевой обмен данными осуществляется между двумя отдельными защищенными сетями. При типичном сценарии использования облачные сервисы в облачной среде либо должны быть подключены к внутренним приложениям и системам потребителя облачной службы, либо должны быть подключены к другим облачным сервисам, работающим в другой облачной среде.

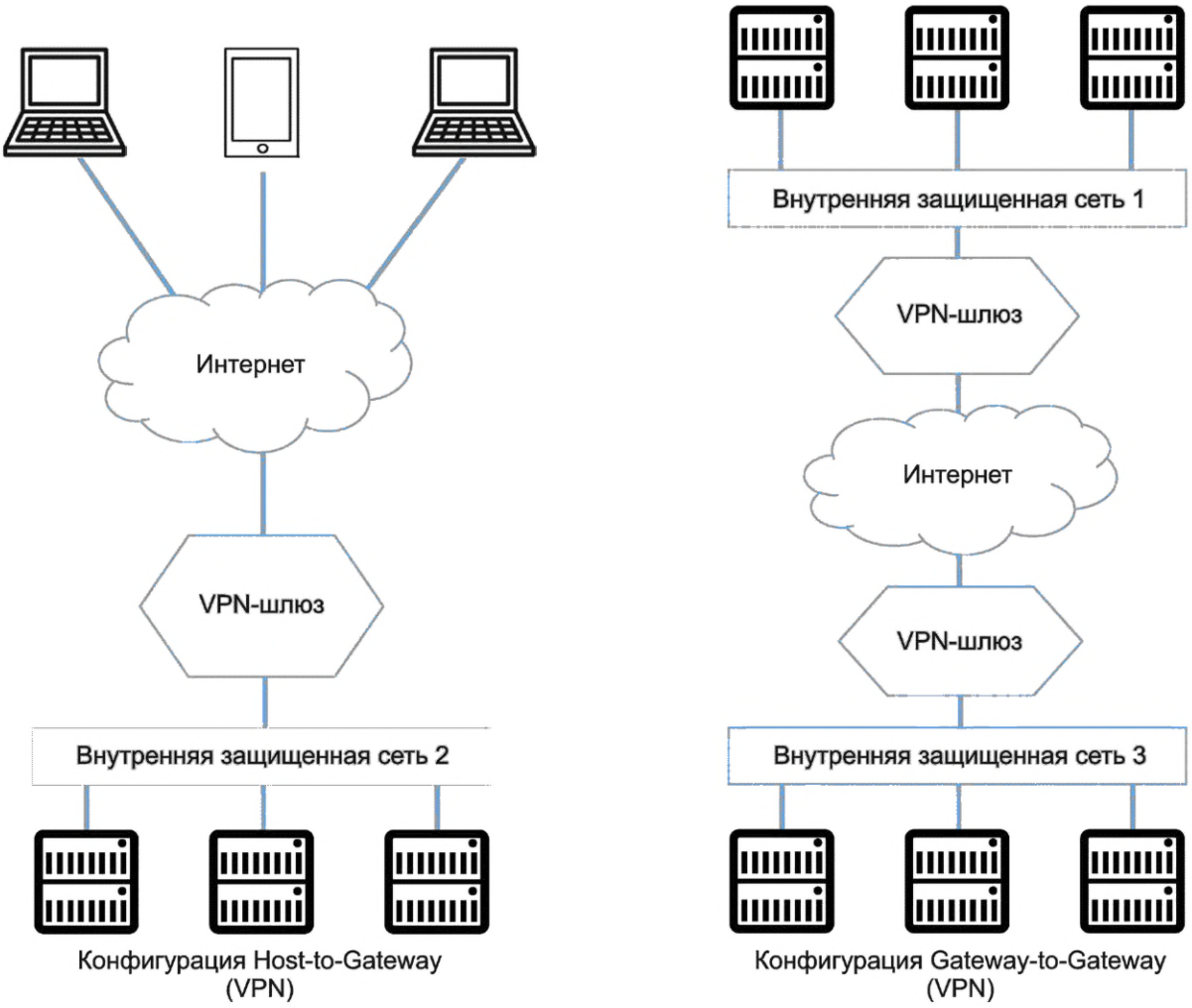

Рисунок 9 — Типы конфигурации сетей VPN

Для создания сетей VPN используются различные технологии, такие как L2TP/IPsec и OpenVPN. Публичными поставщиками облачных служб, как правило, поддерживаются один или несколько типов сетей VPN.

