МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

WEB-дизайн и мультимедиа
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Исследование архитектур построения облачных систем»

Студент И.С. Рачицкая
(И.О. Фамилия) (личная подпись)

Руководитель канд. пед. наук, доцент, Т.А. Агошкова
(ученая степень, звание, И.О. Фамилия)

Консультант М. В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Тема выпускной квалификационной работы – «Исследование архитектур построения облачных систем».

Как показывает практика, эффективность и надежность облачной системы зависит от ее архитектуры.

Исследование архитектур построения облачных систем представляет научно-практический интерес.

Объектом исследования бакалаврской работы являются облачные системы.

Предметом исследования бакалаврской работы являются архитектуры построения облачных систем.

Цель выпускной квалификационной работы – исследование особенностей архитектур, используемых для построения облачных систем.

Методы исследования – облачные технологии, методы проектирования информационных систем.

Как показал анализ лучшими характеристиками обладает решение облачное решение Oracle, главными преимуществами которого являются гибкость и широкие функциональные возможности. Описан алгоритм и представлен фрагмент программного кода Java для создания WebLogic-сервиса.

Результаты бакалаврской работы представляют научно-практический интерес и могут быть рекомендованы разработчикам облачных систем.

Выпускная квалификационная работа состоит из 45 страниц текста с приложением, 19 рисунков, 5 таблиц и 22 источников.

Abstract

The topic of the present graduation work is Study of architectures to build cloud systems.

The author dwells on the architectures features used to build the cloud systems.

The graduation work consists of 19 figures, 5 tables and the list of 22 references.

The study of the architectures to build cloud systems is of scientific and practical interest.

It is highlighted that the effectiveness and reliability of a cloud system depends on its architecture.

The object of the graduation work is the cloud systems.

The subject of the graduation work is the architecture to build the cloud systems.

The goal of the investigation is to study the architectures features used to build the cloud systems.

To achieve the goal, cloud technologies and information systems design methods are applied.

The conducted analysis show that the Oracle cloud solution has the best performance. Its main advantages are flexibility and high functionality. The graduation work also describes the algorithm and presents a fragment of Java programme code to develop a WebLogic service.

In conclusion, we would like to underline that the results of the work can be recommended for developers dealing with cloud systems.

Оглавление

Введение

вычисления - это предоставление вычислительных услуг, включая серверы, хранилище, базы данных, сети, программное обеспечение, аналитику и интеллектуальные данные, через Интернет-облако, предлагая клиентам более быстрые инновации, гибкие ресурсы и минимизацию затрат при расширении бизнеса.

Компании платят только за используемые облачные сервисы, что способствует снижению эксплуатационные расходов, более эффективному управлению и масштабированию ИТ-инфраструктуры по мере изменения потребностей бизнеса.

Задачи обеспечения предоставления услуг облачных вычислений решаются с помощью облачных систем, которые представляет собой совокупность аппаратно- программного обеспечения и облачной ИТ-инфраструктуры.

Как показывает практика, эффективность и надежность облачной системы зависит от ее архитектуры.

Исследование архитектур построения облачных систем представляет научно-практический интерес.

Объектом исследования бакалаврской работы являются облачные системы.

Предметом исследования бакалаврской работы являются архитектуры построения облачных систем.

Цель выпускной квалификационной работы – исследование особенностей архитектур, используемых для построения облачных систем.

Для достижения данной цели необходимо выполнить следующие задачи:

– произвести анализ существующих архитектур построения облачных систем;

– проанализировать методологии проектирования архитектур

информационных систем и разработать модель логической архитектуры для построения облачной системы;

– проанализировать эталонные архитектуры промышленных платформ и разработать модель физической архитектуры для построения облачной системы.

Методы исследования – облачные технологии, методы проектирования информационных систем.

Практическая значимость бакалаврской работы заключается в разработке модели архитектуры облачной системы, обеспечивающей высокую эффективность предоставления услуг пользователям.

Данная работа состоит из введения, трех глав, заключения, списка используемой литературы и приложения.

Первая глава посвящена обзору и анализу архитектур построения облачных систем.

Вторая глава посвящена выбору метода и проектированию логической архитектуры для построения облачной системы.

В третьей главе описан процесс проектирования физической архитектуры для построения облачной системы на основе SOA.

В заключении описываются результаты выполнения выпускной квалификационной работы.

Приложение содержит фрагменты программного кода приложения.

Бакалаврская работа состоит из 45 страниц текста с приложением, 19 рисунков, 5 таблиц и 22 источников.

Глава 1 Обзор и анализ архитектур построения облачных систем

1.1 Постановка задачи исследования

Архитектура облачных вычислений относится к компонентам и подкомпонентам, необходимым для облачных вычислений. Эти компоненты обычно состоят из интерфейсной платформы (толстый, тонкий или мобильный клиент), серверных платформ (серверы, хранилище), облачной доставки и сети (Интернет, Интранет, Intercloud) [15].

Вместе эти компоненты составляют архитектуру облачных вычислений.

Следует отметить, что в настоящее время в российской и зарубежной практике проектирования информационных систем (ИС) термин «архитектура системы» используется очень широко, но при этом имеет столь же широкое множество различных трактовок.

Наиболее популярным является следующее определение: «Архитектура ИС - организационная структура и связанное с этой структурой поведение ИС» [11].

Архитектура облачной системы состоит из фронт-энда и бэк-энда [14].

Фронт-энд развернут на стороне клиента содержит клиентские интерфейсы и приложения, необходимые для доступа к платформам облачных вычислений. Фронт-энд включает в себя веб-браузеры (Firefox, Internet Explorer и др.), тонкие и толстые клиенты, планшеты и мобильные устройства [13].

Бэк-энд используется поставщиком услуг. Он управляет всеми ресурсами, необходимыми для предоставления услуг облачных вычислений.

Бэк-энд включает в себя огромное количество хранилищ данных, механизм безопасности, виртуальные машины, модели развертывания, серверы, механизмы управления трафиком и др.

Архитектура типовой облачной системы представлена на рисунке 1.

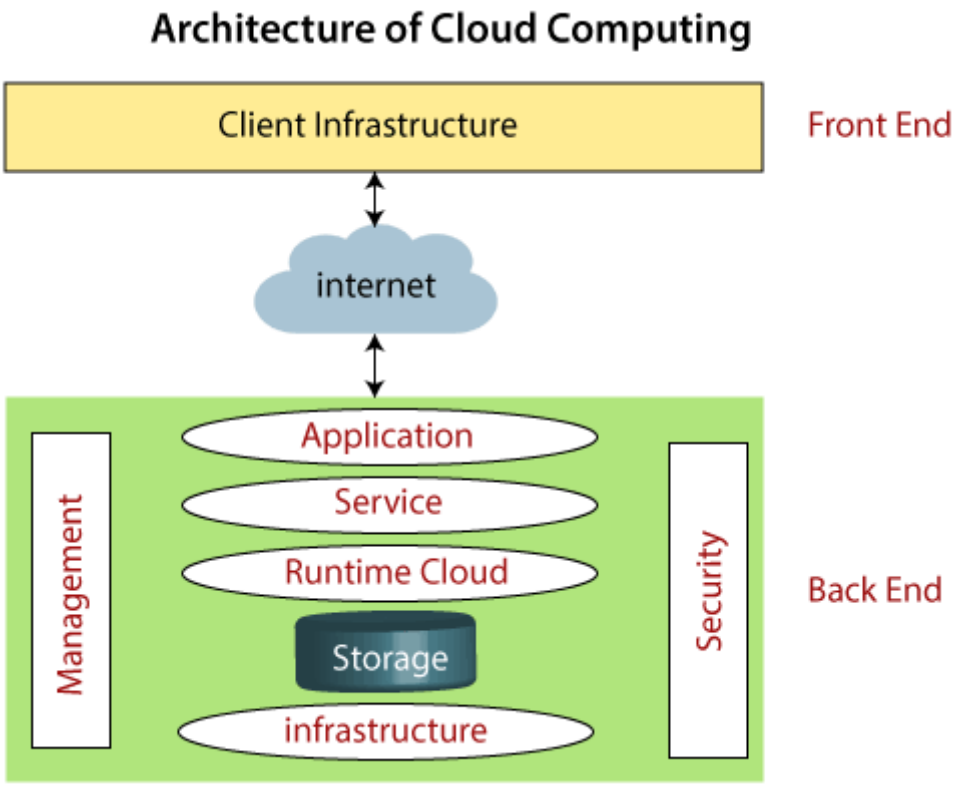

Рисунок 1 – Архитектура типовой облачной системы

К компонентам архитектуры облачных систем относятся:

1. Client Infrastructure (инфраструктура клиента) - это компонент переднего плана. Он предоставляет графический интерфейс пользователя (GUI) для взаимодействия с облаком.

2. Компонент Application (приложение) может быть любым программным обеспечением или платформа, к которой клиент хочет получить доступ.

3. Компонент Service (сервис) представляет собой облачную службу, которая управляет процессом доступа клиента к облачным сервисам.

Облачные вычисления предлагают следующие три типа сервисов:

– программное обеспечение как услуга (SaaS) - службы облачных приложений. В большинстве случаев приложения SaaS запускаются непосредственно через веб-браузер и поэтому не требуют установки приложений.

Примеры: Google Apps, Salesforce Dropbox, Slack, Hubspot, Cisco WebEx и др.

– платформа как услуга (PaaS), которая предоставляет клиенту платформу для создания программного обеспечения.

Пример: Windows Azure, Force.com, Magento Commerce Cloud, OpenShift и др.;

– инфраструктура как услуга (IaaS). Этот сервис отвечает за управление данными приложений, промежуточным программным обеспечением и средами выполнения.

Примеры: Amazon Web Services (AWS) EC2, Google Compute Engine (GCE), Cisco Metapod и др.

4. Компонент Runtime Cloud (облако времени выполнения). Данный компонент предоставляет виртуальным машинам среду выполнения.

5. Компонент Storage (хранение) - один из важнейших компонентов облачных вычислений. Он предоставляет огромный объем хранилища в облаке для хранения данных и управления ими.

6. Компонент Infrastructure (инфраструктура) предоставляет услуги на уровне хоста, уровне приложений и сети. Облачная инфраструктура включает в себя аппаратные и программные компоненты, такие как серверы, хранилище, сетевые устройства, программное обеспечение виртуализации и другие ресурсы хранения, которые необходимы для поддержки модели облачных вычислений.

7. Компонент Management (управление) используется для управления такими компонентами, как приложение, служба, среда выполнения, хранилище, инфраструктура и другими проблемами безопасности в серверной части, а также для установления координации между ними.

8. Компонент Security (безопасность) - это встроенный серверный компонент облачных вычислений. Он реализует механизм безопасности в серверной части.

9. Интернет - это среда, через которую клиентская часть и серверная

часть могут взаимодействовать и общаться друг с другом.

Для представления архитектуры облачной системы используются следующие уровни:

– концептуальная архитектура, которая определяет компоненты системы и их назначения, обычно в неформальном виде. Это представление часто используется для обсуждения с нетехническими специалистами, такими как руководство, бизнес-менеджеры и конечные пользователи функциональных характеристик системы (что система должна уметь с точки зрения конечного пользователя);

– логическая архитектура, которая описывает программные компоненты системы и взаимодействие, интерфейсы и используемые протоколы;

– физическая архитектура, которая описывает привязку к конкретным узлам размещения, типам оборудования и системного программного обеспечения.

На практике при разработке облачных системных архитектур используются различные варианты моделей архитектур, которые можно изменить и настроить в соответствии с требованиями конкретного проекта.

Рассмотрим особенности и проанализируем базовые концептуальные модели архитектур построения облачных систем [16].

1.2 Обзор и анализ моделей архитектур построения облачных систем

Приведенные ниже модели являются предоставлениями базовых архитектур, используемых при построении облачных систем.

