

Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет информатики и
радиоэлектроники»

Номер зачетной книжки _____
Преддипломная практика зачтена с оценкой
_____ (_____)
(цифрой) (прописью)

(подпись руководителя практики от БГУИР)
_____._____.2024

ОТЧЕТ
по преддипломной практике
Место прохождения практики: ООО «СТЭКЛЭВЭЛ ГРУПП»
Сроки прохождения практики: с 25.03.2024 по 21.04.2024

Руководитель практики от
предприятия:

(подпись руководителя)
М.П. К.С. Улезко

Студент группы 013801

(подпись студента) К.Г. Хоменок
Руководитель практики от БГУИР
Шнейдеров Е.Н. – канд.техн.наук,
доцент

Минск 2024

СОДЕРЖАНИЕ

Введение.....	3
1 План-проспект дипломного проекта.....	4
2 Анализ исходных данных и постановка задач на дипломное проектирование	6
2.1 Анализ исходных данных к дипломному проекту	6
2.2 Обзор существующих облачных провайдеров по теме дипломного проекта.....	8
2.3 Обоснование и описание выбора облачного провайдера	16
2.4 Постановка задач на дипломное проектирование	18
3 Описание и проектирование облачной инфраструктуры	20
3.1 Terraform «инфраструктура как код»	20
3.2 Описание и обоснование используемых распределенных веб-сервисов.....	23
3.3 Контейнеризация и оркестрация с помощью Docker и Docker Compose	39
3.4 Описание и обоснование использования CI/CD	44
3.5 Проектирование облачной инфраструктуры.....	46
4 Практическая реализация облачной инфраструктуры для веб-сервиса	49
4.1 Обзор разворачиваемого веб-сервиса и используемых библиотек.....	49
4.2 Реализация инфраструктуры в виде кода для облачного окружения....	52
4.3 Настройка непрерывной интеграции и доставки (CI/CD)	63
4.4 Описание технологий разворачивания распределенного Web-сервиса с использованием Terraform.....	75
Заключение	76
Список используемых источников.....	77
Приложение А (обязательное) Листинги программного кода	79
Приложение Б (обязательное) Графический материал, поясняющий используемые DevOps-технологии	90

ВВЕДЕНИЕ

Современная технологическая эпоха требует эффективных и гибких инструментов для развертывания, управления и масштабирования веб-сервисов. В контексте этой потребности возникает актуальность исследования и разработки DevOps технологий для поддержки распределенных веб-сервисов на платформе AWS с использованием Terraform.

С увеличением объемов данных, скорости разработки и требований к безопасности, становится необходимым создание интегрированных и автоматизированных процессов управления инфраструктурой. DevOps методология предлагает подход, направленный на сближение разработки и эксплуатации, что позволяет сократить время развертывания проектов и улучшить их качество.

Целью данного дипломного проекта является разработка системы поддержки распределенных веб-сервисов на платформе AWS с применением DevOps технологий и инструментов автоматизации, основанных на Terraform.

Для достижения поставленной цели необходимо решить следующие задачи:

- изучение особенностей распределенных веб-сервисов для AWS;
- анализ требований и выбор соответствующих технологий;
- разработка и реализация инфраструктурной части системы с использованием Terraform;
- интеграция DevOps практик для оптимизации процессов развертывания и масштабирования веб-сервисов;
- тестирование разработанной системы.

Объектом данного дипломного проекта является развертывание распределенного веб-сервиса на платформе Amazon Web Services (AWS). Предметом исследования являются DevOps-технологии и практики, включая Continuous Integration (CI) и Continuous Deployment (CD), а также использование инструмента инфраструктуры как кода Terraform для автоматизации процессов развертывания и управления инфраструктурой сервиса.

1 ПЛАН-ПРОСПЕКТ ДИПЛОМНОГО ПРОЕКТА

Содержание плана-проспекта дипломного проекта представлено в таблице 1.1.

Таблица 1.1 – План-проспект дипломного проекта

Этап	Описание	Сроки выполнения
Введение	Актуальности темы дипломного проекта; формулировка цели и задач дипломного проектирования; выделение объекта и предмета исследования дипломного проектирования.	26.03.2024 – 27.03.2024
Анализ исходных данных и постановка задач на дипломное проектирование	Изучение особенностей распределенных веб-сервисов для AWS; определение функциональных требований к разворачиваемому веб-сервису; анализ существующих облачных провайдеров по теме дипломного проектирования; обоснование и описание выбора облачного провайдера; постановка задач на дипломное проектирование.	27.03.2024 – 31.03.2024
Описание и проектирование облачной инфраструктуры	Terraform «инфраструктура как код», описание и обоснование используемых распределенных веб-сервисов, контейнеризация и оркестрация с помощью Docker и Docker Compose, описание и обоснование использования CI/CD, проектирование и разработка облачной инфраструктуры.	01.04.2024 – 08.04.2024
Практическая реализация облачной инфраструктуры для веб-сервиса	Обзор разворачиваемого веб-сервиса, реализация инфраструктуры в виде кода для облачного окружения, настройка непрерывной интеграции и доставки (CI/CD), описание технологий разворачивания распределенного Web-сервиса с использованием Terraform	09.04.2024 – 15.04.2024

Продолжение таблицы 1.1

Этап	Описание	Сроки выполнения
Оценка количественных показателей функционирования разворачиваемого веб-сервиса	Оценка временных показателей; оценка ресурсных показателей.	22.04.2024 – 04.05.2024
Разработка графического материала	Выполнение чертежа «IDEF0 диаграмма декомпозиции», выполнение плаката «UML диаграмма развертывания веб-сервиса», выполнение плаката «Структура манифеста Terraform», выполнение плаката «Схема инфраструктуры AWS», выполнение плаката «Графический интерфейс веб-сервиса», выполнение чертежа «Схема алгоритма развертывания инфраструктуры»	16.04.2024 – 14.05.2024
Экономическое обоснование применения DevOps-технологий поддержки распределенных Web-сервисов	Выполнение задания по экономическому обоснованию применения DevOps-технологий поддержки распределенных Web-сервисов для AWS с использованием Terraform.	22.04.2024 – 04.05.2024
Оформление отчёта дипломного проекта	Составление введения, заключения, списка используемой литературы, приложений, ведомости дипломного проекта.	02.05.2024 – 20.05.2024

Таким образом, дата окончания работы над дипломным проектом должна быть не позднее 27.05.2024.

2 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ НА ДИПЛОМНОЕ ПРОЕКТИРОВАНИЕ

2.1 Анализ исходных данных к дипломному проекту

Анализ исходных данных к дипломному проекту представляет собой фундаментальный этап в разработке проекта, направленного на создание и поддержку распределенных веб-сервисов для платформы Amazon Web Services (AWS) с использованием DevOps технологий и инструментов автоматизации.

В соответствии с утвержденным приказом университета от 19.03.2024 № 595-с, тема проекта определена как «DevOps технологии поддержки распределенных Web сервисов для AWS с использованием Terraform». Этот проект предполагает разработку инфраструктуры и развертывания веб-сервиса для поддержки распределенных веб-сервисов на платформе AWS с использованием современных методов DevOps.

Исходные данные к дипломному проекту описывают систему распределенных веб-сервисов, предназначенных для платформы Amazon Web Services (AWS). Цель данной системы заключается в обеспечении поддержки распределенных веб-сервисов на платформе AWS с применением DevOps технологий и инструментов автоматизации, а также в оптимизации процессов развертывания и масштабирования.

В соответствии с функциональными требованиями, веб-сервис должен предоставлять возможность регистрации и аутентификации пользователей, наличие личного кабинета, реляционной базы данных, логирования действий пользователей и поддержки нескольких ролей. Инфраструктурная часть включает использование облачных виртуальных серверов (AWS EC2), облачных сетевых функций, Terraform для реализации «инфраструктуры как код», Docker и Docker Compose для контейнеризации программных средств, а также применение облачных технологий контейнеризации и CI/CD.

Требования к графическому интерфейсу предполагают его соответствие принципам инженерного дизайна и современным подходам к проектированию. Программное окружение описывается как Amazon Web Services (VPC, Route Table, Internet Gateway, Public Subnet, Security Group, EC2, RDS, S3, Elastic IP, IAM Policy, IAM Role, CloudWatch), Terraform, GitHub Actions. Все подключаемые библиотеки должны иметь некоммерческую лицензию.

Проектирование системы должно соответствовать документам, включая общие требования к дипломным проектам, стандарты ISO/IEC по качеству

систем и программного обеспечения, а также нормативные документы по разработке программного обеспечения и созданию документации пользователя.

Исходные данные содержат подробное описание системы, ее назначение, требования к функциональности, графическому интерфейсу, программному окружению, а также указания по соответствующим стандартам и нормативам, которые должны учитываться при проектировании и разработке данного проекта.