14 Масштабируемость облачных вычислений

14.1 Подходы к обеспечению масштабируемости

В ГОСТ ISO/IEC 17788 одними из ключевых характеристик облачных вычислений являются динамичная адаптивность и масштабируемость. Динамичная адаптивность и масштабируемость описываются как отличительные особенности облачных вычислений, которые позволяют быстро и адаптивно регулировать физические или виртуальные ресурсы, в некоторых случаях автоматически, для оперативного увеличения или уменьшения объема ресурсов.

Адаптивность — это способность облачных сервисов подстраиваться под изменения рабочей нагрузки посредством выделения и удаления ресурсов (обычно в автоматическом режиме) так, чтобы имеющиеся ресурсы максимально точно соответствовали текущей потребности в ресурсах. Масштабируемость — это способность облачных сервисов увеличивать или уменьшать объем ресурсов, выделяемых для обработки определенной рабочей нагрузки.

Масштабируемость разделяется на две большие категории: горизонтальное масштабирование и вертикальное масштабирование.

Горизонтальное масштабирование. В этой категории несколько ресурсов используются параллельно, причем количество параллельно используемых ресурсов меняется для поддержания необходимого объема ресурсов. В качестве вычислительных ресурсов используется несколько компонентов (физические компьютеры, ВМ, контейнеры). В качестве ресурсов хранения данных используется несколько устройств хранения (несколько накопителей). В качестве сетевых ресурсов используется несколько сетей.

Вертикальное масштабирование. В этой категории объем отдельного ресурса меняется для поддержания необходимого объема ресурсов. Для вычислительных ресурсов это означает изменение количества процессоров или объема оперативной памяти данного ресурса. Для ресурсов хранения данных это означает изменение объема устройства (накопитель большей или меньшей емкости) или изменение скорости операций чтения/записи, доступной на ресурсе. Для сетевых ресурсов это означает переход на сеть с большей или меньшей пропускной способностью.

Вертикальное масштабирование, как правило, проще с точки зрения рабочей нагрузки и конструкции решения. Однако вертикальное масштабирование имеет свои ограничения, поскольку отдельные физические вычислительные ресурсы имеют определенный лимит в отношении количества доступных процессоров и максимального объема доступной оперативной памяти. Виртуализация одной вычислительной нагрузки на нескольких физических вычислительных ресурсах не является целесообразной, поэтому лимиты физических вычислительных ресурсов напрямую ограничивают возможность вертикальной масштабируемости компьютерных вычислений, доступной для облачных сервисов. Что касается ресурсов хранения данных, то отдельные устройства хранения имеют ограничения по емкости, однако виртуализированные облачные сервисы хранения данных могут использовать несколько физических ресурсов хранения данных, но воспринимать их как единый ресурс, что обеспечивает широкие возможности для увеличения вертикальной масштабируемости. Для сетевых ресурсов одна физическая сеть имеет ограничения по пропускной способности. Виртуализированный сетевой сервис может использовать несколько параллельных сетевых ресурсов, что обеспечивает широкие возможности для увеличения вертикальной масштабируемости.