1) Модель архитектуры «Все в одном» с единым сервером (Single "All-in-one" Server).

В данном решении (рисунок 2) используется один из универсальных серверных шаблонов (Server Templates), например, LAMP (Linux, Apache, MySQL, PHP), для запуска одного сервера, содержащего веб-сервер (Apache),

а также приложение (PHP) и СУБД (MySQL) [19].

Рисунок 2 – Модель архитектуры с единым сервером

2) Модель архитектуры сайта в едином облаке (Single Cloud Site Architectures).

В стандартной трехуровневой архитектуре веб-приложения есть как минимум один выделенный сервер на каждом уровне архитектуры системы - сервер балансировки нагрузки, сервер приложений, сервер базы данных (БД).

Рассмотрим варианты данной модели:

– трехуровневая архитектура без резервирования.

Если тестируется только интерактивность между каждым уровнем архитектуры, то можно использовать нерезервированную системную архитектуру, чтобы сэкономить на затратах и ресурсах. Такая модель в основном используется для базовых целей тестирования и разработки.

На представленной на рисунке 3 модели серверы выделены для каждого уровня веб-приложения.

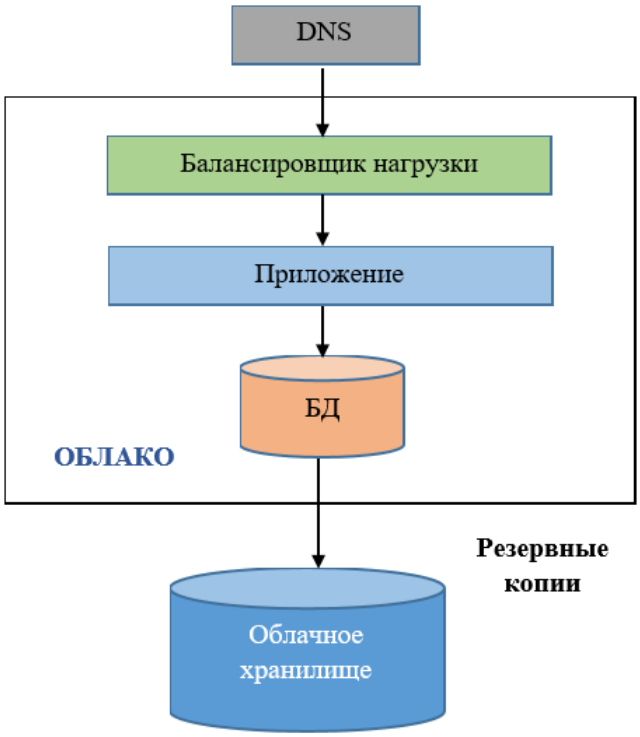

Рисунок 3 – Модель трехуровневой архитектуры без резервирования

Трехуровневую архитектуру без резервирования не рекомендуется использовать для производственных сред;

– трехуровневая архитектура с резервированием.

Любая производственная среда, запускаемая в облаке, также должна иметь избыточную архитектуру для аварийного переключения и восстановления. Как правило, используется серверный массив для уровня приложения, чтобы воспользоваться преимуществами автомасштабирования в облаке, однако могут быть некоторые сценарии, в которых приложение не предназначено для автомасштабирования.

В таких случаях все равно можно создать многоуровневую архитектуру, в которой избыточность будет присутствовать на каждом ее уровне.

В приведенном на рисунке 4 примере есть два сервера балансировки нагрузки, два сервера приложений, а также главный и подчиненный серверы баз данных. Архитектура с резервированием поможет защитить веб-приложение от простоя системы.

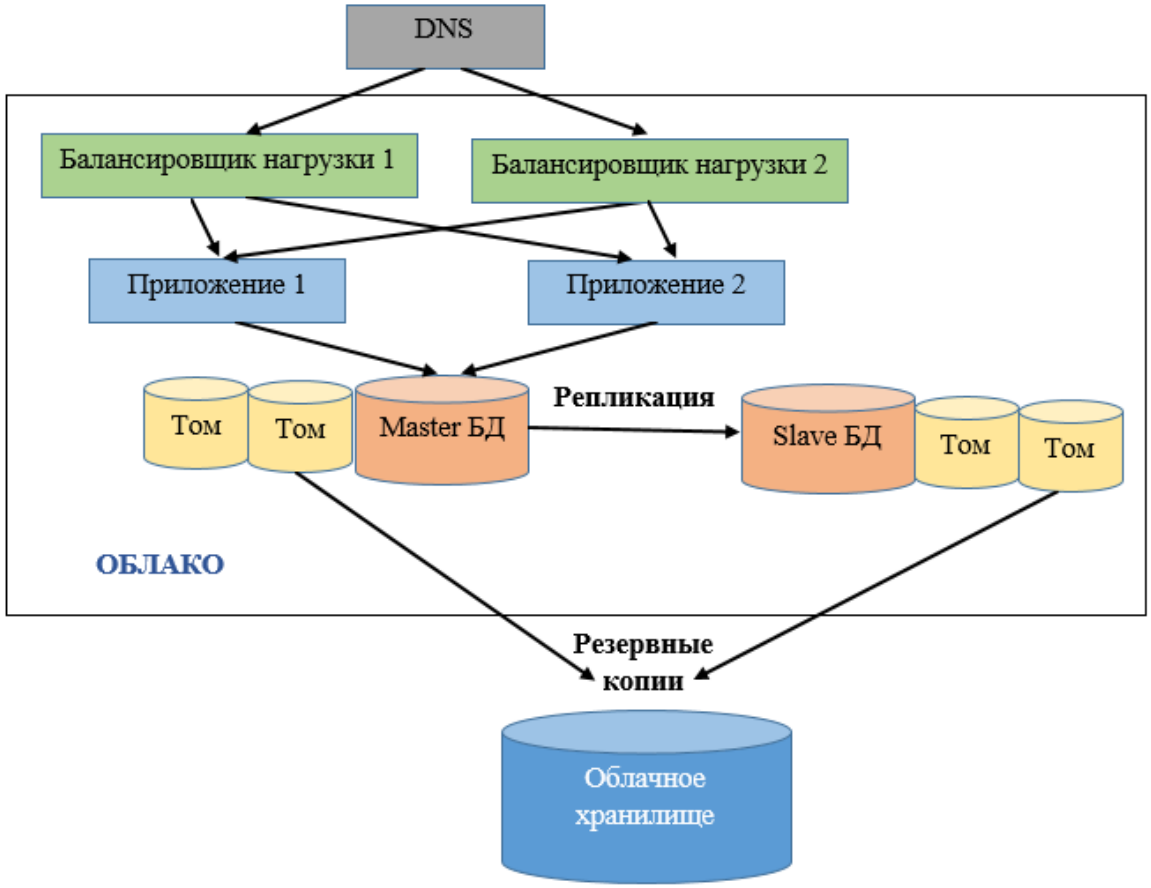

Рисунок 4 – Модель трехуровневой архитектуры с резервированием

Этот пример также демонстрирует использование чередующегося тома, установленного на уровне БД.

Если БД большая и требует более быстрого резервного копирования, можно рассмотреть возможность использования набора чередующихся томов для хранения данных;

– архитектура с несколькими центрами обработки данных.

Если облачная инфраструктура поддерживает несколько центров обработки данных (или зон), рекомендуется распределить архитектуру системы между несколькими центрами обработки данных, чтобы добавить еще один уровень избыточности и защиты (рисунок 5).

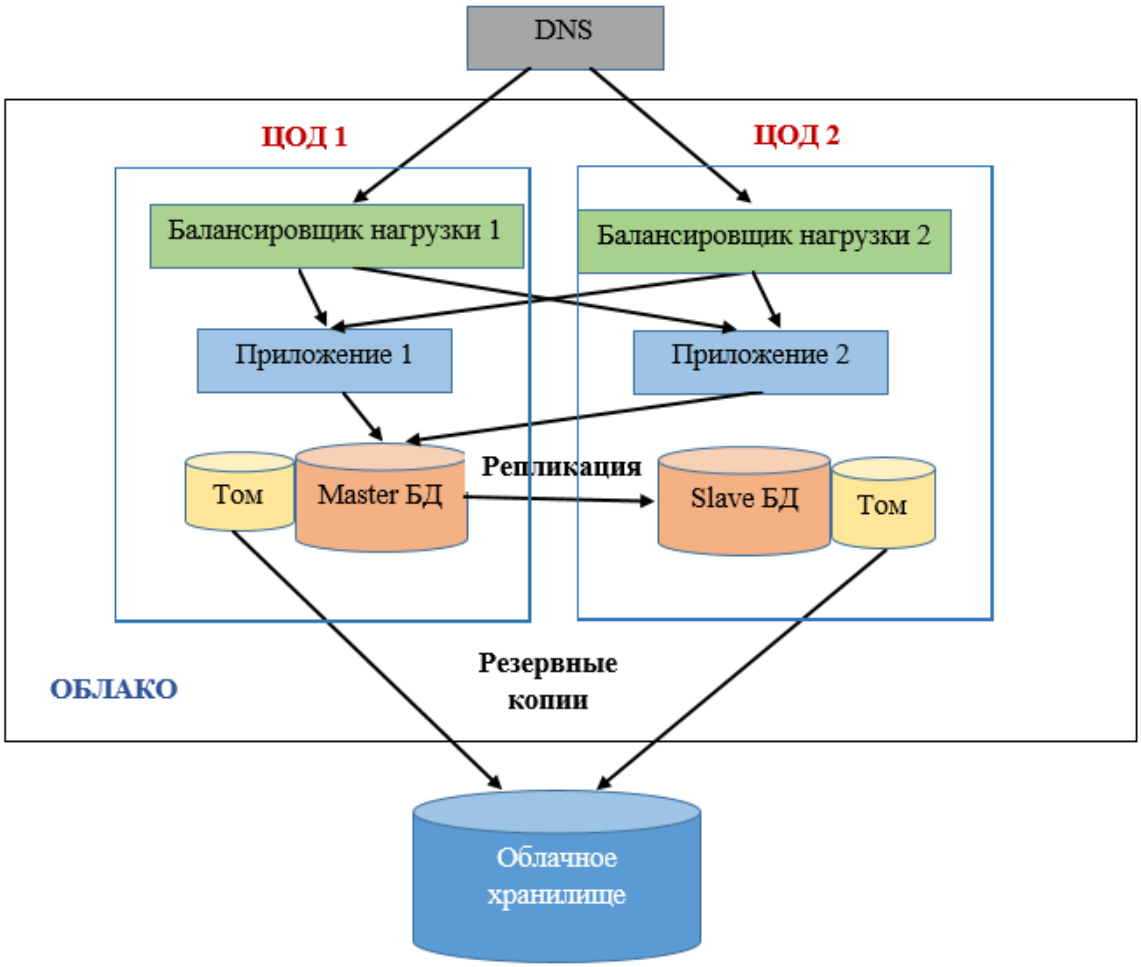

Рисунок 5 – Модель архитектуры с несколькими центрами обработки данных

Каждый центр обработки данных (ЦОД) в облаке спроектирован как изолированный сегмент внутри одного и того же географического облака.

Таким образом, если в одном ЦОД произойдет сбой питания, это не повлияет на другие центры обработки данных.

Например, в облаке может быть несколько пулов ресурсов, называемых зонами доступности и центрами обработки данных.

Преимущество использования нескольких центров обработки данных заключается в защите всего приложения (сайта) от негативного воздействия какого-либо типа сбоя сети (питания), отсутствия доступных ресурсов или сбоев в обслуживании, характерных для конкретного ЦОД.

Рекомендуется всегда использовать несколько центров обработки

данных в эталонной архитектуре, если они поддерживаются облачной инфраструктурой.