Для анализа исходных данных к дипломному проекту также необходимо изучить особенности распределенных веб-сервисов Amazon Web Services (AWS). Распределенные веб-сервисы являются ключевым элементом сетевых веб-сервисов, способствующим эффективной обработке запросов от множества пользователей. Однако, для оптимальной реализации таких сервисов необходимо учитывать ряд особенностей, характерных для инфраструктуры облачных вычислений, таких как AWS.

Первоначально, эффективное проектирование и развертывание распределенных веб-сервисов на AWS требует глубокого понимания архитектурных принципов, присущих облачным вычислениям. Это включает в себя управление ресурсами, автомасштабирование, отказоустойчивость и обеспечение безопасности данных.

Далее, адаптация инфраструктуры распределенных веб-сервисов под AWS включает в себя использование соответствующих сервисов и инструментов, предоставляемых платформой. Например, для обеспечения высокой доступности и отказоустойчивости можно использовать сервисы, такие как Amazon Elastic Compute Cloud (EC2) для развертывания веб-серверов в различных регионах, а Amazon Relational Database Service (RDS) для хранения данных.

Наконец, обеспечение безопасности является критическим аспектом при развертывании распределенных веб-сервисов на AWS. Это включает в себя использование средств аутентификации, авторизации, а также механизмов шифрования данных, предоставляемых AWS Identity and Access Management (IAM).

Изучение особенностей распределенных веб-сервисов для AWS требует комплексного подхода, объединяющего понимание принципов облачных вычислений с практическим применением соответствующих сервисов и инструментов, предоставляемых платформой AWS.

Функциональные требования к разворачиваемому веб-сервису, ориентированному на платформу WordPress, включают следующие аспекты:

1 Регистрация и авторизация пользователей: веб-сервис должен обеспечивать возможность регистрации новых пользователей и

аутентификации уже зарегистрированных. Это включает в себя формы для ввода учетных данных пользователей, проверку их достоверности и управление процессом аутентификации.

2 Наличие личного кабинета: необходимо, чтобы веб-сервис обеспечивал возможность зарегистрированным пользователям получать доступ к личному кабинету. В этом кабинете пользователи смогут просматривать и управлять своими персональными данными, профилем и другой информацией, связанной с их аккаунтом.

3 Наличие реляционной базы данных: для хранения информации о пользователях, их профилях, а также других сопутствующих данных (например, контента сайта) необходимо использовать реляционную базу данных. Это обеспечит структурированное хранение и эффективное управление данными.

4 Наличие логирования действий пользователей: веб-сервис должен осуществлять запись (логирование) действий пользователей, таких как вход в систему, изменение данных профиля, публикация контента и другие события, имеющие значение для безопасности и аналитики.

5 Наличие нескольких ролей: веб-сервис должен поддерживать механизм управления доступом на основе ролей пользователей. Это означает, что различным категориям пользователей должны быть назначены соответствующие роли с определенными правами доступа к функциональности сайта.

Проанализировав исходные данные к проекту, можно сделать вывод о необходимости разработки и поддержки распределенных веб-сервисов для AWS с использованием DevOps технологий. Требования к функциональности включают регистрацию и аутентификацию пользователей, наличие личного кабинета, реляционной базы данных, логирования действий пользователей и управления ролями. Эффективная реализация проекта требует учета особенностей облачных вычислений, выбора соответствующих сервисов AWS и соблюдения современных методов решения развертывания веб-сервиса.

2.2 Обзор существующих облачных провайдеров по теме дипломного проекта

Анализ ключевых облачных провайдеров позволяет выявить характеристики, предлагаемые услуги и инструменты, что способствует принятию обоснованного решения при выборе провайдера для реализации проектов.

Среди ведущих облачных провайдеров выделяются Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Alibaba Cloud и

Oracle Cloud. Каждый из них обладает уникальными особенностями, предлагая широкий спектр сервисов для обеспечения гибкости, масштабируемости и надежности в различных проектах (рисунок 2.1).



Рисунок 2.1 – Популярные облачные провайдеры [1]

В топ 5 облачных провайдеров входит AWS, Microsoft Azure, Google Cloud Platform (GCP), Alibaba и IBM Cloud [2].

Далее следует более детальное рассмотрение каждого облачного провайдера и его преимуществ в контексте дипломного проекта, ориентированного на поддержку распределенных веб-сервисов с использованием DevOps технологий.

2.2.1 Amazon Web Services как облачный провайдер

Первым облачным провайдером, который заслуживает внимания, является Amazon Web Services (AWS) – гигант в сфере облачных вычислений и один из лидеров среди топ-5 облачных провайдеров мира (рисунок 2.2).



Рисунок 2.2 – Облачный провайдер AWS [3]

AWS предлагает широкий спектр облачных услуг и инструментов, включая те, которые значительно облегчают применение методологии DevOps:

1 Вычислительные ресурсы: AWS предоставляет широкий спектр виртуальных машин (Amazon EC2) с различными типами экземпляров, а также управляемые контейнерные службы, такие как Amazon Elastic Container Service (ECS) и Amazon Elastic Kubernetes Service (EKS).

2 Хранилище данных: AWS предлагает различные сервисы хранения данных, такие как Amazon S3 для хранения неструктурированных данных, Amazon RDS для реляционных баз данных, Amazon DynamoDB для NoSQL баз данных и Amazon Redshift для аналитических данных.

3 Сетевые ресурсы: AWS предоставляет инструменты для создания и управления виртуальными сетями, балансировки нагрузки, контентной доставки, а также виртуальных частных сетей (Amazon VPC) для изоляции ресурсов.

4 Управление ресурсами и мониторинг: AWS предоставляет инструменты для автоматизации развертывания и управления инфраструктурой, такие как AWS CloudFormation, а также сервисы мониторинга и журналирования, такие как Amazon CloudWatch и AWS CloudTrail.

5 Инструменты разработки: AWS предлагает набор инструментов для разработчиков, включая AWS CodeCommit (сервис хранения кода), AWS CodeBuild (сервис для сборки кода), AWS CodeDeploy (сервис для автоматизации развертывания) и AWS CodePipeline (сервис для создания непрерывной интеграции и доставки).

6 Интеграция с открытыми источниками: AWS поддерживает множество открытых технологий и инструментов, таких как Docker, Kubernetes, Jenkins, Git и другие, что делает его привлекательным выбором для компаний, работающих в среде DevOps.

7 Безопасность: AWS обеспечивает широкий спектр инструментов и служб для обеспечения безопасности программных средств и данных, включая механизмы аутентификации, управления доступом, шифрования данных и т. д.

Amazon Web Services (AWS) была запущена в 2006 году и стала одним из ведущих облачных провайдеров в мире. AWS предоставляет 105 зон доступности в 33 географических регионах с планами на расширение в 18 новых зон доступности и 6 новых регионов.

AWS предлагает широкий спектр сервисов (более 210). С долей рынка облачных услуг в 33% (по состоянию 2021 года), AWS является лидером в этой области, что делает предпочтительным выбором AWS для многих

организаций. Amazon EC2 предоставляет мощные виртуальные машины с возможностью масштабирования, а Amazon VPC позволяет создавать изолированные сети в облаке [4].

Среди клиентов AWS такие крупные компании, как Netflix, BMW и Samsung. Ценовая политика AWS включает гибкую ценовую модель, включая оплату за час, резервированные экземпляры и бесплатный уровень услуг.

Чистый объем продаж Amazon Web Services (AWS) вырос с примерно 8 миллиардов долларов в 2015 году до более чем 17 миллиардов долларов в 2017 году, достиг 35 миллиардов долларов к 2019 году и в настоящее время составляет 96,8 миллиарда долларов в год. Этот ошеломляющий рост был обусловлен расширением AWS с 32 зон доступности в 2015 году до 52 зон доступности в 2017 году и до 105 зон доступности в настоящее время. Параллельно компания запустила тысячи новых сервисов AWS, что также способствовало ее росту [6].

С учетом высокой производительности, масштабируемости и безопасности AWS остается одним из наиболее популярных облачных провайдеров среди разработчиков и инженеров по всему миру. Все эти факторы делают AWS предпочтительным выбором для различных типов проектов, включая дипломные работы, ориентированные на разработку и поддержку распределенных веб-сервисов. Возможность использования инфраструктуры как кода, широкий спектр сервисов DevOps и высокие стандарты безопасности делают AWS привлекательным решением для разработчиков, стремящихся к успешной реализации своих проектов в облаке.

2.2.2 Microsoft Azure как облачный провайдер

Заголовочная «About» страница, которая рассказывает о том, что такое Microsoft Azure объясняет, – «Облачная платформа Azure включает более 200 продуктов и облачных служб, которые помогут в создании новых решений для сегодняшних и будущих задач. Создавайте и запускайте приложения и управляйте ими в нескольких облаках, локально и в пограничной среде, используя удобные для вас инструменты и платформы». Логотип представлен на рисунке 2.3 [5].