Горизонтальное масштабирование, когда несколько экземпляров ресурса используются параллельно, обычно требует, чтобы рабочая нагрузка и решение имели конструкцию, которая специально адаптирована для этого типа масштабирования. Для решения проблем, связанных с горизонтальным масштабированием, используются специальные методы. В случае с ресурсами хранения виртуализация базовых ресурсов хранения данных может скрыть наличие нескольких физических устройств хранения. Тем не менее такой виртуализированный ресурс хранения данных все равно может раскрывать характеристики базовых физических ресурсов, такие как ограничение на размер отдельных файлов или объектов, поскольку может случиться так, что такие объекты хранения данных не могут охватывать несколько физических устройств. При проектировании конструкции решений, предусматривающих использование горизонтального масштабирования, такие ограничения учитываются. Горизонтальное масштабирование сетевых ресурсов подразумевает, что конструкция решения использует несколько отдельных соединений через различные доступные сетевые ресурсы. Однако виртуализированный сетевой сервис может прозрачно использовать несколько базовых сетевых ресурсов, что устраняет необходимость в непосредственном учете этого фактора в конструкции решения.

Горизонтальное масштабирование вычислительных ресурсов широко используется в сфере облачных вычислений по той причине, что виртуализировать границы вычислительных ресурсов невозможно с технической точки зрения. Горизонтальное масштабирование вычислительных ресурсов подразумевает, что несколько экземпляров данного программного компонента выполняются параллельно, а поступающая нагрузка распределяется между этими экземплярами. Такая конструкция открыто демонстрирует ограничения по вычислительным ресурсам (количество процессоров и объем оперативной памяти), поэтому выбор делается в пользу подхода к проектированию, в рамках которого ПО разделяется на отдельные компоненты, выполняемые в отдельных процессах. Это позволяет уменьшить количество процессоров и объем оперативной памяти, необходимой для каждого компонента. В таких подходах к проектированию, как создание архитектуры микросервисов, специально учитывается этот аспект горизонтального масштабирования.

14.2 Параллельные экземпляры и балансировка нагрузки

Если при использовании горизонтального масштабирования вычислительных ресурсов несколько параллельных экземпляров программного компонента обрабатывают входящие запросы, возникают определенные проблемы, с которыми должен справиться проект решения.

Основная проблема заключается в том, что входящие запросы, поступающие в определенный программный компонент, должны распределяться между всеми экземплярами этого программного компонента. Логически данную ситуацию можно рассматривать как обработку потока запросов, направленных на определенную конечную точку сервиса (формально эта конечная точка имеет сетевой адрес, на который и оправляются запросы). Несколько параллельных экземпляров программного компонента, реализующих сервис, имеют свою собственную (частную) конечную точку, каждая из которых имеет свой сетевой адрес и комбинацию портов. Компонент, который называется балансировщиком нагрузки, используется для управления конечной точкой сервиса и направления каждого входящего запроса в один из параллельных экземпляров.

Балансировщик нагрузки подключается к конечной точке сервиса и обрабатывает все входящие запросы. Балансировщик нагрузки ведет список всех экземпляров соответствующего программного компонента. При этом балансировщику нагрузки должен быть известен адрес конечной точки и порт каждого экземпляра. Каждый входящий запрос направляется на один из экземпляров программного компонента. Выбор экземпляра для входящего запроса определяется алгоритмом диспетчеризации (алгоритмы диспетчеризации могут быть различными).

Примером простого алгоритма диспетчеризации является циклический алгоритм (round-robin), когда каждый из экземпляров используется по очереди, а алгоритм стремится отправить одинаковое количество запросов на все экземпляры. Одной из разновидностей этого алгоритма является циклический взвешенный алгоритм (weighted round-robin), когда каждому из экземпляров присваивается весовой коэффициент, от которого зависит количество передаваемых ему экземпляров. Более сложный алгоритм диспетчеризации связан с рабочей нагрузкой. Он работает по принципу невыполненных запросов. При этом балансировщику нагрузки должно быть известно количество запросов, которые еще обрабатываются каждым экземпляром, и новые входящие запросы должны отправляться на наименее загруженный экземпляр.

Балансировщик нагрузки должен обрабатывать динамическое количество экземпляров. Могут запускаться новые экземпляры, а существующие могут останавливаться. Запуск и остановка могут выполняться в целях поддержания масштабирования и адаптивности либо вследствие отказа экземпляра.