На схемах эталонной архитектуры, представленных ниже, также рекомендуется использовать несколько центров обработки данных;

– архитектура с автомасштабированием (рисунок 6).

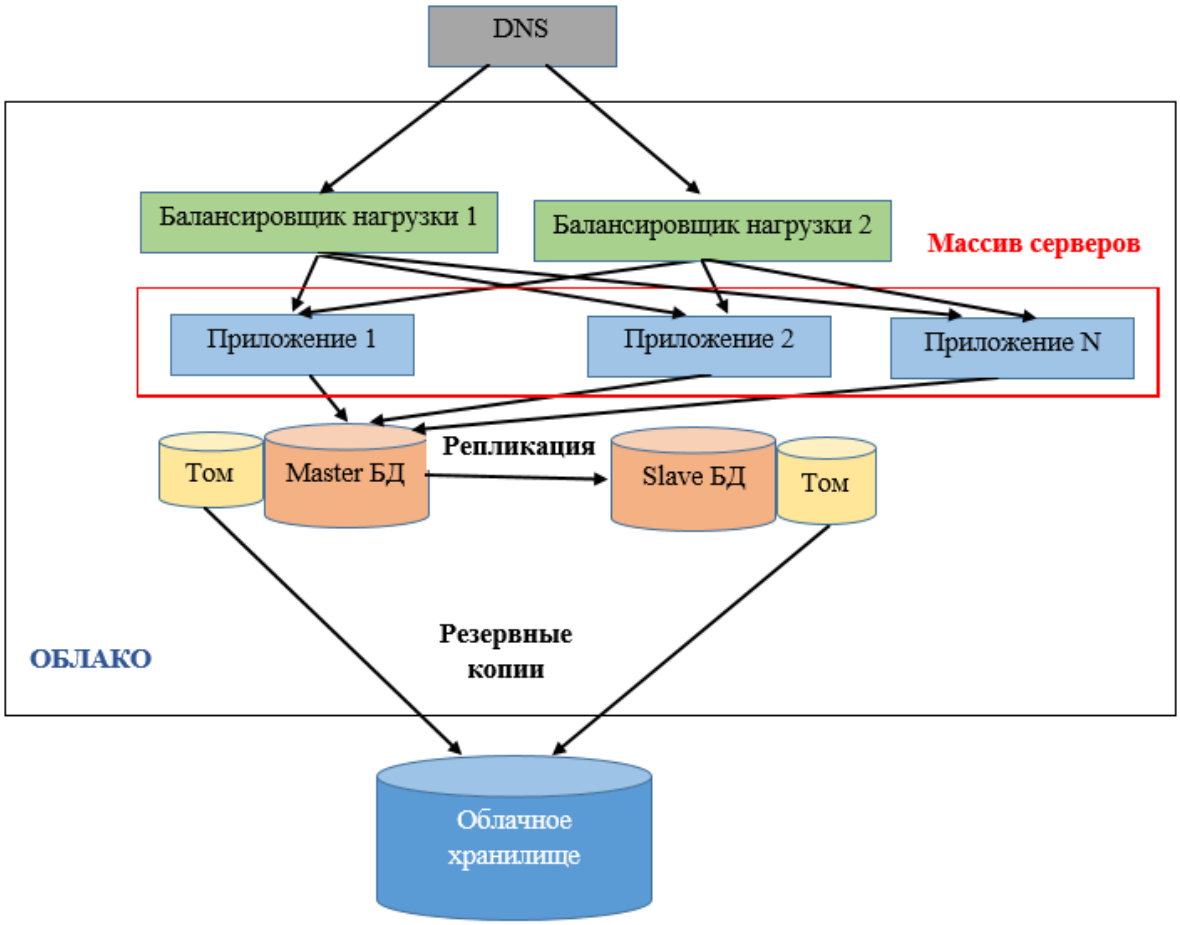

Рисунок 6 – Модель архитектуры с автомасштабированием

Одним из ключевых преимуществ облака является возможность горизонтального масштабирования, т. е. увеличения или уменьшения количества работающих ресурсов сервера по мере того, как меняются требования к приложению (сайта).

Архитектура с автомасштабированием чаще всего используется для уровня приложений облачной системы.

– масштабируемая архитектура с СУБД NoSQL.

Данная архитектура позволяет использовать узлы облачной СУБД NoSQL, например, Couchbase для своего уровня базы данных (рисунок 7) [17].

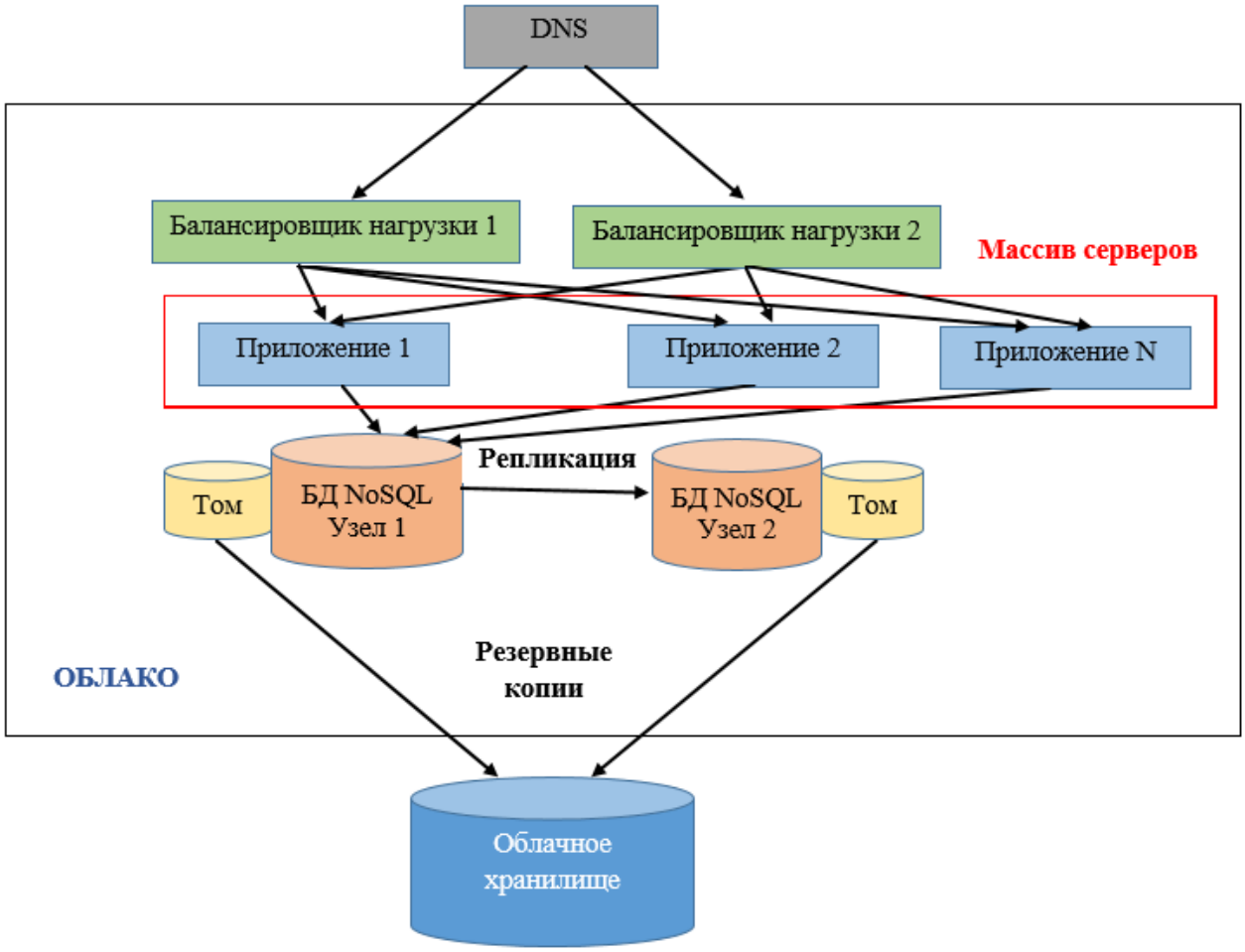

Рисунок 7 – Модель архитектуры с СУБД NoSQL

Решение представляет собой распределенную БД, которая реплицирует данные на все узлы.

3) Модель архитектуры гибридного облачного сайта.

Еще один способ защитить свой сайт или приложение в облаке - это разработать архитектуру гибридного облачного сайта, которая использует несколько общедоступных или частных облачных инфраструктур, а также выделенные серверы.

Рассмотрим варианты данной модели:

– масштабируемая мультиоблачная архитектура.

В приведенном ниже примере (рисунок 8) облачная инфраструктура X используется для размещения сайта или приложения. Также создан массив серверов для автомасштабирования уровня приложения в другой облачной инфраструктуре Y.

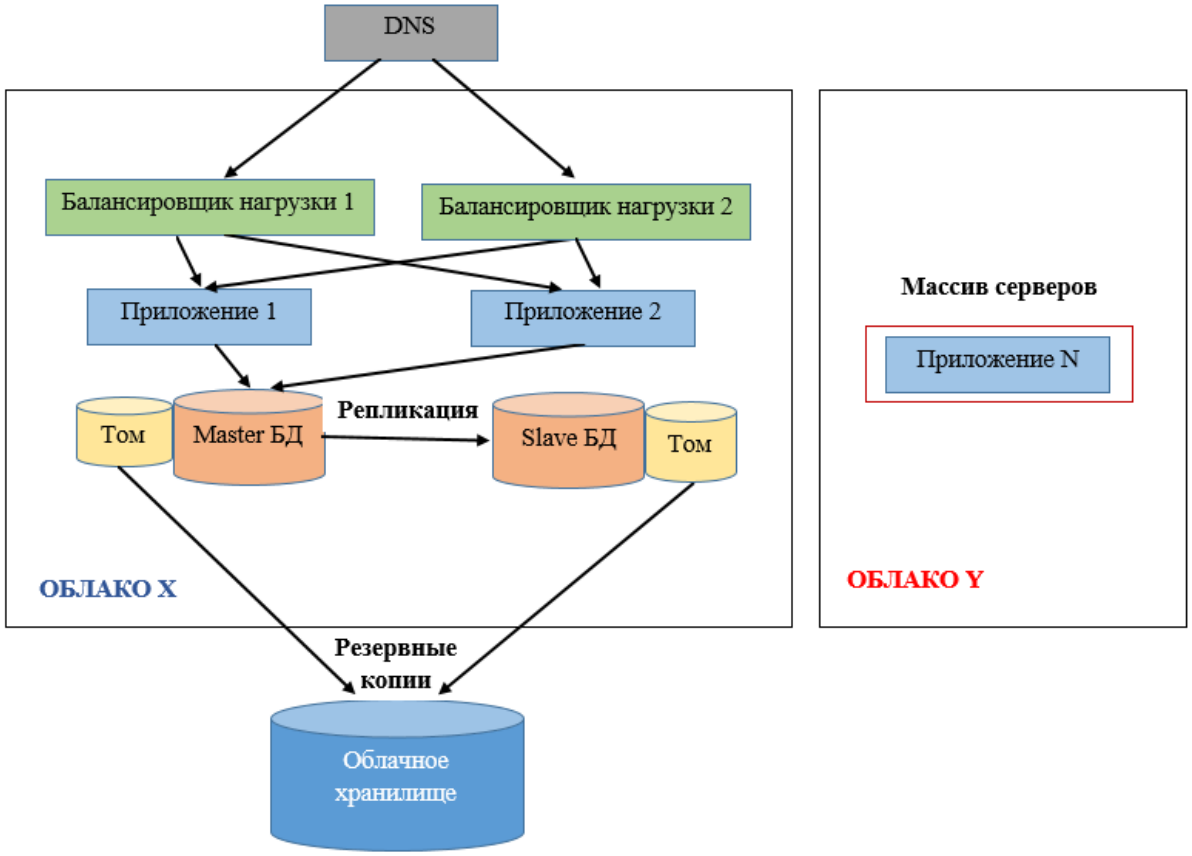

Рисунок 8 – Модель масштабируемой мультиоблачной архитектуры

Представленное решение архитектуры дает возможность пользователям размещать приложение в собственной облачной инфраструктуре, а также при необходимости автоматически масштабироваться в общедоступное облако для увеличения емкости сервера.