Рисунок 2.3 – Облачный провайдер Microsoft Azure

Microsoft Azure предоставляет разнообразные услуги для обеспечения вычислительных ресурсов, хранилища данных, сетевых ресурсов, управления ресурсами и мониторинга, инструментов разработки, интеграции с открытыми источниками и обеспечения безопасности. В числе предоставляемых услуг – виртуальные машины различных конфигураций, хранилище данных Azure Blob Storage, Azure SQL Database, Azure Cosmos DB, виртуальные сети, службы балансировки нагрузки, а также инструменты разработки, такие как Azure DevOps Services, Azure DevTest Labs и Azure Repos. Безопасность также является одним из приоритетов Azure, что подтверждается инвестициями в сумме более 1 миллиарда долларов США за последний год. Обновления и расширения платформы продолжаются, включая разработку новых инструментов и услуг для облегчения процесса разработки, развертывания и управления веб-сервисами.

Облачный провайдер Microsoft Azure был запущен в 2010 году и стал одним из ведущих в мире. Azure предоставляет доступ к более чем 64 регионам по всему миру и более чем 126 зонам доступности. Клиенты могут выбирать из различных географических локаций в зависимости от своих потребностей.

Azure предлагает более 200 облачных сервисов, включая вычисления (виртуальные машины, контейнеры и сервера), хранение данных (Azure Blob Storage, Azure SQL Database, Azure Cosmos DB), искусственный интеллект (Azure Machine Learning, Azure Cognitive Services), сетевые решения (виртуальные сети, балансировщики нагрузки, VPN-шлюзы), аналитику и большие данные (Azure Data Lake, Azure HDInsight).

С долей рынка облачных услуг в 20%, Azure является важным игроком на рынке, набирая популярность среди предприятий и разработчиков. В Azure используются виртуальные машины, аналогичные Amazon EC2, для развертывания программных средств и обработки данных. Azure Virtual Network (VNet) позволяет создавать изолированные сети в облаке и управлять трафиком.

Среди клиентов Azure такие крупные компании, как Adobe, BMW и General Electric. Ценовая политика Azure основана на оплате использованных ресурсов за час.

По всему миру расположено более 300 физических центров обработки данных Microsoft Azure, в которых размещены компьютерные серверы, каждый из которых оснащен независимым питанием, охлаждением и сетями. Компания соединяет инфраструктуру дата-центров более чем 175 000 миль волоконно-оптических линий связи в 140 странах.

Доходы Microsoft от интеллектуального облака, включающего в себя доходы от Azure, других облачных сервисов и серверных продуктов, в

последнем квартале достигли 25,9 млрд долларов, что на 19 % больше, чем в прошлом году. Таким образом, в годовом исчислении доход Microsoft от Intelligent Cloud составляет 103,5 миллиарда долларов. Однако Microsoft не раскрывает информацию о доходах от Azure, что означает, что Azure – это лишь часть общего дохода от Intelligent Cloud.

2.2.3 Google Cloud Platform как облачный провайдер

Google Cloud Platform (GCP) представляет собой мощный облачный провайдер, который предоставляет разнообразные услуги и инструменты для компаний, занимающихся разработкой и поддержкой программного обеспечения в контексте DevOps. Одним из ключевых преимуществ GCP является его гибкая и масштабируемая вычислительная инфраструктура, которая позволяет эффективно управлять вычислительными ресурсами и разворачивать программное обеспечение в облаке с использованием виртуальных машин и контейнерных сред, таких как Kubernetes Engine. Логотип GCP представлен на рисунке 2.4.



Рисунок 2.4 – Google Cloud Platform

GCP предлагает широкий набор сервисов для хранения данных, включая Cloud Storage для объектов, Cloud SQL для реляционных баз данных и Firestore для NoSQL хранилищ, что обеспечивает разработчикам разнообразные решения для хранения и обработки данных.

Важным аспектом в области DevOps является эффективное управление и мониторинг ресурсов. GCP предоставляет инструменты автоматизации процессов развертывания и управления инфраструктурой, такие как Cloud Build и Stackdriver, которые позволяют разработчикам легко контролировать производительность и доступность своих проектов.

Инструменты разработки на GCP, такие как Cloud Source Repositories и интеграция с Jenkins, обеспечивают командам DevOps средства для эффективной совместной работы над кодом и автоматизации процессов разработки и развертывания.

Сервисы, созданные Google, используют ту же облачную инфраструктуру, которую компания использует для своих внутренних продуктов, таких как Gmail, Google Search, Google Photos и YouTube.

Год основания Google Cloud Platform – 2008 год. С тех пор GCP стал одним из ведущих провайдеров облачных услуг в мире. На сегодняшний день Google Cloud насчитывает 40 регионов и 121 зону доступности. Эти регионы и зоны доступности расположены в США, Северной и Южной Америке, Европе, Азиатско-Тихоокеанском регионе, а также на Ближнем Востоке и в Африке. Клиенты могут выбирать из различных географических локаций в соответствии с их потребностями.

Google Cloud Platform предлагает широкий спектр сервисов, включая вычисления (виртуальные машины, контейнеры, сервера), хранилище данных (Google Cloud Storage, Google Cloud SQL), искусственный интеллект (Google AI Platform, TensorFlow), сеть (Virtual Private Cloud, Google Cloud Load Balancing).

Доля рынка GCP в облачных услугах составляет 9% и продолжает расти. Это делает его значимым игроком на рынке, привлекательным для разработчиков и организаций. В GCP используются виртуальные машины, предоставляющие масштабируемые ресурсы для развертывания проектов и обработки данных. Virtual Private Cloud (VPC) в GCP позволяет создавать изолированные сети и управлять трафиком.

Среди клиентов GCP такие компании, как Snap Inc., Spotify и HSBC. Ценообразование в Google Cloud Platform зависит от использованных ресурсов и предлагает гибкие опции оплаты за использованные ресурсы.

Подразделение Google Cloud компании Alphabet Inc получает доход за счет платы, взимаемой за инфраструктуру, платформу и другие услуги. В последнем квартале выручка Google Cloud составила 9,2 млрд долларов, что на 26 % больше, чем в прошлом году. Таким образом, в годовом исчислении выручка Google Cloud составляет 37 миллиардов долларов.

2.2.4 Alibaba Cloud как облачный провайдер

Alibaba Cloud, как облачный провайдер, играет ключевую роль в сфере DevOps, предоставляя комплексные решения для ускорения процессов разработки и обеспечения непрерывной доставки программных средств. С помощью своих сервисов и инструментов, Alibaba Cloud способствует созданию эффективной и безопасной среды для DevOps, что позволяет компаниям быстро адаптироваться к изменениям и оптимизировать рабочие процессы.

Подразделение Alibaba Group, занимающееся облачными вычислениями и известное как Alibaba Cloud, является четвертым по величине поставщиком облачных услуг в мире, основным поставщиком облачных услуг в Азиатско-Тихоокеанском регионе и крупнейшим поставщиком облачных услуг в Китае. Через Alibaba Cloud компания предлагает облачные услуги, включая эластичные вычисления, базы данных, хранилища, сетевую виртуализацию, крупномасштабные вычисления, безопасность, управление и аналитика больших данных, и машинное обучение.

В настоящее время Alibaba Cloud работает в 30 регионах и 89 зонах доступности. В континентальном Китае Alibaba является доминирующим поставщиком облачных услуг, имея 15 регионов по всей стране. За пределами материкового Китая Alibaba Cloud работает в США, Европе, Азиатско-Тихоокеанском регионе и на Ближнем Востоке.

Alibaba Group в основном получает доходы от облачных вычислений от корпоративных клиентов в зависимости от продолжительности и использования их услуг. Выручка подразделения Cloud Intelligence Group, в которое входит Alibaba Cloud, за последний квартал составила 3,95 млрд долларов (28 066 млн юаней), что на 3 % больше, чем в прошлом году. Таким образом, в годовом исчислении выручка Alibaba Cloud в настоящее время составляет 15,8 млрд долларов.

В заключение, Alibaba Cloud предоставляет мощные инструменты и сервисы для DevOps, которые помогают компаниям достигать высокого качества и эффективности в доставке программных средств. С помощью продвинутых технологий и интеграции с другими продуктами Alibaba Cloud, DevOps-команды могут реализовывать сложные проекты с уверенностью в стабильности и безопасности своих решений [7][8].

2.2.5 Oracle Cloud как облачный провайдер

Oracle Cloud представляет собой облачную платформу, которая предлагает решения для DevOps, обеспечивая непрерывную интеграцию и непрерывную доставку (CI/CD) для команд разработчиков, работающих на Oracle Cloud Infrastructure (OCI). Это интегрированная служба, которая обеспечивает согласованность идентификации, безопасности, логирования и других аспектов инфраструктуры Oracle Cloud, а также предварительно настроенные безопасные развертывания в OCI Compute Services.