Балансировщик нагрузки может быть реализован с использованием обратного прокси-сервера, который может предоставлять дополнительные функциональные возможности, такие как кеширование статического контента, обработка SSL (шифрование/дешифрование) и защита от атак.

14.3 Адаптивность и автоматизация

Адаптивность представляет собой подстраивание объема ресурсов, используемых конкретным компонентом, к изменениям рабочей нагрузки на этот компонент. Адаптивность максимально эффективно применяется с помощью инструментов автоматизации, хотя ее можно выполнять и в ручном режиме. Внедрение инструментов адаптивности в ручном режиме не самое лучшее решение, поскольку требуется постоянное наличие персонала для выполнения рабочих операций, и, скорее всего, эти операции будут выполняться медленнее.

Автоматическая адаптивность подразумевает постоянный мониторинг соответствующих компонентов на предмет использования их ресурсов инструментами адаптивности. Инструменты адаптивности соблюдают правила, связанные с использованием ресурсов, таким образом, что, если использование ресурсов превышает некоторые заданные уровни, предпринимаются действия по увеличению выделенных ресурсов, а если использование ресурсов падает ниже некоторых заданных уровней, предпринимаются действия по уменьшению выделенных ресурсов.

Правила регулировки адаптивности должны учитывать тот факт, что для выделения и удаления ресурсов требуется время. Это особенно важно при выделении дополнительных ресурсов, поскольку существует вероятность того, что текущие ресурсы будут полностью израсходованы до получения новых.

14.4 Масштабирование базы данных

Масштабирование базы данных является важным аспектом облачных вычислений. Масштабирование базы данных выполняется как в отношении физического размера базы данных, так и в отношении скорости выполнения операций (запросов и обновлений).

Распространенным подходом, используемым для масштабирования баз данных, является тот или иной вид горизонтального масштабирования с использованием нескольких ВМ и нескольких устройств хранения для одной базы данных. Основная проблема при использовании горизонтальной масшта-

бируемости заключается в необходимости обеспечения точной синхронизации нескольких ресурсов в отношении друг друга.

Одним из подходов к синхронизации ресурсов является разделение, или сегментирование, базы данных, так чтобы различные группы записей базы данных хранились и обрабатывались разными ресурсами. Входящие запросы разделяются и передаются каждому соответствующему ресурсу, а результаты, полученные от каждого ресурса, объединяются перед отправкой ответа на запрос.

Другой подход к синхронизации ресурсов заключается в использовании технологии кластеризации, которая располагает ресурсы поблизости друг от друга (с точки зрения скорости обмена данными между ресурсами) и позволяет осуществлять управление с помощью монитора глобальных транзакций, который стремится поддерживать согласованность между различными ресурсами. Эти два подхода обеспечивают так называемую строгую согласованность между различными ресурсами: клиент, использующий такую базу данных, всегда получает один и тот же результат независимо от того, какие ресурсы используются для обработки запроса. Строгая согласованность необходима для транзакционных приложений — для них упорядочение транзакций является важным условием.

Для облачных вычислений существует принципиально другой подход к масштабированию баз данных. Этот подход использует концепцию окончательной согласованности, которая может применяться для не транзакционных типов приложений. В таких случаях реплики базы данных хранятся в нескольких местах и обновления выполняются независимо для каждой из реплик, что может привести к возникновению различий между содержимым реплик в течение определенного периода времени, а также к конфликту обновлений. Принцип окончательной согласованности используется в отдельных базах данных NoSQL, таких как открытая база данных CouchDB.

15 Безопасность и общие технологии облачных вычислений

15.1 Общие положения

Безопасность является ключевым аспектом всех систем облачных вычислений. Безопасность включает в себя набор средств контроля, которые обладают функциональными возможностями для устранения различных рисков. Описанные здесь средства контроля являются общими в том смысле, что они могут применяться и к необлачным системам. Особенности их применения в облачных вычислениях приведены в 15.2.