– отказоустойчивая мультиоблачная архитектура;

В приведенном ниже примере (рисунок 9) одни и те же шаблоны Server Templates и сценарии используются для настройки и запуска функциональных

серверов в облаке X или Y.

При проектировании мультиоблачной архитектуры облачной необходимо учитывать несколько факторов. Работающий сервер Slave-DB, который служит горячей резервной копией, но он реплицирует данные с Master-DB через общедоступный, а не частный IP-адрес.

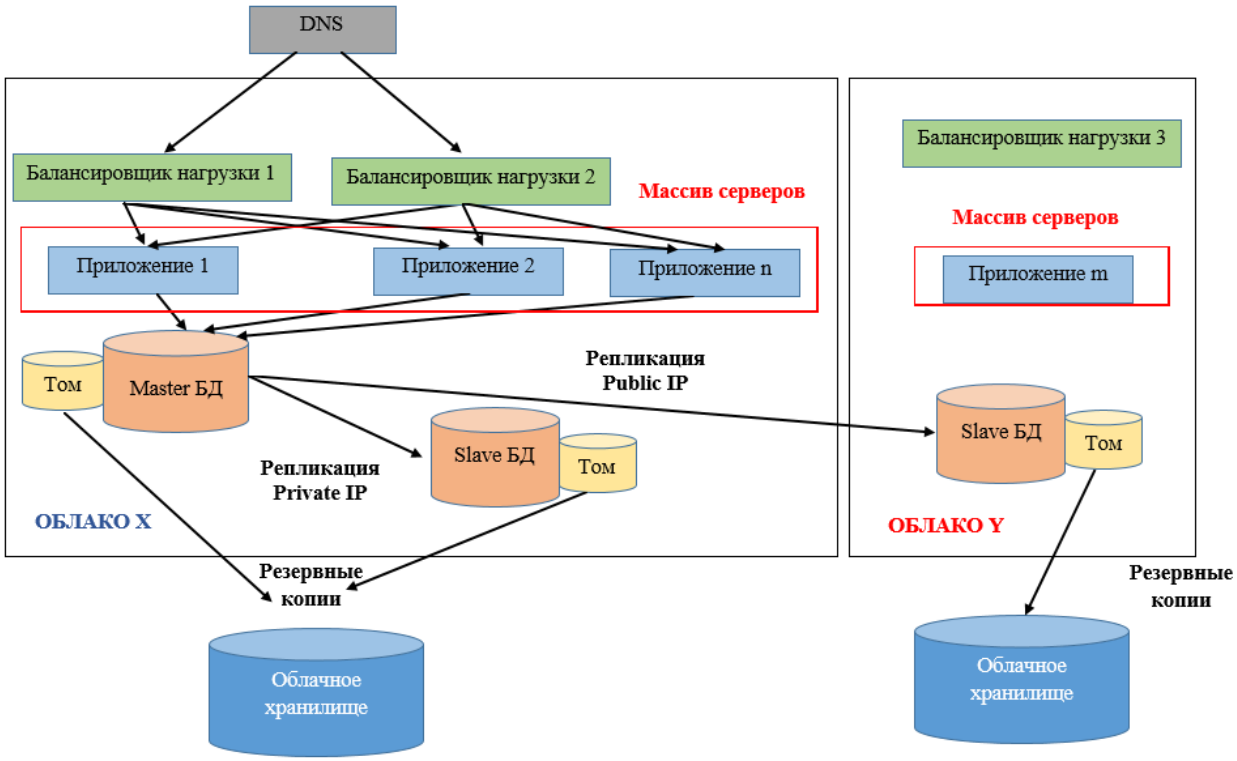

Рисунок 9 – Модель отказоустойчивой мультиоблачной архитектуры

Необходимо иметь в виду, что только серверы в одной облачной инфраструктуре могут общаться через частный IP-адрес. Однако, если когда-либо возникнет проблема или сбой, который потребует переключения облаков, мультиоблачная архитектура позволит пользователю легко перенести сайт или приложение.

Для оценки рассмотренных архитектур используем следующие характеристики:

– стоимость - необходимо разработать модели ценообразования, связанные с проектируемой облачной системой, и оценить затраты на ее

реализацию;

– масштабируемость – способность архитектуры расширяться в случае необходимости;

– простота - упрощенные архитектуры всегда будет легче проектировать и управлять ими. Более сложное решение следует использовать только в том случае, если более простой версии недостаточно;

– скорость - для приложений, интенсивно использующих пользовательские ресурсы, дополнительная задержка, возникающая в результате обмена данными между облаками, может быть неприемлема;

– переносимость - возможность в случае необходимости переноса конкретного уровня архитектуры к другому поставщику облачных услуг;

– безопасность – обеспечение необходимо уровня защиты данных клиентов.

Для сравнения представленных моделей и методов используем таблицу 1, составленную на основе анализа блогов по данной тематике.

Таблица 1 – Сравнительный анализ архитектур построения облачных систем

Характеристика/балл	Единый сервер	Трехуровневая архитектура с резервированием	Гибридный облачный сайт
Стоимость	3	2	1
масштабируемость	0	2	3
простота	3	2	1
скорость	3	2	1
переносимость	2	2	2
Безопасность	1	3	1
Итого	12	13	9

Критерии оценивания:

0 – полное несоответствие требованиям;

1 – значительное несоответствие требованиям;

2 – незначительное несоответствие требованиям;

3 – полное соответствие требованиям.

Таким образом, наилучшими характеристиками обладает трехуровневая архитектура с резервированием. Однако данная архитектура не обеспечивает необходимый уровень масштабирования облачной системы.

Выводы к главе 1

Первая глава посвящена постановке задачи на исследование, обзору и анализу существующих архитектур построения облачных систем.

Результаты проделанной работы позволили сделать следующие выводы:

1. В настоящее время в российской и зарубежной практике проектирования ИС термин «архитектура системы» используется очень широко, но при этом имеет столь же широкое множество различных трактовок.

2. Для описания архитектуры облачной системы используются концептуальный, логический и физический уровни представления.

3. Архитектура типовой облачной системы состоит из фронт-энда и бэк-энда.

4. Сравнительный анализ архитектур построения облачных систем показал, что наилучшими характеристиками обладает трехуровневая архитектура с резервированием. Однако данная архитектура не обеспечивает необходимый уровень масштабирования облачной системы.

Глава 2 Выбор метода и проектирование логической архитектуры для построения облачной системы

В настоящее время для проектирования архитектуры приложений используются два подхода:

– разработка архитектуры на основе интеграции приложений (концепция Enterprise Application Integration – EAI);

– разработка сервис-ориентированной архитектуры (Service Oriented Architecture – SOA).

Рассмотрим данные подходы.

2.1 Архитектура на основе интеграции приложений

Интеграция корпоративных приложений (EAI) - это задача объединения баз данных и рабочих процессов, связанных с бизнес-приложениями для гарантии того, что бизнес использует информацию согласованно и что изменения в основных бизнес-данных, внесенные одним приложением, правильно отражаются в других (рисунок 10).

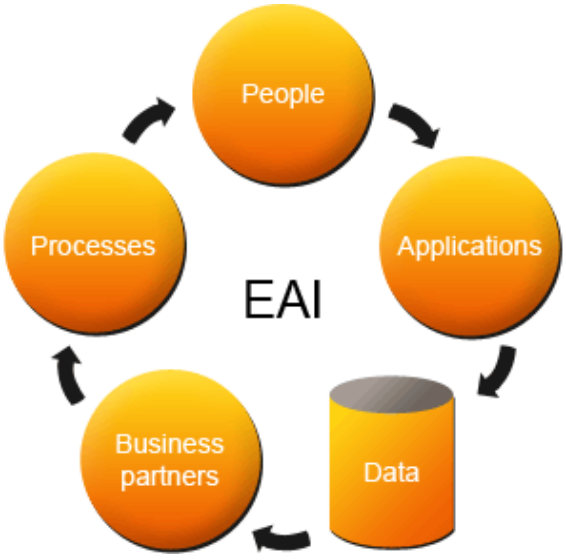

Рисунок 10 – Обобщенная модель архитектуры КИС на основе EAI

EAI является технологией, при помощи которой организация достигает централизации и оптимизации интеграции корпоративных приложений, используя, как правило, подходящие формы технологии оперативной доставки информации (push technology), которая управляется внешними событиями (event-driven) [18].

Интеграция корпоративных приложений дает множество преимуществ, таких как:

– процессы нескольких приложений можно автоматизировать, зарезервировав ИТ и другие ресурсы предприятия для стратегических действий, а не для рутинных задач обслуживания;

– интеграция программного обеспечения позволяет предприятиям быстро создавать ИТ-инфраструктуру и своевременно реагировать на растущие потребности клиентов.

EAI помогает уменьшить коммуникационные препятствия и оптимизирует бизнес-процессы, предоставляя доступ к информации из различных приложений в простой в использовании интерфейс.

Недостатком этого подхода является сложность. Реальные EAI-проекты, как правило, оказываются дорогостоящими и длительными и требуют значительных экспертных знаний.

Разработка архитектуры на основе концепции EAI в настоящее время больше применимо при построении системы на основе готовых существующих приложений.

2.2 Сервис-ориентированная архитектура

SOA, или сервис-ориентированная архитектура, предлагает способ сделать программные компоненты повторно используемыми и совместимыми через сервисные интерфейсы. Службы используют общие стандарты интерфейса и архитектурный шаблон, поэтому их можно быстро включить в новые приложения [2].

«С точки зрения пользователя, программный продукт представляет собой набор простых в использовании веб-сервисов с удобным графическим веб-интерфейсом. Сервис-ориентированная модель должна быть расширяемой: пользователь должен иметь возможность добавить новые сервисы или изменить набор доступных сервисов.

Пользователи также должны иметь возможность обращаться к сервисам через сеть с самых различных по своим возможностям устройств – desktop-машин, мобильных устройств и т.д.

Метод реализации веб-сервисов (.NET, Java и др.) для пользователя несущественен. Разработчик должен иметь возможность публикации своих веб-сервисов. Поддержка SOA осуществляется во многих современных программных инструментах, в том числе: Microsoft SharePoint - простой инструмент для создания расширяемых Web-страниц и Web-сервисов); UDDI (Universal Discovery, Description and Integration) – технология для публикации и поиска веб-сервисов (Microsoft).

С данной точки зрения, облачные вычисления соответствуют принципам SOA [7]. Типовая схема архитектуры SOA представлен на рисунке 11 [4].