OCI DevOps гибко интегрируется с существующими рабочими процессами и инструментами, такими как GitHub, GitLab, Jenkins и другие, включая частные и облачные ресурсы. Это позволяет командам разработчиков

сосредоточиться на коде и рабочих процессах, не заботясь о серверах, поскольку платформа масштабируется для поддержки параллельных сборок.

Облачные сервисы корпорации Oracle включают в себя Oracle Cloud Software-as-a-Service (SaaS) и Oracle Cloud Infrastructure (OCI). В рамках OCI компания является поставщиком облачных услуг, предоставляя инфраструктурные технологии как услугу, включая вычисления, хранение и сетевые сервисы.

В настоящее время Oracle Cloud имеет 48 регионов и 58 зон доступности. Эти регионы и зоны доступности расположены в США, Канаде, Европе, на Ближнем Востоке и в Африке (ЕМЕА), Латинской Америке и Азиатско-Тихоокеанском регионе. Кроме того, Oracle Cloud предлагает правительственные облачные регионы для правительства США, Министерства обороны США (DoD) и правительства Великобритании, а также два суверенных региона для клиентов из стран Европейского союза (ЕС).

Oracle Cloud Infrastructure (OCI) обычно взимает предоплату, которая постепенно снижается по мере потребления услуг OCI клиентом в течение определенного периода времени. В последнем квартале выручка Oracle Cloud IaaS достигла 1,8 млрд долларов, увеличившись на 49% по сравнению с предыдущим годом. Таким образом, в годовом исчислении выручка Oracle Cloud составляет 7,2 млрд долларов.

2.3 Обоснование и описание выбора облачного провайдера

Для обоснования выбора облачного провайдера необходимо провести анализ и сравнительную оценку нескольких провайдеров облачных услуг, включая AWS, в контексте требований и целей дипломного проекта. Этот анализ должен включать в себя составление сравнительных таблиц, в которых будут отображены ключевые параметры и характеристики каждого провайдера.

Сравнительные таблицы позволят объективно оценить преимущества и недостатки каждого провайдера и определить, какой из них лучше соответствует требованиям и целям дипломного проекта. Кроме того, при анализе необходимо учитывать факторы, специфичные для проекта, такие как поддержка DevOps методологии, совместимость с выбранными инструментами разработки и управления инфраструктурой, а также поддержка распределенных веб-сервисов.

После проведения анализа и сравнительной оценки провайдеров необходимо будет принять обоснованное решение о выборе оптимального облачного провайдера для реализации дипломного проекта.

В таблице 2.1 представлена сравнительная таблица облачных провайдеров (AWS, Azure, GCP).

Таблица 2.1 – Сравнительная таблица облачных провайдеров (AWS, Azure, GCP)

	AWS	Azure	GCP
Год запуска	2004	2010	2008
Зоны доступности	105	126	121
Регионы	33	64	40
Сервисы	210+	200+	100+
Облачный рынок	33%	21%	8%
Вычислительный движок	EC2 (Elastic Compute Search)	Virtual Machine	Compute Engine
Сеть	VPC (Virtual Private Cloud)	VNET (Virtual Network)	Cloud Virtual Network
Ценовая политика	по часам	по минутам	по минутам

В таблице 2.2 представлен сравнительный анализ облачных провайдеров (Alibaba Cloud, Oracle Cloud).

Таблица 2.2 – Сравнительная таблица облачных провайдеров (Alibaba Cloud и Oracle Cloud)

	Alibaba Cloud	Oracle Cloud
Год запуска	2009	2016
Зоны доступности	89	58
Регионы	30	48
Сервисы	–	–
Облачный рынок	7.7%	2.1%
Вычислительный движок	Alibaba ECS	Compute
Сеть	VPC (Virtual Private Cloud)	VCN (Virtual Cloud Network)
Ценовая политика	постоянная или по мере использования	постоянная или по мере использования

После анализа и сравнительной оценки нескольких ведущих облачных провайдеров, таких как AWS, Azure, GCP, Alibaba Cloud и Oracle Cloud, было

принято обоснованное решение о выборе AWS в качестве оптимального варианта.

Одним из основных факторов, подтверждающих выбор AWS, является его лидерство на мировом рынке облачных провайдеров. По данным сравнительной таблицы, AWS занимает второе место по количеству зон доступности, регионов и предоставляемых сервисов. Но приоритетом является то, что AWS имеет значительно большую долю рынка в сравнении с конкурентами, что свидетельствует о высокой степени доверия со стороны пользователей.

Кроме того, AWS предлагает широкий спектр вычислительных движков и сетевых решений, таких как EC2 и VPC, которые хорошо подходят для реализации распределенных веб-сервисов. Ценовая политика AWS, основанная на оплате за использование по часам, также является привлекательным фактором для студентов и начинающих специалистов, позволяя экономить ресурсы при разработке и тестировании проекта.

Важным аспектом выбора AWS является его простота в использовании и доступность документации. Интуитивно понятный интерфейс платформы и обширная база знаний делают процесс разработки и управления инфраструктурой более эффективным и прозрачным.

Таким образом, на основании представленного сравнительного анализа и учета основных факторов, обоснованным решением является выбор облачного провайдера AWS для реализации дипломного проекта. Учитывая все вышеперечисленные факторы, выбор Amazon Web Services (AWS) в качестве облачного провайдера для проекта по разработке и поддержке распределенных веб-сервисов на платформе AWS с использованием DevOps технологий и инструментов автоматизации является логичным и обоснованным решением. Все аспекты, начиная от широкого набора сервисов и высокой производительности, и заканчивая доступностью знаний и ресурсов поддержки, делают AWS оптимальным выбором для успешной реализации проекта.

2.4 Постановка задач на дипломное проектирование

Постановка задач на дипломное проектирование предполагает развертывание веб-сервиса на инфраструктуре облачного провайдера Amazon Web Services (AWS) с использованием распределенных веб-сервисов и инструментов DevOps. Для автоматизации управления инфраструктурой и ее конфигурацией необходимо применять Terraform, – инструмент инфраструктурного кодирования.

Перед началом развертывания веб-сервиса и применения DevOps-технологий необходимо провести анализ задания к дипломному проекту. Данный этап играет ключевую роль в формировании требований и ожиданий, предъявляемых к разрабатываемому дипломному проекту. Анализ задания позволяет выявить основные задачи и цели проекта, определить требования к функциональности и качеству разрабатываемого веб-сервиса, а также установить рамки и ограничения, с которыми необходимо работать. В типовом задании к дипломному проекту прописываются основные этапы работы, ожидаемые результаты и критерии оценки.

Графические материалы и пояснительная записка к дипломному проекту оформляются согласно установленным стандартам и требованиям. Используются соответствующие ГОСТы, регламентирующие оформление документации и графических материалов.

Проведение анализа документации по облачному провайдеру AWS, Terraform и реализации CI/CD позволит ознакомиться с теоретическими и практическими аспектами, необходимыми для успешной реализации дипломного проекта. Это включает изучение основных концепций и функциональных возможностей данных технологий, а также методов их применения в практике.

Следующим этапом является реализация DevOps технологий поддержки распределенных веб-сервисов для AWS с использованием Terraform в соответствии с требованиями, определенными в типовом задании дипломного проекта. Это включает проектирование инфраструктуры, ее описание и последующий запуск (создание) для тестирования или деплоя, а также реализацию DevOps-практики CI/CD с использованием Terraform и GitHub Actions.

Написание пояснительной записки и разработка графического материала являются важными составляющими процесса дипломного проектирования. Пояснительная записка должна содержать описание всех этапов работы, принятых решений и полученных результатов, а также анализ выполненной работы. Графический материал должен наглядно иллюстрировать основные аспекты проекта и поддерживать текст пояснительной записки.

Завершающим этапом, подтверждающим успешное выполнения задания к дипломному проекту, является защита как преддипломной практики, так и самого дипломного проекта.

3 ОПИСАНИЕ И ПРОЕКТИРОВАНИЕ ОБЛАЧНОЙ ИНФРАСТРУКТУРЫ

3.1 Terraform «инфраструктура как код»

HashiCorp Terraform представляет собой инструмент инфраструктуры в формате кода, предназначенный для определения облачных и локальных ресурсов в читаемых человеком конфигурационных файлах. Эти файлы могут быть версионированы, повторно и совместно использоваться. Помимо этого, Terraform использует последовательный рабочий процесс для обеспечения и управления всей инфраструктурой на протяжении ее жизненного цикла. Возможности Terraform включают управление как низкоуровневыми компонентами, такими как вычислительные ресурсы, ресурсы хранения и сетевые ресурсы, так и высокоуровневыми компонентами, такими как записи DNS и функции SaaS.