15.2 Межсетевые экраны

Межсетевой экран — это компонент системы сетевой безопасности, который отслеживает и контролирует входящий и исходящий сетевой трафик, разрешая или блокируя определенный трафик на основе набора правил безопасности. Цель межсетевого экрана — создать барьер между доверенными защищенными и контролируемыми внутренними сетями и ненадежными внешними сетями, по сути, предотвращая несанкционированный и неконтролируемый доступ к внутренним сетям.

В контексте облачных вычислений часто бывает так, что ненадежными внешними сетями являются сеть интернет или другие сети доступа к облачным средам (см. 13.2). Доверенные защищенные сети — это сети, используемые для внутриоблачных сетевых подключений (см. 13.3).

Потребитель облачной службы может развернуть в облачной среде собственное ПО межсетевого экрана, используя соответствующий облачный сервис IaaS. Этот вариант эффективен, если потребитель облачной службы желает использовать определенное ПО межсетевого экрана либо если потребителю необходим детальный контроль конфигурации межсетевого экрана.

Однако более распространенная ситуация заключается в том, что межсетевые экраны предоставляются поставщиком облачной службы в качестве облачного сервиса в рамках облачной среды. Межсетевые экраны, предоставляемые поставщиком облачной службы, могут быть как аппаратными, так и программными, и окончательное решение остается за поставщиком.

Межсетевые экраны обычно устанавливаются на всех конечных точках, через которые проходят подключения из внешних сетей к внутренним сетям.

Что касается прикладных облачных сервисов, поставщик облачной службы, как правило, предоставляет функциональные возможности межсетевого экрана как часть облачного сервиса.

15.3 Защита конечных точек

Использование облачных сервисов обычно предполагает, что потребитель облачной службы делает одну или несколько конечных точек каждого решения в качестве видимых извне. Каждая такая

публичная конечная точка нуждается в защите, которая обычно предоставляется как часть облачного сервиса. Защита, как правило, включает в себя:

- межсетевые экраны;
- защиту от распределенных атак типа «отказ в обслуживании» (DDoS);
- сканирование уязвимостей.

15.4 Управление идентификацией и доступом пользователей

Контроль доступа к ресурсам облачных вычислений — это важная функциональная возможность для любых облачных вычислений. Задействованные ресурсы включают в себя не только сами облачные сервисы, но и компоненты потребителя облачной службы, такие как приложения, данные и сервисы, развернутые в облачных сервисах.

Сами облачные сервисы обычно предоставляются поставщиком облачной службы с возможностью управления идентификацией и доступом. Для других ресурсов функциональные возможности для управления идентификацией и доступом обычно предоставляются в виде облачного сервиса, предлагаемого поставщиком облачной службы. Один и тот же облачный сервис управления идентификацией и доступом зачастую может использоваться как для самих облачных сервисов, так и для ресурсов потребителя облачной службы.

К числу функциональных возможностей, которые следует учитывать при выборе облачного сервиса управления идентификацией и доступом, относятся:

- поддержка передовых методов аутентификации, включая биометрическую и многофакторную аутентификацию;
- поддержка интеграции локальных возможностей управления идентификацией и доступом с облачным сервисом;
- поддержка системы единого входа (SSO);
- поддержка детальной авторизации.

15.5 Шифрование данных

Шифрование данных является важной функцией для обеспечения конфиденциальности и защиты. В облачных вычислениях данные могут шифроваться при их хранении и перемещении.

Шифрование данных должно осуществляться с применением защищенных протоколов, в которых используются криптографические механизмы, определяемые национальными документами по стандартизации Российской Федерации, такие как TLS 1.2/TLS 1.3 (см. [6], [7]) или IPSec (см. [8], [9]). Функциональные возможности шифрования могут быть предоставлены облачными сервисами.

Шифрование данных при хранении может выполняться потребителем облачной службы (или программным кодом, установленным и управляемым потребителем), однако многие облачные сервисы хранения данных также предлагают возможность шифрования хранящейся информации.