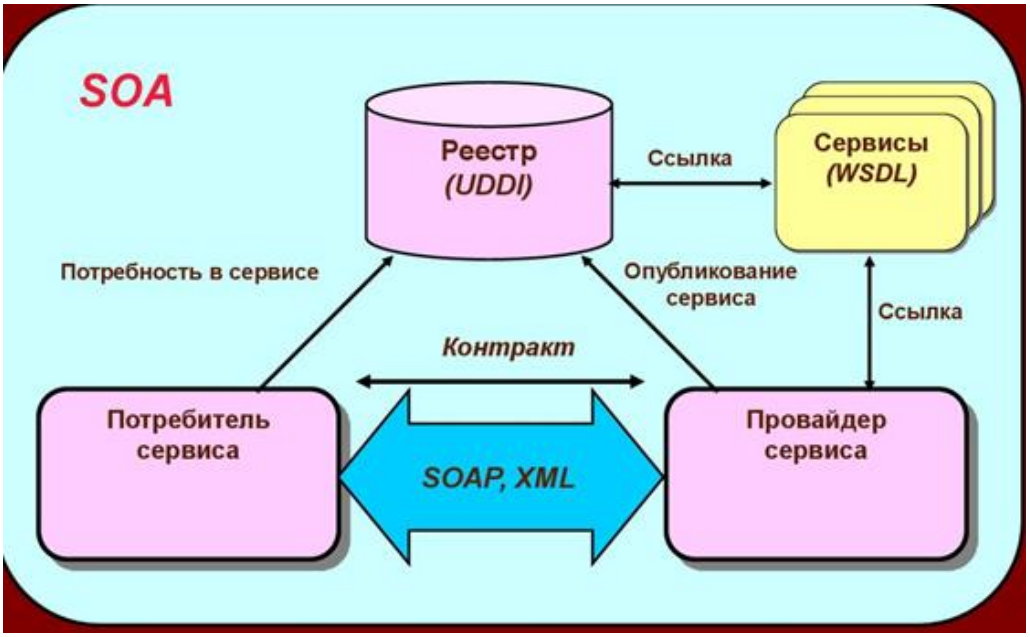

Рисунок 11- Типовая схема архитектуры SOA

На рисунке 12 показаны облачные вычисления SOA вместе с моделями услуг [22].

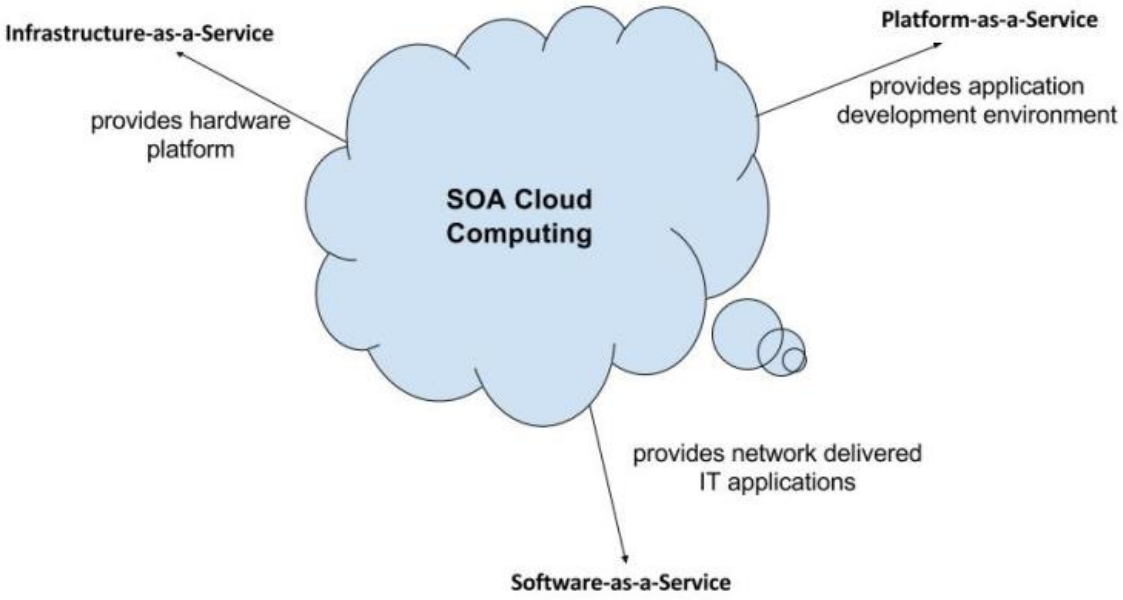

Рисунок 12- Схема облачных вычислений SOA

Основная цель SOA в облаке - лучше согласовать бизнес пользователей с информационными технологиями и другими ресурсами.

Это улучшает рабочие процессы, обеспечивая [21]:

- услуги управления доступом;
- простоту мониторинга и управления;
- простоту обмена данными;
- совместимость;
- платформенно-нейтральный подход;
- надежность;
- многоразовые коды и услуги;
- простая системная интеграция и т.д.

Сравним представленные подходы на предмет разработки качественной архитектуры построения облачной системы. Для этого используем таблицу 2.

Таблица 2 – Сравнительный анализ подходов к разработке архитектуры облачной системы

Характеристика/балл	EAI	SOA
Стоимость	1	2
Масштабируемость	3	3
простота реализации	1	3
Итого	5	8

Таким образом, наилучшими характеристиками обладает подход, основанный на сервис-ориентированной архитектуре.

2.3 Разработка логической модели архитектуры построения облачной системы на основе технологии SOA

В качестве методологии проектирования логического уровня архитектуры для построения облачной системы выбираем методологию RUP. Rational Unified Process (RUP) — методология разработки программного обеспечения, созданная компанией Rational Software [5]. В методологии RUP для успешного процесса разработки ПО необходимы три составляющие: процесс, нотация и набор утилит. Процесс описывает, что мы делаем, в каком порядке и каким образом; нотация является средством общения; набор утилит помогает автоматизировать процесс и управлять им.

Основная цель RUP - создание высококачественного ПО с предсказуемым бюджетом и временными рамками. При необходимости каждую из фаз жизненного цикла можно повторять до тех пор, пока не будут достигнуты основные цели. В концепции RUP архитектура – это фундаментальная организация ИС, заключенная в своих компонентах, в их взаимоотношениях, в окружении, а также принципы, определяющие проектирование, создание и развитие ИС.

В методологии RUP архитектура программной системы представляется

множеством из следующих пяти архитектурных точек зрения, которые соответствуют основным элементам в соответствующих моделях: The use case view – структура вариантов использования ИС; The Logical View – логическое представление архитектуры; The Implementation View – структура программной реализации системы; The Process View – структура объединения подсистем и процессов; The Deployment View – структура физического распределения компонентов ИС по аппаратным средствам.

В качестве примера рассмотрим облачную систему предоставления SaaS-услуги [2, 4]. В качестве языка моделирования в RUP используется UML [20].

2.3.1 Разработка диаграммы вариантов использования архитектуры облачной системы

Для построения структуры вариантов использования архитектуры облачной системы используем соответствующую диаграмму UML. Диаграммы вариантов использования моделируют взаимодействия между задачами (прецедентами), которые выполняет система, и внешними объектами-акторами, которым необходимо выполнение этих задач [6].

Варианты использования представлены в таблицах 3, 4.

Таблица 3 - Описание прецедента: Доступ к веб-сервису

Прецедент: Доступ к веб-сервису
ID: 1
Краткое описание: обеспечение доступа к веб-сервису
Главный актор: Протоколы SOAP или REST
Второстепенные акторы: нет
Предусловие: нет
Основной поток: Протоколы SOAP или REST обеспечивают доступ к веб-сервису
Альтернативные потоки: нет

Таблица 4 - Описание прецедента: Доступ к SaaS-услуге

Прецедент: Доступ к SaaS-услуге
ID: 2
Краткое описание: обеспечение доступа к SaaS-услуге
Главный актор: Интерфейс пользователя
Второстепенные акторы: протоколы SOAP или REST
Предусловие: обеспечение доступа к веб-сервису
Основной поток: Интерфейс пользователя обеспечивает последнему доступ к SaaS-услуге
Альтернативные потоки: нет

На рисунке 13 изображена диаграмма вариантов использования архитектуры облачной системы.

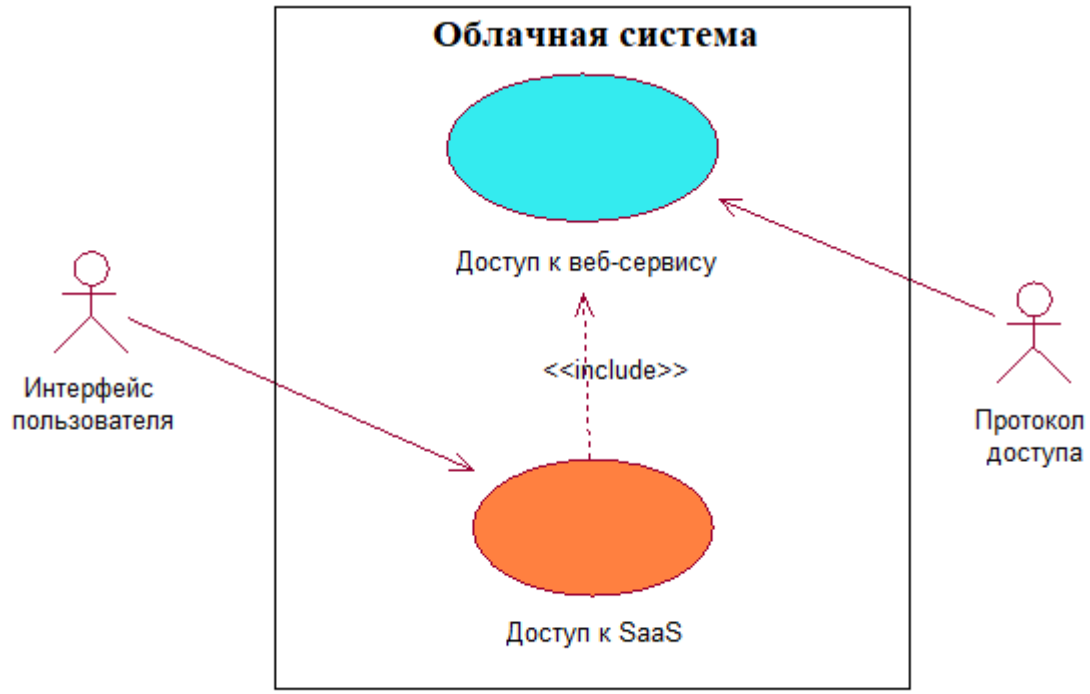

Рисунок 13 - Диаграмма вариантов использования архитектуры облачной системы

Данная диаграмма отражает концептуальный аспект архитектуры облачной системы.

2.3.2 Разработка диаграммы классов архитектуры облачной системы

Для построения логического представления архитектуры облачной системы используем диаграмму классов UML [8].

Диаграмма классов описывает атрибуты и операции класса, а также ограничения, налагаемые на систему. Диаграммы классов широко используются при моделировании объектно-ориентированных систем, поскольку они являются единственными UML-диаграммами, которые можно отображать напрямую с помощью объектно-ориентированных языков.

Диаграмма классов показывает набор классов, интерфейсов, ассоциаций, взаимодействий и ограничений.

Диаграмма классов архитектуры облачной системы представлена на рисунке 14.