Terraform создает и управляет ресурсами на облачных платформах и других сервисах через их интерфейсы прикладного программирования (API). Провайдеры позволяют Terraform работать практически с любой платформой или сервисом с доступным API.

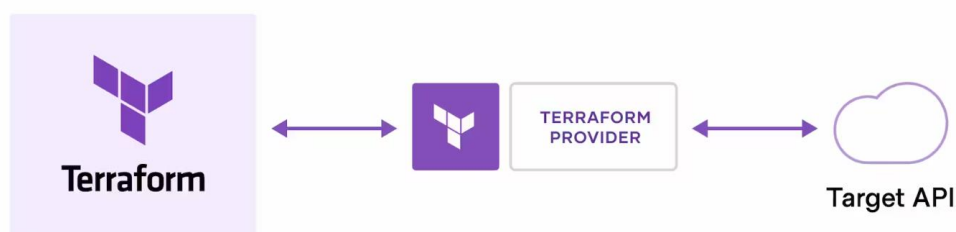


Рисунок 3.1 – Принцип работы Terraform [9]

В основе работы Terraform и принципа инфраструктуры как код лежит процесс, начинающийся с загрузки конфигурационных файлов Terraform. Эти файлы описывают желаемую инфраструктуру, включая ресурсы и их параметры. Этот этап представляет собой начальную точку, с которой Terraform начинает свою работу по созданию, изменению или удалению инфраструктурных ресурсов в соответствии с описанием, предоставленным в конфигурационных файлах. Ниже приведен алгоритм работы Terraform:

1 Анализ конфигурации: Terraform анализирует конфигурационные файлы и создает внутреннее представление желаемой инфраструктуры в виде

графа зависимостей между ресурсами. Этот граф описывает порядок, в котором ресурсы должны быть созданы и связаны друг с другом.

2 Инициализация провайдера: Terraform обращается к указанным провайдерам, используя предоставленные в конфигурации ключи доступа и настройки. Провайдеры – это компоненты, которые взаимодействуют с API конкретного облачного провайдера или другой системы для управления ресурсами.

3 Планирование изменений: Terraform включает в себя сравнение текущего состояния инфраструктуры, которое сохранено локально в файлах состояния, с желаемым состоянием, описанным в конфигурации. В результате этого процесса определяются ресурсы, которые требуется добавить, изменить или удалить для достижения желаемого состояния. На основе этого сравнения Terraform генерирует план изменений, который содержит необходимые действия для синхронизации текущего и желаемого состояний инфраструктуры.

4 Применение изменений: после того как план изменений сгенерирован и подтвержден пользователем, Terraform начинает внесение изменений в инфраструктуру, взаимодействуя с API провайдера для создания, изменения или удаления ресурсов. Terraform следит за порядком и зависимостями ресурсов, чтобы убедиться, что изменения применяются в правильном порядке.

5 Обновление состояния: по мере применения изменений Terraform обновляет локальное состояние инфраструктуры, чтобы отразить актуальное состояние. Это позволяет Terraform отслеживать текущее состояние инфраструктуры и использовать его при следующих операциях управления.

Таким образом, Terraform обеспечивает автоматизированное управление инфраструктурой с использованием концепции инфраструктуры как кода, упрощая процесс развертывания и управления облачными ресурсами.

NashiCorp и сообщество Terraform разработали значительное количество провайдеров для управления разнообразными типами ресурсов и сервисов. В реестре Terraform представлены все общедоступные провайдеры, включая Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, Docker и многие другие.

Основной рабочий процесс Terraform включает три этапа:

1 Написание манифеста: на этом этапе определяются ресурсы, которые могут присутствовать у различных облачных провайдеров и сервисов. Например, необходимо создать конфигурацию для развертывания веб-сервиса на виртуальных машинах в сети виртуального частного облака (VPC) с группами безопасности и балансировщиком нагрузки.

2 Планирование (plan): Terraform генерирует план выполнения, который описывает инфраструктуру, предполагаемую для создания, обновления или уничтожения. Этот план формируется на основе существующего состояния инфраструктуры и конфигурации, предоставленной пользователем.

3 Применение (apply): после подтверждения пользователя Terraform выполняет предложенные операции в правильном порядке, учитывая все зависимости от ресурсов. Например, если происходит обновление свойств VPC и изменение количества виртуальных машин в этом VPC, Terraform создает VPC заново перед масштабированием виртуальных машин. Один из полезных функциональных возможностей Terraform заключается в возможности выполнения проверки после написания инфраструктуры в виде кода с помощью команды `terraform plan` в терминале. Эта команда, как правило, выявляет семантические ошибки, если таковые имеются, и выводит подробный план построения инфраструктуры, включая этапы выполнения (см. рисунок 3.2).

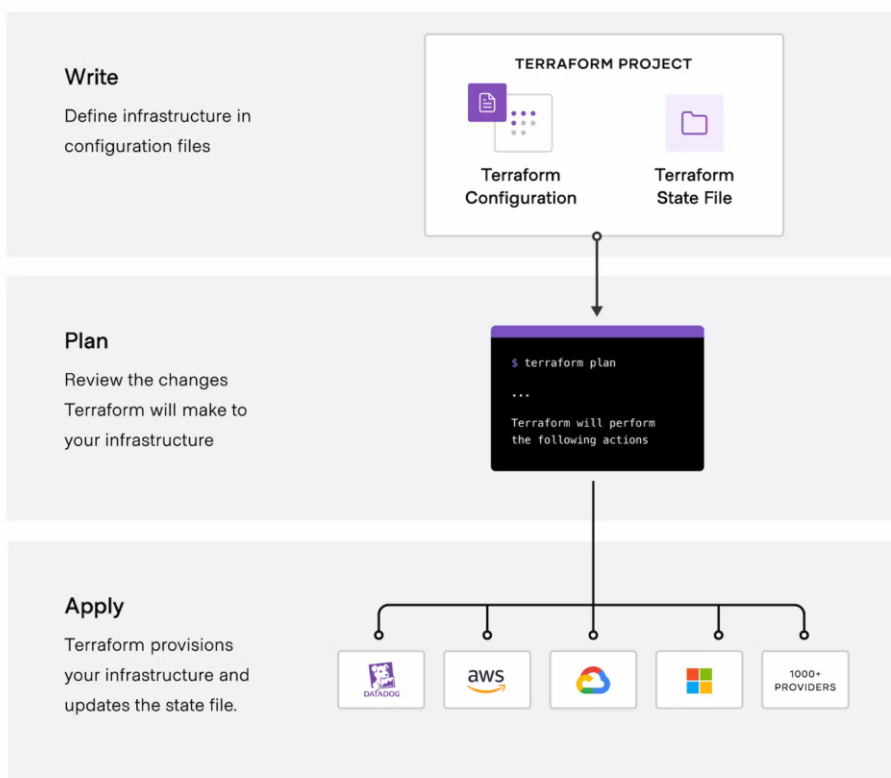


Рисунок 3.2 – Рабочий процесс Terraform [10]

Преимущества использования Terraform:

1 Управление любой инфраструктурой и провайдером: Terraform предоставляет доступ к реестру провайдеров для множества платформ и

сервисов, а также позволяет создавать собственные провайдеры. Использование неизменяемого подхода к инфраструктуре упрощает процесс обновления или модификации сервисов и инфраструктуры.

2 Отслеживание изменений инфраструктуры: Terraform генерирует план изменений и запрашивает подтверждение перед их применением. Файл состояния служит источником истины для среды, а Terraform использует его для определения изменений, необходимых для соответствия конфигурации.

3 Автоматизация изменений: Файлы конфигурации Terraform являются декларативными, что позволяет описывать конечное состояние инфраструктуры без необходимости написания пошаговых инструкций. Terraform эффективно управляет логикой создания и изменения ресурсов, строя граф зависимостей и параллельно обрабатывая независимые ресурсы.

4 Стандартизация конфигураций: Terraform поддерживает модули – это многократно используемые компоненты конфигурации, что позволяет экономить время на создание настраиваемых коллекций инфраструктуры.

5 Совместная работа: Terraform позволяет фиксировать манифесты в системе контроля версий и использовать Terraform Cloud или другие облачные провайдеры для эффективного управления рабочими процессами в командах. Terraform Cloud обеспечивает безопасный доступ к общему состоянию, контроль доступа на основе ролей и другие полезные функции. Terraform – это программное обеспечение с открытым исходным кодом, которое позволяет описывать инфраструктуру кодом у специального провайдера, после чего проверять её, масштабировать и изменять.

Таким образом, Terraform, как инструмент (программное обеспечение) написания «инфраструктуры как кода», является важным элементом для того, чтобы эффективно и автоматизировано управлять инфраструктурой разворачиваемого проекта (веб-сервиса) в рамках данного дипломного проекта.