15.6 Управление ключами

Использование шифрования обязательно предполагает использование ключей шифрования. Управление ключами шифрования необходимо для обеспечения надлежащей безопасности. Не следует хранить ключи вместе с приложением. В этом случае, например, злоумышленник сможет завладеть копией ключей и получить доступ к зашифрованным данным.

Функциональные возможности управления ключами часто предоставляются в виде облачных сервисов для интеграции с ресурсами, которые могут быть сами облачными сервисами или ресурсами, развернутыми потребителем облачной службы в облачных сервисах. Облачные сервисы по управлению ключами часто основаны на использовании поставщиком облачной службы аппаратных модулей безопасности для обеспечения надежной защиты ключей шифрования, которыми они управляют. Некоторые облачные сервисы управления ключами сами генерируют и хранят ключи, в то время как другие поддерживают концепцию создания собственных ключей (BYOK), в рамках которой потребитель облачной службы может генерировать необходимые ключи и передавать их облачному сервису для хранения и использования. Эту концепцию можно эффективно использовать при развертывании нескольких облаков, когда требования к шифрованию распространяются на несколько различных облачных сервисов. Примером может служить ситуация, когда реплика некоторых зашифрованных данных хранится в облачном сервисе, который отличается от исходного облачного сервиса (возможно, у другого поставщика облачной службы).

Библиография

[1] ИСО/МЭК 17789:2014 Информационные технологии. Облачные вычисления. Эталонная архитектура (Information technology — Cloud computing — Reference architecture)

[2] ИСО/МЭК 22123-1:2023 Информационные технологии. Облачные вычисления. Часть 1. Словарь (Cloud Computing Security — Cloud Computing — Part 1: Vocabulary)

[3] ИСО/МЭК 9660:2023 Обработка информации. Объем и файловая структура компакт-диска (CD-ROM) для обмена информацией (Information processing — Volume and file structure of CD-ROM for information interchange)

[4] ИСО/МЭК 13346-1:1995 Информационные технологии. Структура тома и файла на носителях с однократной записью и перезаписью с использованием непоследовательной записи для информационного обмена. Часть 1. Общие положения (Information technology — Volume and file structure of write-once and rewritable media using non-sequential recording for information interchange — Part 1: General)

[5] Р 1323565.1.042—2022 Информационная технология. Криптографическая защита информации. Режим работы блочных шифров, предназначенный для защиты носителей информации с блочно-ориентированной структурой

[6] Р 1323565.1.020—2020 Информационная технология. Криптографическая защита информации. Использование российских криптографических алгоритмов в протоколе безопасности транспортного уровня (TLS 1.2)

[7] Р 1323565.1.030—2020 Информационная технология. Криптографическая защита информации. Использование российских криптографических алгоритмов в протоколе безопасности транспортного уровня (TLS 1.3)

[8] Р 1323565.1.035—2021 Информационная технология. Криптографическая защита информации. Использование российских криптографических алгоритмов в протоколе защиты информации ESP

[9] МР 26.2.001—2022 Информационная технология. Криптографическая защита информации. Использование российских криптографических алгоритмов в протоколе обмена ключами в сети Интернет версии 2 (IKEv2)

[10] ИСО/МЭК 9075-1:2016 Информационная технология. Языки базы данных. SQL. Часть 1. Структура (SQL/структура) [Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)]

УДК 006.034:004.056.5:006.354 ОКС 35.210

Ключевые слова: облачные вычисления, виртуальные машины, контейнеры, виртуализированные сети, контейнерные службы, системы хранения данных

Редактор *Е.В. Якубова*
Технический редактор *В.Н. Прусакова*
Корректор *М.В. Бучная*
Компьютерная верстка *Е.А. Кондрашовой*

Сдано в набор 24.08.2023. Подписано в печать 04.09.2023. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 6,05. Уч.-изд. л. 5,51.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Создано в единичном исполнении в ФГБУ «Институт стандартизации»
для комплектования Федерального информационного фонда стандартов,
117418 Москва, Нахимовский пр-т, д. 31, к. 2.

gost.e-ecolog.ru gost.e-ecolog.ru