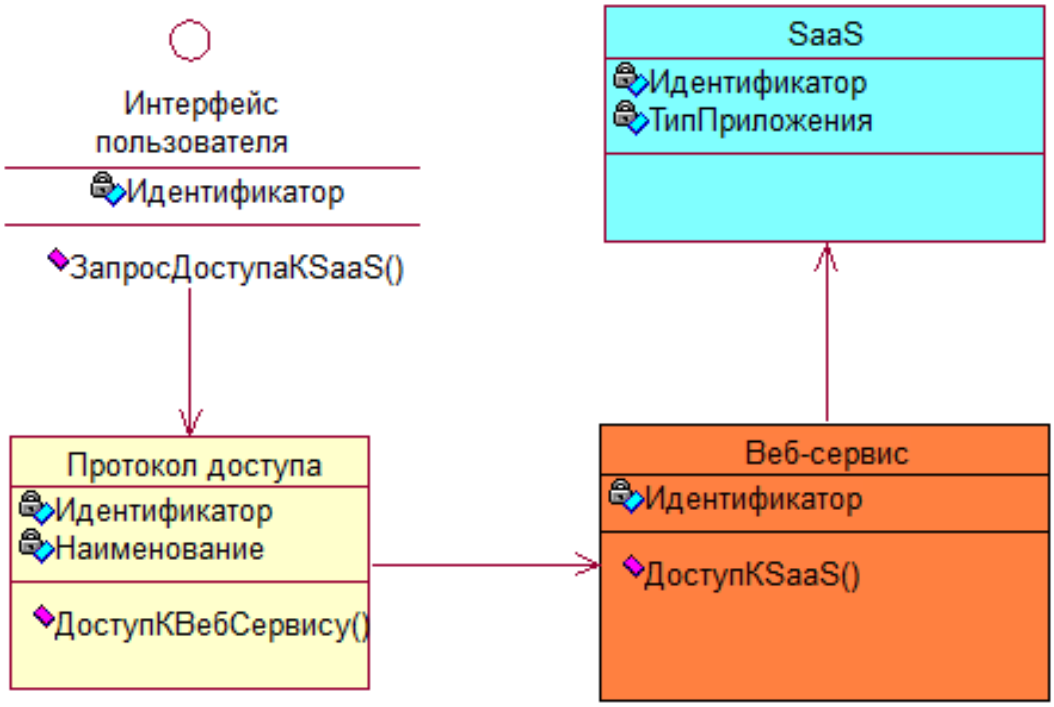

Рисунок 14 - Диаграмма классов архитектуры облачной системы

Спецификация классов архитектуры облачной системы приведена в таблице 5.

Таблица 5 – Спецификация классов архитектуры облачной системы

Класс	Описание
Интерфейс пользователя	Класс объектов, представляющих на логическом уровне интерфейсы пользователя
Протокол доступа	Класс объектов, представляющих на логическом уровне протоколы доступа к веб-сервисам
Веб-сервис	Класс объектов, представляющих на логическом уровне веб-сервисы
SaaS	Класс объектов, представляющих на логическом уровне облачные SaaS-услуги

Диаграмма классов отражает логический аспект архитектуры облачной системы.

2.3.3 Разработка диаграммы компонентов архитектуры облачной системы

Для построения представления программной архитектуры облачной системы используем диаграмму компонентов UML.

Диаграмма компонентов UML показывает компоненты программной архитектуры, интерфейсы, порты и отношения между ними.

Диаграммы компонентов UML предлагают разработчикам высокоуровневое архитектурное представление системы, которую они будут строить, а также сведения о логических компонентах ее программного обеспечения.

Диаграмма компонентов UML имеет более высокий уровень абстракции, чем диаграмма классов UML.

Диаграмма компонентов программной архитектуры облачной системы

представлена на рисунке 15.

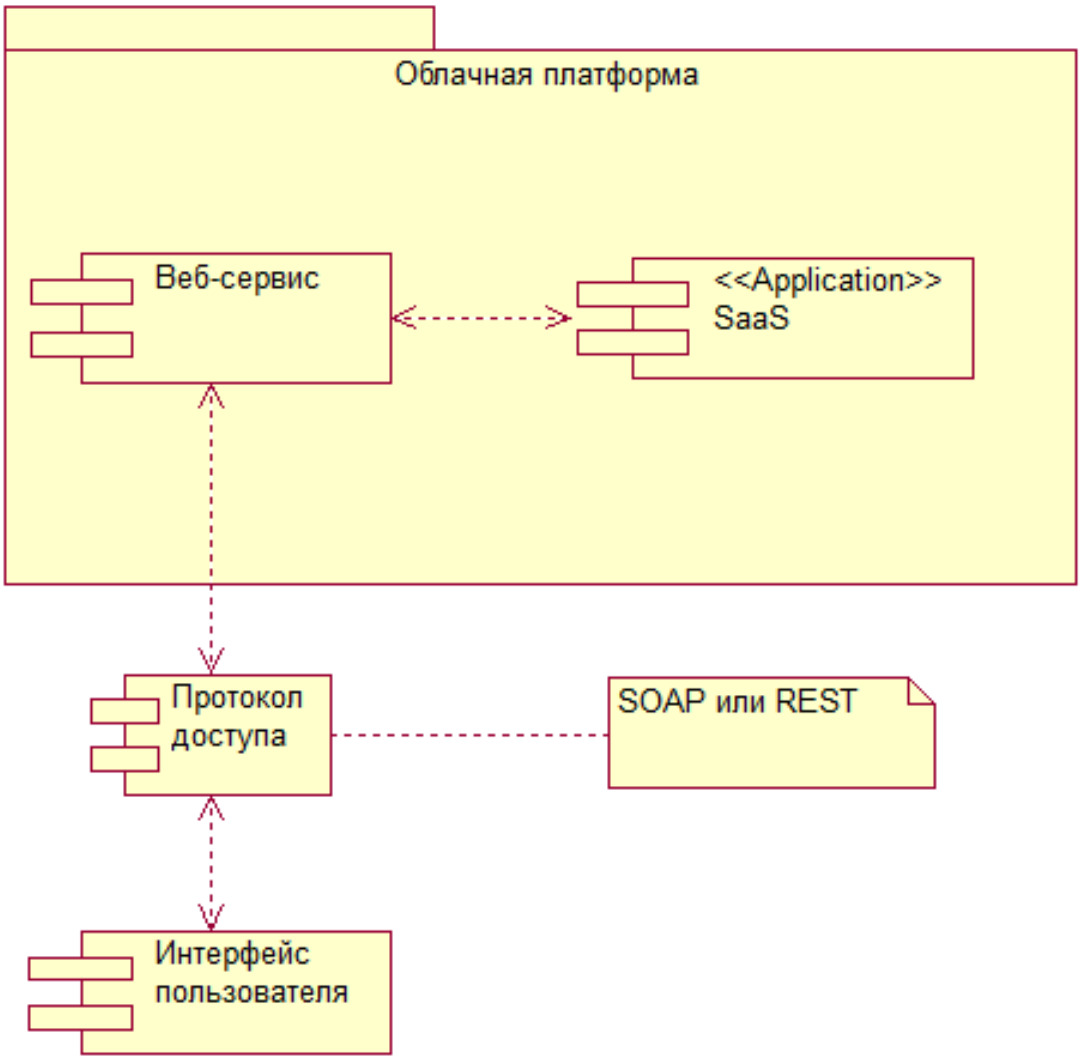

Рисунок 15 - Диаграмма компонентов программной архитектуры облачной системы

Диаграмма компонентов отражает аспект реализации программной архитектуры облачной системы.

2.3.4 Разработка диаграммы развертывания архитектуры облачной системы

Для построения представления физического распределения компонентов облачной системы по аппаратным средствам используем

диаграмму развертывания UML.

В UML диаграммы развертывания моделируют физическую архитектуру системы.

Диаграммы развертывания показывают физическое расположение узлов в распределенной системе, артефакты, которые хранятся на каждом узле, а также компоненты и другие элементы, которые реализуют артефакты.

Диаграмма развертывания архитектуры облачной системы представлена на рисунке 16.

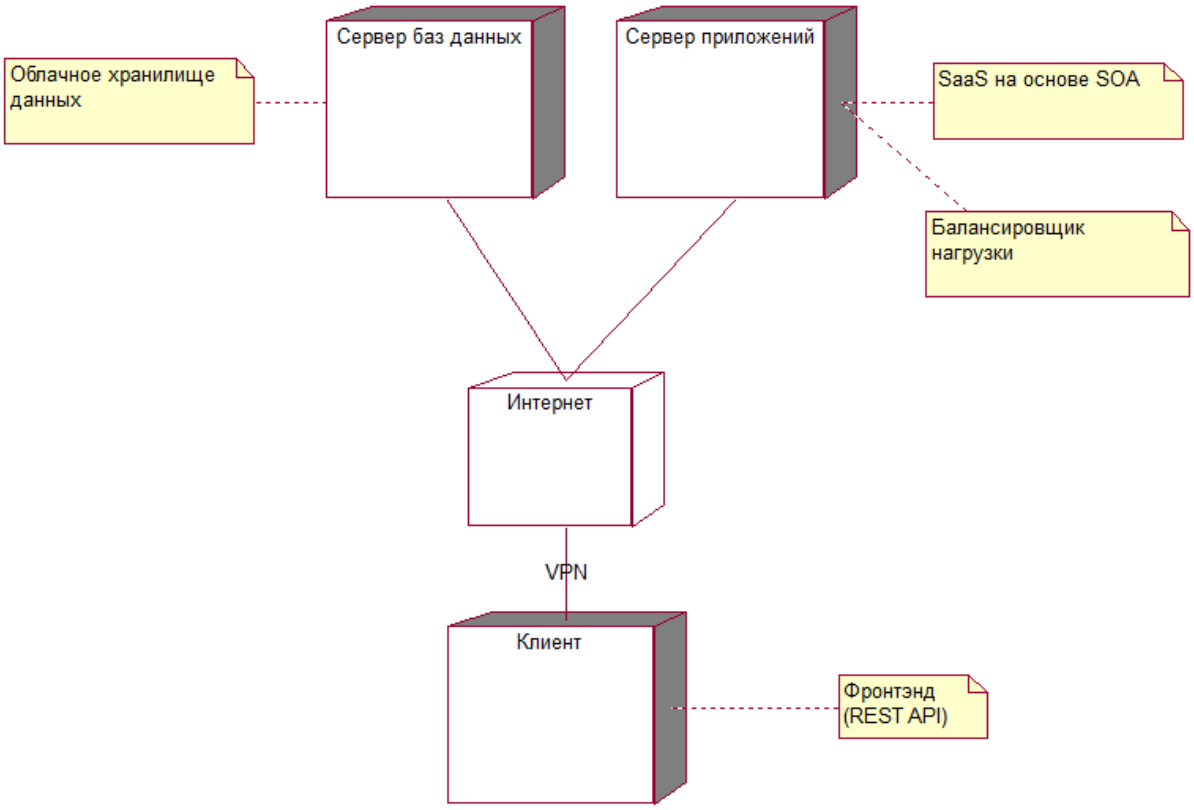

Рисунок 16 - Диаграмма развертывания архитектуры облачной системы

Диаграмма развертывания отражает физический аспект архитектуры облачной системы.

Разработанная в виде комплекса UML-диаграмм модель архитектуры является основой для выбора облачной платформы для построения облачной системы.

Выводы к главе 2

1. В настоящее время для разработки архитектуры приложений используются два подхода: разработка архитектуры на основе концепция EAI и сервис-ориентированной архитектуры – SOA. Как показал анализ, наилучшими характеристиками обладает подход, основанный на сервис-ориентированной архитектуре.

2. Для представления архитектуры облачной системы используем следующие уровни: концептуальный, логический и физический. В качестве методологии проектирования описанных уровней архитектуры выбрана методология RUP.

3. В методологии RUP архитектура программной системы представляется множеством из следующих пяти архитектурных точек зрения, которые соответствуют основным элементам в соответствующих моделях.

4. Разработанная в виде комплекса UML-диаграмм модель логической архитектуры является основой для построения физической архитектуры облачной системы.

Глава 3 Проектирование физической архитектуры для построения облачной системы на основе SOA

Для построения физической архитектуры построения облачной системы на основе логической ее модели используем эталонную архитектуру промышленного облачного решения.

Эталонная архитектура представляет собой документ или набор документов с рекомендуемыми структурами и процессами интеграции ИТ-продуктов и услуг для создания единого решения. В эталонной архитектуре отражен самый передовой опыт отрасли и обычно предлагаются оптимальные методы предоставления конкретных технологий.

Для анализа эталонных архитектур промышленных облачных платформ на предмет построения облачной системы на основе предлагаемой модели используем архитектурный подход.

«Архитектурный подход - соглашения, принципы и практики для описания архитектуры, установленные для конкретной области применения и/или конкретным сообществом стейкхолдеров.

В рамках данного подхода во главу угла ставится создание фреймворков, которые могут быть легко адаптированы ко всем потенциальным требованиям всех потенциальных заказчиком.