3.2 Описание и обоснование используемых распределенных веб-сервисов

Распределенные веб-сервисы имеют стратегическое значение в сфере построения облачной инфраструктуры, обеспечивая необходимую масштабируемость и надежность при предоставлении онлайн-услуг.

Данная глава направлена на анализ и обоснование выбора распределенных веб-сервисов, включая VPC, Route Table, Internet Gateway, Public Subnet, Security Group, RDS Instance, S3, Elastic IP, IAM Policy, IAM Role и CloudWatch.

В этом контексте следующим этапом является рассмотрение первого из распределенных веб-сервисов – Virtual Private Cloud (VPC).

3.2.1 VPC

Amazon Virtual Private Cloud (VPC) представляет собой сервис, предоставляющий возможность разворачивать ресурсы AWS в логически изолированной виртуальной сети, настраиваемой пользователем. Это обеспечивает полный контроль над окружением виртуальной сети, включая возможность определения собственного диапазона IP-адресов, создание подсетей, а также конфигурацию таблиц маршрутизации и сетевых шлюзов. Для большинства ресурсов в VPC доступны как IPv4, так и IPv6, что обеспечивает безопасный и удобный доступ к ресурсам и веб-сервисам (рисунок 3.3).

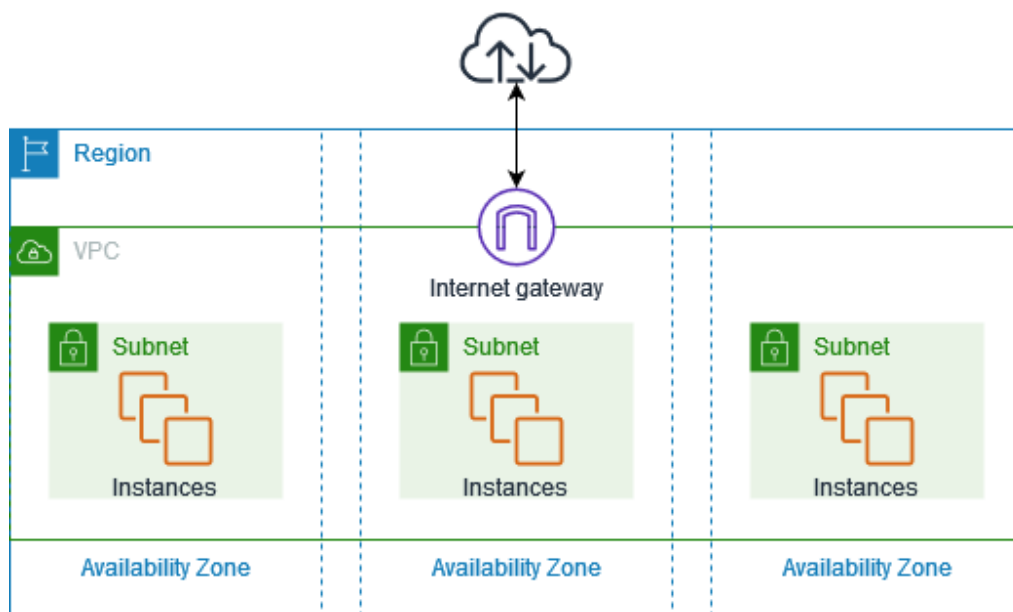


Рисунок 3.3 – Инфраструктура с использованием VPC [11]

В качестве одного из основополагающих сервисов AWS, Amazon VPC обеспечивает удобную настройку параметров сети VPC в соответствии с потребностями пользователя. Путем создания общедоступной подсети для веб-серверов с доступом к Интернету, а также частной подсети для серверных систем, таких как базы данных или сервера программных средств, пользователи могут эффективно организовать свою инфраструктуру. Amazon VPC также предлагает использование многоуровневой системы безопасности, включающей в себя группы безопасности и сетевые списки контроля доступа. Эта система обеспечивает контроль доступа к виртуальным машинам Amazon

Elastic Compute Cloud (Amazon EC2) в каждой подсети, повышая безопасность и управляемость сетевой инфраструктуры.

Помимо прочего, в рамках виртуальной частной сети (VPC) возможна настройка ряда сервисов, включая журналы потоков, IP Address Manager (IPAM) для управления IP-адресами, входящую маршрутизацию, Network Access Analyzer для анализа доступа к сети, список контроля доступа к сети, сетевой менеджер, а также Reachability Analyzer – инструмент для анализа статических конфигураций. Кроме того, в рамках VPC предусмотрены группы безопасности и зеркалирование трафика.

Отображение VPC на рисунке 3.3 подчеркивает его значимость как главного и основополагающего распределенного веб-сервиса. VPC выступает в качестве центральной сети инфраструктуры, существенно важной для организации, так как находится выше только по уровню региона (включая доступные зоны и/или зоны доступности).

Выбор сервиса Amazon Virtual Private Cloud (VPC) обоснован его ключевой ролью в построении безопасной, масштабируемой и надежной облачной инфраструктуры на платформе AWS. VPC позволяет создавать изолированные виртуальные сети с гибкой настройкой параметров, таких как диапазон IP-адресов, подсети и таблицы маршрутизации.

Добавление VPC в инфраструктуру разворачиваемого веб-сервиса является ключевым шагом для создания надежной и эффективной облачной инфраструктуры.

3.2.2 Route Table

Route Table – это один из сервисов AWS, основанный на AWS VPC, таблица маршрутов содержит набор правил, называемых маршрутами, которые определяют, куда направлять сетевой трафик из подсети или шлюза.

Основные концепции для таблицы маршрутов включают главную таблицу маршрутов, которая предоставляется автоматически с VPC и управляет маршрутизацией для всех подсетей, не связанных с другими таблицами маршрутов, а также пользовательскую таблицу маршрутов, создаваемую для конкретного VPC.

Каждый маршрут определяется понятиями «назначение» и «цель», где назначение указывает диапазон IP-адресов, куда должен быть направлен трафик (CIDR назначения), а цель определяет шлюз, сетевой интерфейс или соединение, через которые трафик будет направлен, например, интернет-шлюз.

Для связи таблиц маршрутов с подсетями, интернет-шлюзами или виртуальными частными шлюзами используется ассоциация таблиц

маршрутов, а таблица маршрутов подсети связывается непосредственно с определенной подсетью.

Локальный маршрут представляет собой маршрут по умолчанию для внутренней связи в пределах VPC, в то время как распространение маршрутов включает автоматическое добавление маршрутов для VPN-соединения в таблицы маршрутов подсетей при подключении виртуального частного шлюза к VPC.

Таблица маршрутов шлюза связана с интернет-шлюзом или виртуальным частным шлюзом, а ассоциация с границами используется для маршрутизации входящего трафика VPC на конкретное устройство.

Таблица маршрутов транзитного шлюза связана с транзитным шлюзом, тогда как таблица маршрутов локального шлюза связана с локальным шлюзом Outposts.

Использование распределенного веб-сервиса Route Table на AWS обосновано его ключевой ролью в управлении трафиком между различными сегментами виртуальной сети. Route Table предоставляет возможность определения маршрутов для сетевого трафика в зависимости от его назначения, что обеспечивает эффективную маршрутизацию внутри облачной инфраструктуры.

3.2.3 Internet Gateway

Интернет-шлюз представляет собой ключевой компонент горизонтально масштабируемой и высокодоступной виртуальной частной сети (VPC), обеспечивающий устойчивое соединение между VPC и Интернетом. С его помощью возможна передача трафика как по протоколу IPv4, так и IPv6, обеспечивая эффективное взаимодействие ресурсов как в публичных, так и в локальных сетях. Интернет-шлюз действует в качестве посредника для связи между ресурсами, обладающими публичными IP-адресами, и обеспечивает возможность инициирования соединений как с внешних ресурсов, так и из внутренней сети.

Специфическим функционалом интернет-шлюза является обеспечение соответствующих маршрутов в таблицах маршрутизации VPC для трафика, направляемого через Интернет. Кроме того, для обмена данными по протоколу IPv4, Internet Gateway выполняет функцию трансляции сетевых адресов (NAT), что способствует эффективному использованию ресурсов и обеспечивает безопасность сетевого трафика. Однако, в случае использования IPv6, такая функция не требуется, поскольку IPv6-адреса являются общедоступными и не требуют дополнительной трансляции.

Экземпляры EC2 в облаке AWS, а также другие ресурсы, размещенные в публичных подсетях, могут взаимодействовать с Интернетом благодаря наличию публичных IP-адресов и обработке трафика через интернет-шлюз. Этот механизм обеспечивает эффективное управление сетевым трафиком и обеспечивает надежное соединение как для облачных, так и для локальных сред.