Отличительная особенность данного стиля состоит в том, что задача проектирования разбивается на две отдельные подзадачи: создание многократно используемого фреймворка и создание конкретного приложения на его основе.

При этом эти две задачи могут решать разные специалисты. Основная цель создания данного подхода — это устранение недостатков стиля, основанного на управлении требованиями.

Использование архитектурного подхода позволяет реализовать инкрементное и итеративное проектирование, т.е. оперативно изменять существующую и добавлять новую функциональность» [3].

В качестве фреймворка в рассматриваемом случае используем промышленную облачную платформу.

Рассмотрим эталонные архитектуры облачных платформ на основе SOA.

3.1 Базовая архитектура корпоративной интеграции в Microsoft Azure

Microsoft Azure - это служба облачных вычислений, созданная Microsoft для создания, тестирования, развертывания и управления приложениями и службами через центры обработки данных, управляемые Microsoft.

Базовые сценарии корпоративной интеграции в Microsoft Azure - эта эталонная архитектура, которую использует Azure Integration Services для оркестрации вызовов к корпоративным серверным системам.

Серверные системы могут включать в себя системы SaaS, службы Azure и существующие веб-службы предприятия [1].

Схема архитектуры представлена на рисунке 17.

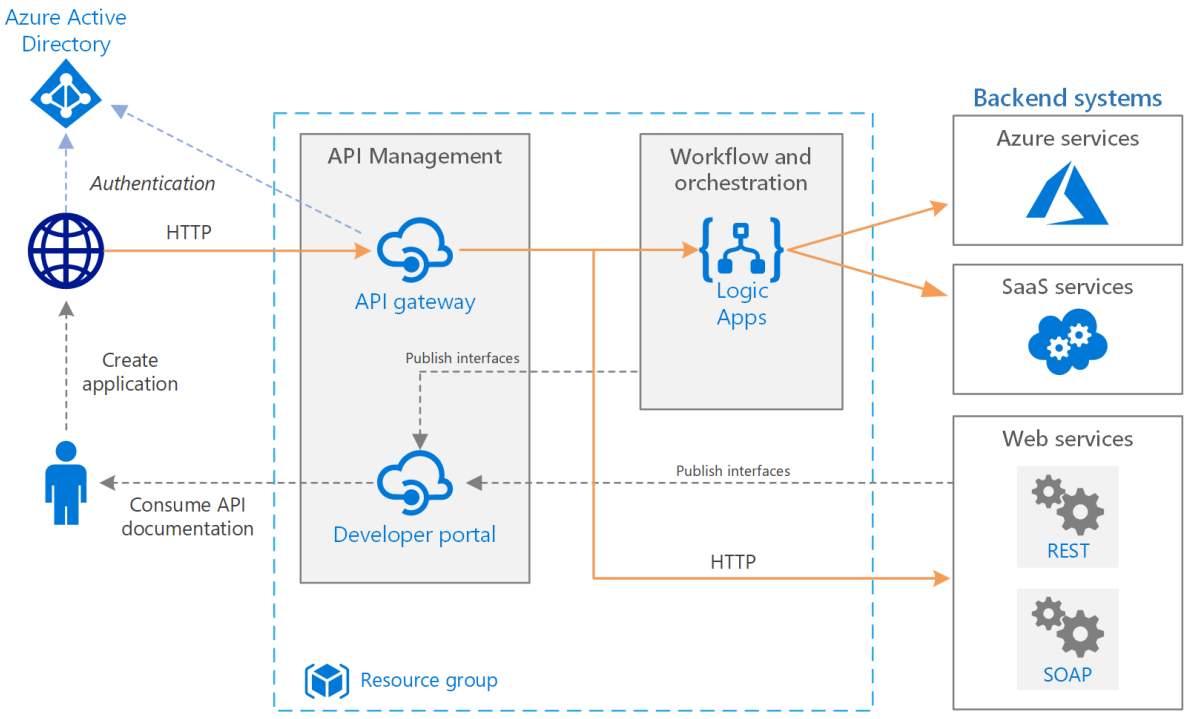

Рисунок 17 – Схема архитектуры корпоративной интеграции MS Azure

Эталонная архитектура состоит из следующих компонентов:

– серверные системы. Справа на схеме показаны различные внутренние системы, развернутые или используемые предприятием. Сюда могут входить системы SaaS, другие службы Azure или другие веб-службы, которые предоставляют конечные точки REST или SOAP;

– Azure Logic Apps. Logic Apps — это бессерверная платформа для создания рабочих процессов на предприятии, которая объединяет приложения, данные и службы. В этой архитектуре приложения логики активируются с помощью HTTP-запросов. Можно создавать вложенные рабочие процессы для более сложной оркестрации. Служба Logic Apps использует соединители для интеграции с часто используемыми службами. Logic Apps предлагает сотни соединителей. Также можно создавать собственные соединители;

– управление API Azure. Управление API — это управляемая служба, которая позволяет публиковать каталоги API HTTP, чтобы упростить их обнаружение и повторное использование. Служба управления API состоит из двух взаимосвязанных компонентов: шлюз API и портал разработчика;

– в этой архитектуре сложные API создаются путем импорта приложений логики в качестве API. Также можно импортировать существующие веб-службы путем импорта спецификаций OpenAPI (Swagger) или импорта API SOAP из спецификаций языка WSDL;

– шлюз API помогает отделить клиентские приложения от серверной части. Например, он может изменять URL-адреса или преобразовывать запросы перед их передачей в серверную часть. Шлюз также решает многие вопросы взаимодействия, например, проверку подлинности, предоставление общего доступа к ресурсам из разных источников (CORS) и кэширование ответов;

– Azure DNS — это служба размещения для доменов DNS. Azure DNS осуществляет разрешение имен на базе инфраструктуры Microsoft Azure. Размещая домены в Azure, можно управлять своими записями DNS с помощью

тех же учетных данных, интерфейсов API и инструментов и оплачивать использование, как и другие службы Azure. Чтобы использовать имя личного домена, например contoso.com, создайте записи DNS, которые позволяют сопоставить это доменное имя с IP-адресом;

– Azure Active Directory (Azure AD). Для проверки подлинности клиентов, выполняющих вызовы к шлюзу API, можно использовать службу Azure AD. Azure AD поддерживает протокол OpenID Connect (OIDC). Клиенты получают маркер доступа от Azure AD, а шлюз API проверяет маркер для авторизации запроса.

Azure Integration Services — это набор служб для интеграции приложений и данных. Описываемая архитектура использует две из этих служб — Logic Apps (для оркестрации рабочих процессов) и управление API (для создания каталогов API). Этой архитектуры достаточно для базовых сценариев интеграции, в которых рабочий процесс запускается с помощью синхронных вызовов к внутренним службам.

Более сложная архитектура с использованием очередей и событий основывается на этой базовой архитектуре.

3.2. Эталонная архитектура Oracle SOA Cloud Service Architecture

Oracle SOA Cloud Service Architecture поддерживает новые версии Oracle SOA Suite и его составляющих компонентов.

Oracle SOA Suite - это комплексный программный пакет с возможностью горячей замены, который позволяет создавать, развертывать и управлять интеграциями с использованием сервис-ориентированной архитектуры (SOA). Oracle SOA Suite предоставляет следующие возможности:

- единый инструментарий;
- единая модель развертывания и управления;
- сквозная безопасность;
- единое управление метаданными.

Oracle SOA Suite позволяет преобразовать сложные интеграции приложений в гибкие и многократно используемые приложения на основе сервисов, чтобы сократить время вывода на рынок, быстрее реагировать на бизнес-требования и снизить затраты.

Критически важные бизнес-услуги, такие как информация о клиентах, финансах, заказах и другие, которые ранее были доступны только в пользовательских интерфейсах пакетных приложений, можно быстро смоделировать для мобильных устройств, таких как смартфоны и планшеты, с помощью Oracle SOA Suite.

Oracle SOA Cloud предоставляет решение вычислительной платформы iPaaS для запуска приложений платформы интеграции в облаке.

Для предоставления доступны следующие типы услуг:

– Oracle Service Bus - это корпоративная сервисная шина на основе конфигурации и политик, которая предоставляет возможности для обнаружения и посредничества сервисов, быстрого предоставления и развертывания сервисов, а также управления.

– Oracle SOA Cloud обеспечивает бизнес-аналитику с помощью Oracle Business Activity Monitoring (BAM).

– Oracle Managed File Transfer Cloud Service обеспечивает безопасный обмен файлами между внутренними отделами и внешними партнерами.

Пользователю обеспечен полный и неограниченный административный доступ к прикладной среде в облаке.

На рисунке 18 показаны компоненты трехуровневой сетевой архитектуры, состоящей из общедоступного балансировщика нагрузки и экземпляров Oracle Java Cloud Service и Oracle Database Cloud Service в частной подсети [12].

Узел-бастион позволяет администраторам получить доступ к частным вычислительным узлам.

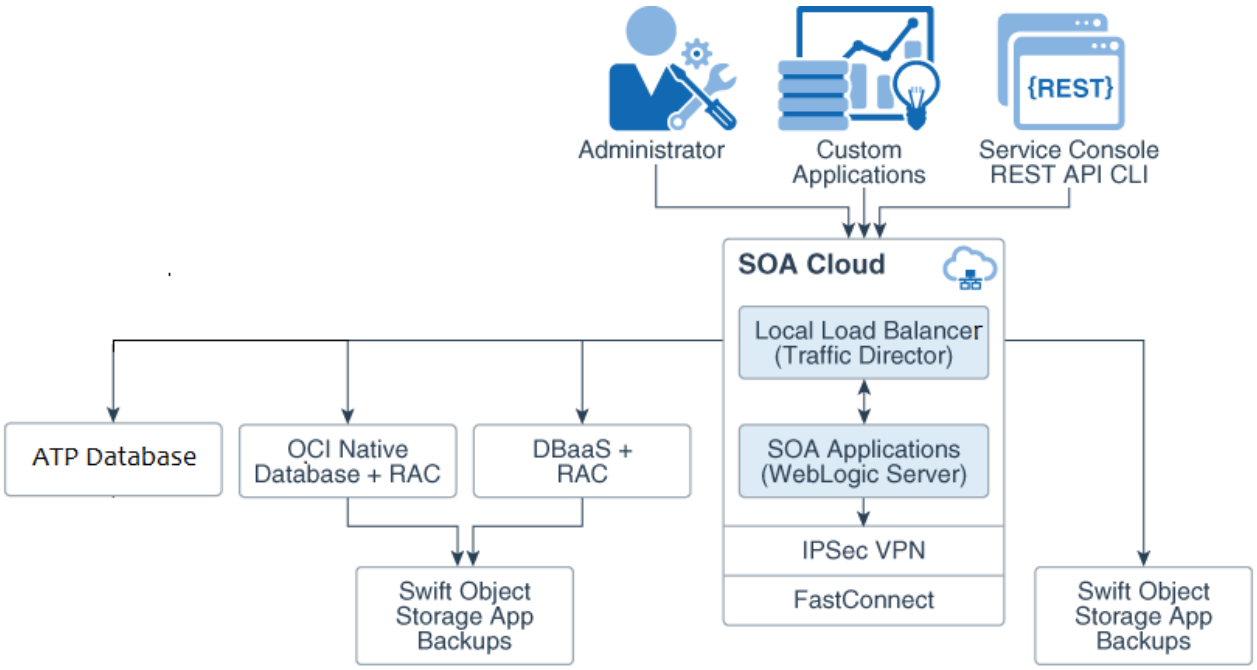

Рисунок 18 – Схема архитектура Oracle SOA Cloud Service Architecture

Сравним рассмотренные архитектуры с помощью таблицы 6.