Internet Gateway (IGW) в AWS играет ключевую роль в обеспечении доступа к интернету для виртуальных ресурсов внутри Amazon Virtual Private Cloud (VPC). Этот компонент инфраструктуры позволяет виртуальным серверам и сервисам в VPC обмениваться данными с внешним миром, обеспечивая доступ к внешним ресурсам, веб-сервисам и клиентам через интернет.

В рамках дипломного проекта «DevOps технологии поддержки распределенных Web сервисов для AWS с использованием Terraform» использование Internet Gateway обосновано необходимостью обеспечения связности и доступности веб-сервисов и сервисов, развернутых в облаке AWS. IGW гарантирует высокую доступность внешнего доступа к веб-сервисам, обеспечивает масштабируемость и гибкость в управлении сетевым трафиком, а также интегрируется с другими сервисами AWS, что позволяет создавать комплексные сетевые конфигурации с учетом требований безопасности и производительности.

3.2.4 Public Subnet

Public Subnet в Amazon Virtual Private Cloud (VPC) играет важную роль в архитектуре облачных компонентов, предоставляя среду для размещения публичных ресурсов, которые требуют доступности извне. Этот сегмент сети обеспечивает прямой доступ к интернету через Internet Gateway на AWS, что позволяет веб-серверам, API-шлюзам и другим публичным ресурсам взаимодействовать с внешними клиентами и сервисами.

Использование Public Subnet обосновано необходимостью предоставления доступа к веб-сервису извне, чтобы потенциальный пользователь мог зайти по адресу домена веб-сервиса и получить к нему доступ. Этот подход позволяет изолировать публичные ресурсы от приватных, повышая уровень безопасности и обеспечивая контроль доступа к важным данным и сервисам. Такая конфигурация позволяет эффективно управлять и обеспечивать безопасность веб-инфраструктуры в облаке AWS, что является ключевым аспектом современных распределенных веб-сервисов.

Это включает в себя обслуживание внешних запросов и обмен информацией с внешним миром. При этом изоляция публичных ресурсов от

приватных обеспечивает повышенный уровень безопасности и контроля доступа к важным данным и сервисам.

3.2.5 Private Subnet

Private Subnet в Amazon Virtual Private Cloud (VPC) представляет собой сегмент сети, который не имеет прямого доступа к интернету и используется для размещения приватных ресурсов, таких как базы данных Amazon RDS. Этот компонент обеспечивает изоляцию приватных ресурсов от внешнего интернета, обеспечивая дополнительный уровень безопасности для данных, хранимых и обрабатываемых в облаке AWS.

Выбор использования Private Subnet с Amazon RDS обоснован необходимостью ограничения доступа к базе данных из внешних сетей. Этот подход способствует повышению уровня безопасности данных, хранимых в базе данных, и предотвращает несанкционированный доступ извне. Подключение базы данных к Private Subnet ограничивает доступ к ней только из ресурсов, находящихся в той же сети, что и RDS, обеспечивая дополнительный слой защиты.

При развертывании веб-сервиса на экземпляре EC2, который находится в публичной подсети (Public Subnet), взаимодействие с базой данных RDS, находящейся в Private Subnet, ограничивается. Экземпляр EC2 предоставляет доступ пользователям извне к веб-сервису, поскольку обладает Internet Gateway и принимает внешний трафик. Однако, Private Subnet принимает трафик исключительно из локальной сети, в которой размещен EC2, и не предоставляет доступ извне. Таким образом, доступ к базе данных RDS возможен только из экземпляра EC2, что создает дополнительный уровень безопасности. Учитывая, что доступ к EC2 осуществляется через SSH-ключ, скомпрометировать который является нереальной задачей, обеспечивается высокий уровень безопасности для базы данных RDS.

3.2.6 Security Group

Распределенный веб-сервис Security Group на AWS представляет собой виртуальный брандмауэр, который контролирует трафик входящий и исходящий для экземпляров Amazon EC2 в Amazon Virtual Private Cloud (VPC). Security Group позволяет определять правила доступа на основе IP-адресов, портов и протоколов, обеспечивая тем самым безопасность виртуальных серверов.

Выбор использования Security Group обоснован необходимостью обеспечения безопасности веб-сервиса в облаке AWS. Использование Security

Group позволяет настраивать правила доступа к виртуальным серверам и сервисам в соответствии с требованиями безопасности проекта, например, блокируя доступ к нежелательным IP-адресам или разрешая доступ только определенным портам и протоколам.

Кроме того, Security Group обеспечивает гибкость и управляемость в управлении правилами безопасности, позволяя администраторам быстро реагировать на изменения в требованиях безопасности и внесение соответствующих правок в конфигурацию. Таким образом, использование распределенного веб-сервиса Security Group в рамках проекта обеспечивает надежную защиту виртуальных серверов в облачной среде AWS, что является критически важным аспектом для обеспечения безопасности и целостности данных и сервисов.

3.2.7 EC2

Эластичное облако вычислений Amazon (Amazon EC2) предлагает самую широкую и глубокую вычислительную платформу с более чем 750 инстансами и набором новейших процессоров, хранилищ, сетей, операционных систем и моделей покупок, обеспечивая должное соответствие нуждам конкретной рабочей нагрузки. AWS EC2 – первый крупный облачный провайдер, который поддерживает работу процессоров Intel, AMD и Arm, единственное облако с инстансами EC2 Mac по требованию и с сетью Ethernet 400 Гбит/с. AWS EC2 предлагает лучшее соотношение цены и производительности машинного обучения, а также самую низкую стоимость инстансов логических выводов в облаке. На AWS выполняется больше рабочих нагрузок SAP, высокопроизводительных вычислений (HPC), машинного обучения и Windows, чем в любом другом облаке.

На следующей схеме (рисунок 3.4) показана базовая архитектура экземпляра Amazon EC2, развернутого в виртуальном частном облаке Amazon (VPC). В этом примере экземпляр EC2 находится в зоне доступности региона. Экземпляр EC2 защищен с помощью группы безопасности, которая представляет собой виртуальный брандмауэр, контролирующий входящий и исходящий трафик. Закрытый ключ хранится на локальном компьютере, а открытый ключ – на экземпляре. Оба ключа указываются как ключевая пара для подтверждения личности пользователя. В этом сценарии за экземпляром закреплена том Amazon EBS. VPC взаимодействует с Интернетом с помощью интернет-шлюза.

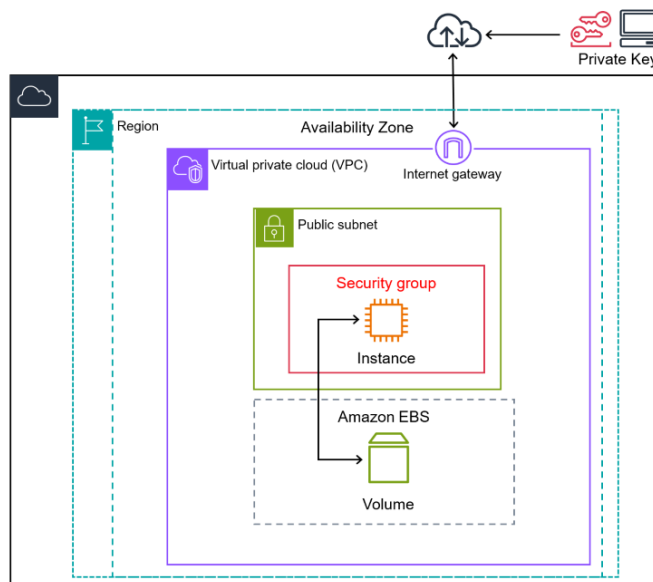


Рисунок 3.4 – Базовая архитектура экземпляра AWS EC2 [12]

Amazon Elastic Compute Cloud (Amazon EC2) предоставляет масштабируемые вычислительные мощности по требованию в облаке Amazon Web Services (AWS). Использование Amazon EC2 позволяет сократить расходы на аппаратное обеспечение и ускорить разработку. С помощью Amazon EC2 можно запустить столько виртуальных серверов, сколько нужно, настроить безопасность и сетевое взаимодействие, а также управлять хранилищем. Помимо этого, можно увеличить мощность (масштабирование) для решения задач, требующих больших вычислений, таких как ежемесячные или ежегодные процессы или скачки посещаемости веб-сервиса. При снижении нагрузки можно снова сократить мощность (уменьшить масштаб).