Таблица 6 – Сравнительный анализ эталонных архитектур на основе SOA

Характеристика/балл	Архитектура корпоративной интеграции MS Azure	Oracle SOA Cloud Service Architecture
стоимость решения	2	2
гибкость	2	3
функциональные возможности	2	3
Итого	6	8

Таким образом, более соответствует требованиям архитектура Oracle SOA Cloud Service Architecture.

Ключевым компонентом решения является Oracle WebLogic Server — единая, расширяемая платформа для разработки, развертывания и запуска корпоративных приложений, таких как Java, в локальной и облачной среде.

WebLogic Server обладает мощным, высокотехнологичным, масштабируемым набором функций для полноценного внедрения Java Enterprise Edition (EE) и Jakarta EE [9].

На рисунке 19 показан алгоритм создания WebLogic-сервиса, который будет запускать WebLogic AdminServer [10].

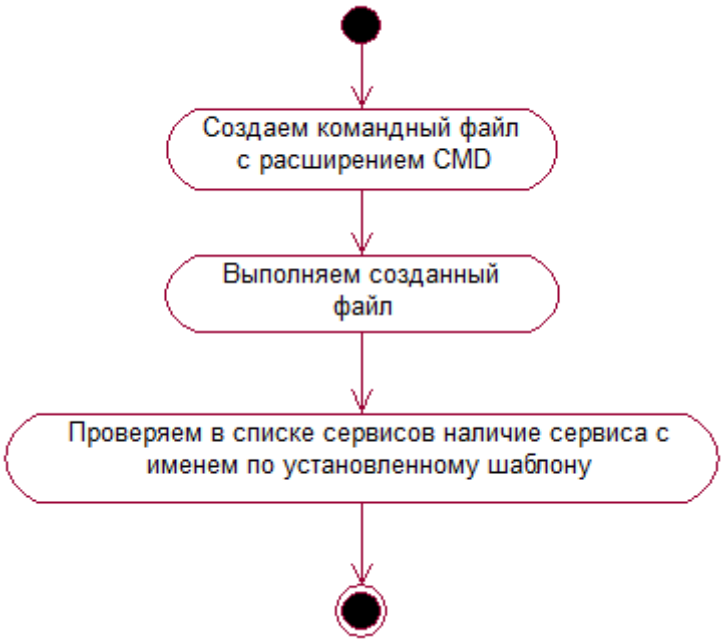

Рисунок 19 - Алгоритм создания WebLogic-сервиса для запуска приложения WebLogic AdminServer

Данный алгоритм состоит из следующих шагов:

Шаг 1. Создаем файл "C:\createSvc.cmd" со следующим содержимым:

```
echo off
SETLOCAL
set DOMAIN_NAME=MAIN_domain
set
USERDOMAIN_HOME=O:\Oracle\Middleware\user_projects\domains\MAIN_domain
set SERVER_NAME=MAIN_AdminServer
```

```
set PRODUCTION_MODE=false
set JAVA_VENDOR=Sun
set JAVA_HOME=O:\Oracle\Middleware\jdk160_24
set MEM_ARGS=-Xms256m -Xmx512m
call "O:\Oracle\Middleware\wlserver_10.3\server\bin\installsvc.cmd"
ENDLOCAL
```

Шаг 2. Выполняем созданный файл.

Шаг 3. Проверяем в списке сервисов наличие сервиса с именем по шаблону "beasvc DOMAIN_NAME_SERVER_NAME" (в нашем случае - "beasvc MAIN_domain_MAIN_AdminServer").

В приложении А представлен фрагмент программного кода (Java) для создания WebLogic-сервиса.

Выводы к главе 3

1. Для анализа эталонных архитектур промышленных облачных платформ на предмет построения облачной системы на основе предлагаемой модели использован архитектурный подход.

2. Анализ показал, что более соответствует современным требованиям архитектура Oracle SOA Cloud Service Architecture.

Заключение

Выпускная квалификационная работа посвящена проблеме исследования архитектур построения облачных систем.

Задачи обеспечения предоставления услуг облачных вычислений решаются с помощью облачных систем, которые представляет собой совокупность аппаратно- программного обеспечения и облачной ИТ-инфраструктуры.

Как показывает практика, эффективность и надежность облачной системы зависит от ее архитектуры.

Исследование архитектур построения облачных систем представляет актуальность и научно-практический интерес.

В процессе работы над ВКР решены следующие задачи:

1. Произведен сравнительный анализ существующих архитектур построения облачных систем. Результаты анализа показали, что наилучшими характеристиками обладает трехуровневая архитектура облачной системы с резервированием. Однако данная архитектура не обеспечивает необходимый уровень масштабирования облачной системы.

2. Проанализированы методологии проектирования архитектур информационных систем. Как показал анализ, в настоящее время для разработки архитектур информационных систем используются два подхода: разработка архитектуры на основе концепция EAI и сервис-ориентированной архитектуры – SOA. При этом наилучшими характеристиками обладает подход, основанный на сервис-ориентированной архитектуре. Главными преимуществами данной архитектуры являются масштабируемость и относительная простота реалиазции. Для описания архитектуры облачной системы используются концептуальный, логический и физический уровни представления. В качестве методологии проектирования описанных уровней архитектуры выбрана методология RUP. Разработанная в виде комплекса UML-диаграмм модель логической архитектуры является основой для

построения физической архитектуры облачной системы.

3. Проанализированы эталонные архитектуры промышленных платформ на предмет разработки физической модели архитектуры облачной системы на основе ее логической модели. Для анализа эталонных архитектур промышленных облачных платформ на предмет построения облачной системы на основе предлагаемой модели использован архитектурный подход. Рассмотрены архитектура корпоративной интеграции MS Azure и Oracle SOA Cloud Service Architecture. Как показал анализ лучшими характеристиками обладает решение Oracle, главными преимуществами которого являются гибкость и широкие функциональные возможности. Описан алгоритм и представлен фрагмент программного кода (Java) для создания WebLogic-сервиса.

Результаты бакалаврской работы представляют научно-практический интерес и могут быть рекомендованы разработчикам облачных систем.

Список используемой литературы

1. Базовые сценарии корпоративной интеграции в Azure [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/azure/architecture/reference-architectures/enterprise-integration/basic-enterprise-integration / (дата обращения: 05.05.2021).

2. Веб-сервисы и облачные вычисления [Электронный ресурс]. URL: https://www.lessons-tva.info/archive/nov032.html (дата обращения: 30.04.2021).

3. Галимянов А.Ф., Галимянов Ф.А. Архитектура информационных систем. – Казань: Казан. ун-т, 2019. – 117 с.

4. Данилов И. Г. Сервис-ориентированная архитектура как основа для построения современных распределённых систем // Известия ЮФУ. Технические науки. 2010. №7. С. 168-173.

5. Методология RUP [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Rational_Unified_Process (дата обращения: 30.04.2021).

6. Носова Л. С. Case-технологии и язык UML : учебно-методическое пособие. — Челябинск, Саратов : Южно-Уральский институт управления и экономики, Ай Пи Эр Медиа, 2019. 67 c. URL: http://www.iprbookshop.ru/81479.html (дата обращения: 04.05.2021).

7. Обзор архитектуры современных программных систем [Электронный ресурс]. URL: https://intuit.ru/studies/educational_groups/1070/courses/614/lecture/13316?page=3 (дата обращения: 30.04.2021).

8. Самуйлов С. В. Объектно-ориентированное моделирование на основе UML : учебное пособие. — Саратов : Вузовское образование, 2016. 37 c. URL: http://www.iprbookshop.ru/47277.html (дата обращения: 04.05.2021).

9. Создаем Веб Сервисы на платформе Oracle WebLogic Server [Электронный ресурс]. URL: https://habr.com/ru/post/111486/ (дата обращения: 05.05.2021).

10. Создание WebLogic-сервиса в Windows [Электронный ресурс]. URL: https://www.f-notes.info/oracle_wls:svc.

11. Трутнев Д. Р. Архитектуры информационных систем. Основы проектирования: Учебное пособие. – СПб.: НИУ ИТМО, 2012. – 66 с.

12. Administering Oracle SOA Cloud Service in a Customer-Managed Environment [Электронный ресурс]. URL: https://docs.oracle.com/en/cloud/paas/soa-cloud/csbcs/oracle-soa-cloud-service-architecture-oci.html (дата обращения: 30.04.2021).

13. Cloud Architecture: Everything You Need to Build a Cloud System [Электронный ресурс]. URL: https://solutionsreview.com/cloud-platforms/cloud-architecture-everything-you-need-to-build-a-cloud-system/ (дата обращения: 30.04.2021).

14. Cloud Computing Architecture [Электронный ресурс]. URL: https://www.javatpoint.com/cloud-computing-architecture (дата обращения: 30.04.2021).

15. Cloud computing architecture [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Cloud_computing_architecture#:~:text=Cloud%20computing%20architecture%20refers%20to,Internet%2C%20Intranet%2C%20Intercloud (дата обращения: 30.04.2021).

16. Cloud Computing System Architecture Diagrams [Электронный ресурс]. URL: https://docs.rightscale.com/cm/designers_guide/cm-cloud-computing-system-architecture-diagrams.html (дата обращения: 30.04.2021).

17. Couchbase [Электронный ресурс]. URL: https://www.couchbase.com/ (дата обращения: 30.04.2021).

18. Enterprise application integration [Электронный ресурс]. URL: https://medium.com/accesa/enterprise-application-integration-eai-df3e0d660482 (дата обращения: 30.04.2021).

19. LAMP [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/LAMP (дата обращения: 30.04.2021).

20. Rational Rose [Электронный ресурс]. URL:

https://www.ibm.com/products/software (дата обращения: 30.04.2021).

21. Role of Service-Oriented Architecture in Cloud Computing Explained [Электронный ресурс]. URL: https://www.hitechnectar.com/blogs/service-oriented-architecture-cloud-computing/ (дата обращения: 30.04.2021).

22. SOA - облачные вычисления [Электронный ресурс]. URL: https://ru.it-brain.online/tutorial/soa/soa_cloud_computing/ (дата обращения: 30.04.2021).

Приложение А

Фрагмент программного кода для создания WebLogic-сервиса

Используем: Weblogic8.1, apache-ant-1.6.5

Создаем проект с такой структурой:

WebServProject

src

my_service

Hellow.java

_build.xml

```
package my_service;
public class Hellow {
public String sayHellow(String inputStr, int inputInt) {
System.out.println("sayHello in webservices.basic.javaclass webservice has " +
"been invoked with arguments " + inputStr + " and " + inputInt);
return "This message brought to you by the " + "letter " + inputStr + " and the
number " + inputInt;
}
}
```

```
<project name="sample web service" default="all" basedir=".
<property name="wlLibHome" value="C:/bea/weblogic81/server/lib"/>
<property name="src" value="src"/>
<property name="dest" value="classes"/>
<path id="project.class.path
<pathelement location="${wlLibHome}/weblogic.jar"/>
<pathelement location="${wlLibHome}/webservices.jar"/>
</path>
<path id="new.project.class.path
```

```
<path refid="project.class.path"/>
<pathelement location="lib/temp.jar"/>
</path>
<taskdef name="servicegen"
classname="weblogic.ant.taskdefs.webservices.servicegen.Serv iceGenTask"
classpathref="project.class.path"/>
<taskdef name="clientgen"
classname="weblogic.ant.taskdefs.webservices.clientgen.Clien tGenTask"
classpathref="project.class.path"/>
<target name="Generate.Service" depends="build
<servicegen
destEar="ears/myWebService.ear"
contextURI="my_services" >
<classpath refid="new.project.class.path"/>
<service
expandMethods="true" generateTypes="true"
javaClassComponents="my_service.Hellow"
protocol="http" serviceName="HellowWS" serviceURI="/service/Hello"
style="rpc" targetNamespace="my_service
</service>
</servicegen>
&nbs p; System.out.println(responce);
}
}
```