Amazon EC2 предоставляет следующие возможности высокого уровня:

- инстансы (instances): виртуальные серверы;
- образы машин Amazon (Amazon Machine Images): предварительно настроенные шаблоны для экземпляров, в которых собраны компоненты, необходимые для сервера (включая операционную систему и дополнительное программное обеспечение);
- типы экземпляров (Instances types): различные конфигурации процессора, памяти, хранилища, сетевых мощностей и графического оборудования для экземпляров;
- пары ключей (Key pairs): защищенная информация для входа в систему для экземпляров, AWS хранит открытый ключ, а пользователь хранит закрытый ключ в безопасном месте;

- тома хранилища экземпляров (Instance store volumes): тома для хранения временных данных, которые удаляются при остановке, спящем режиме или завершении работы экземпляра;
- тома Amazon EBS: постоянные тома для хранения данных с помощью Amazon Elastic Block Store (Amazon EBS);
- регионы и зоны (Regions and Zones): несколько физических местоположений для ресурсов, таких как экземпляры и тома Amazon EBS;
- группы безопасности (security groups): виртуальный брандмауэр, позволяющий указать протоколы, порты и диапазоны IP-адресов источников, через которые могут подключаться экземпляры, а также диапазоны IP-адресов получателей, к которым могут подключаться экземпляры.
- эластичные IP-адреса (elastic ip addresses): статические IPv4-адреса для динамических облачных вычислений.
- теги (tags): метаданные, которые можно создавать и назначать ресурсам Amazon EC2;
- виртуальные частные облака (Virtual Private Clouds): виртуальные сети, логически изолированные от остальной части облака AWS.

Выбор использования EC2 в контексте дипломного проекта обоснован необходимостью предоставления инфраструктурных ресурсов для развертывания и запуска веб-сервиса. EC2 обеспечивает гибкость в выборе типа и размера экземпляров, что позволяет оптимизировать использование ресурсов и обеспечить соответствие требованиям проекта в плане производительности и масштабируемости.

3.2.8 Amazon S3

Amazon Simple Storage Service (Amazon S3) – это распределенный веб-сервис хранения объектов, предлагающий лучшие в отрасли показатели производительности, масштабируемости, доступности и безопасности данных. Клиенты любой величины и из любой промышленной отрасли могут хранить и защищать необходимый объем данных для практически любого примера использования. Например, для пользовательских данных, облачных веб-сервисов и мобильных программных средств. Выгодные классы хранилища и простые в использовании инструменты администрирования позволяют оптимизировать затраты, организовать данные и точно настроить ограничения доступа в соответствии с потребностями бизнеса или законодательными требованиями (рисунок 3.5).



Рисунок 3.5 – Принцип работы Amazon S3 [13]

Amazon S3 используют для хранения большого объема данных, аналитики, искусственного интеллекта, машинного обучения, высокопроизводительных вычислений, резервного копирования и восстановления критических важных файлов, запуска веб-сервисов с оптимизацией для облака. Помимо этого, стоимость архивации данных составляет наименьшую цену.

3.2.9 RDS

Amazon RDS – это управляемый сервис реляционных баз данных для MySQL, PostgreSQL, MariaDB, Oracle (с поддержкой собственных лицензий) или SQL Server.

Amazon RDS представляет собой сервис реляционных баз данных, который призван облегчить управление базами данных и оптимизировать совокупную стоимость владения. Этот сервис обладает простотой в управлении, что позволяет легко настраивать, эксплуатировать и масштабировать его в зависимости от потребностей клиента. Amazon RDS автоматизирует множество рутинных задач управления базами данных, включая выделение ресурсов, настройку, выполнение резервного копирования и установку исправлений.

Клиентам предоставляется возможность создавать новую базу данных всего за несколько минут и гибко настраивать базу данных в соответствии с нужными потребностями. Сервис предлагает широкий выбор движков баз данных и вариантов развертывания, что позволяет клиентам оптимизировать производительность в соответствии с их требованиями. Оптимизированные операции записи и чтения, множество зон доступности с резервными инстансами и возможность выбора из различных вариантов ценообразования

делают Amazon RDS привлекательным выбором для эффективного управления базами данных и управления затратами.

AWS предоставляет самый широкий выбор специализированных баз данных, которые позволяют вам экономить, развиваться и внедрять инновации быстрее. Выбор состоит из более чем 15 специализированных моделей баз данных, например реляционную, документную, графовую, реестровую, а также модель базы данных на основе пар «ключ-значение», базы данных в памяти, базы данных с широким столбцом и базы данных временных рядов.

Производительность при любом масштабе – реляционные базы данных, которые в 3–5 раз быстрее, чем популярные альтернативы, или нереляционные базы данных, обеспечивающие минимальную задержку в микро- или миллисекунды.

Принцип работы Amazon RDS представлен на рисунке 3.6.

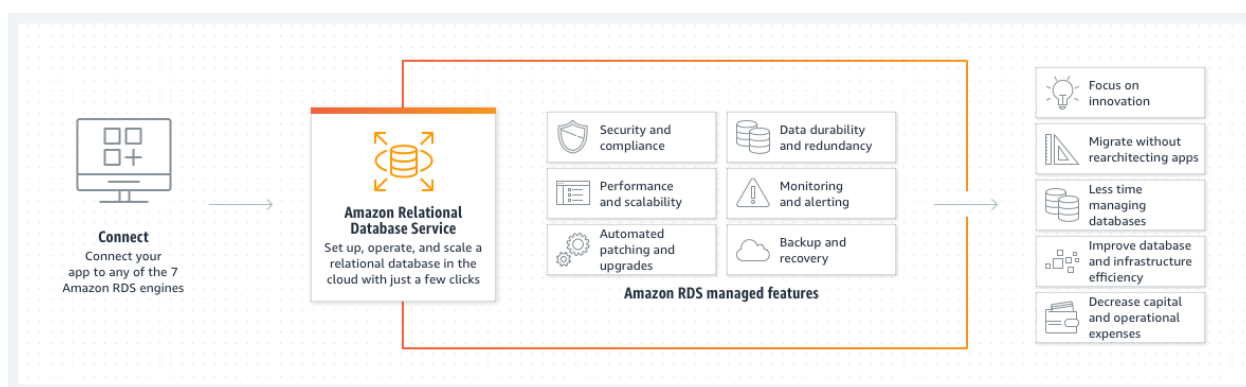


Рисунок 3.6 – Принцип работы Amazon RDS [14]

Выбор распределенного веб-сервиса Amazon RDS обоснован несколькими факторами, включая требования по безопасности и надежности, а также необходимость эффективного взаимодействия с развертываемым веб-сервисом. Размещение Amazon RDS в Private Subnet, в отличие от Public Subnet, обеспечивает дополнительный слой защиты для базы данных, ограничивая доступ к ней извне и минимизируя риски уязвимостей безопасности.

Использование Amazon RDS в качестве виртуальной машины с реляционной базой данных MySQL 5.7 обусловлено требованиями к поддержке конкретной версии базы данных и ее функциональности для развертываемого веб-сервиса. MySQL 5.7 предоставляет широкие возможности по управлению данными и обеспечивает совместимость с многим программным окружением и инструментами разработки.

Также стоит отметить, что Amazon RDS предоставляет высокий уровень автоматизации и управления базами данных, что позволяет уменьшить время и усилия, затрачиваемые на настройку и поддержку инфраструктуры базы данных. Это включает в себя автоматическое выделение ресурсов, резервное копирование и установку исправлений, что способствует оптимизации процесса развертывания и обеспечивает стабильную работу развертываемого веб-сервиса.

3.2.10 Elastic IP

Elastic IP address является ключевым элементом в инфраструктуре облачных вычислений, обеспечивая статический IPv4-адрес, который динамически назначается различным облачным ресурсам. Это предоставляет гибкость и надежность в управлении сетевой конфигурацией, позволяя пользователям AWS сохранить постоянную точку доступа к своим облачным вычислениям. Эластичный IP-адрес является уникальным для каждой учетной записи AWS и остается в ее распоряжении до тех пор, пока не будет освобожден, обеспечивая постоянство в сетевых настройках в течение всего времени использования.

Использование Elastic IP address позволяет эффективно управлять отказами экземпляров или программного обеспечения, предоставляя возможность быстрого переназначения адреса на другой экземпляр в рамках той же учетной записи AWS. Это существенно сокращает время простоя и обеспечивает непрерывную доступность облачных ресурсов. Кроме того, Elastic IP address легко интегрируется в DNS-записи для доменов, обеспечивая удобный механизм указания доменного имени на экземпляре в облаке AWS.

Использование Elastic IP (EIP) предоставляет ряд значительных преимуществ для облачной инфраструктуры. Первое преимущество заключается в стабильности и надежности, поскольку EIP обеспечивает постоянный IP-адрес, который остается неизменным при перезапуске экземпляров EC2 или других виртуальных ресурсов. Это обеспечивает стабильный доступ к веб-сервису, что является критическим для непрерывной работы системы.

Второе преимущество связано с управлением и масштабируемостью. EIP позволяет гибко управлять IP-адресами и привязывать их к различным экземплярам EC2 и другим ресурсам в облаке AWS. Такая гибкость обеспечивает удобство при настройке сетевой инфраструктуры и позволяет эффективно масштабировать веб-сервис в соответствии с потребностями.

Третье преимущество состоит в предоставлении бесплатной зоны переноса для EIP со стороны AWS. Это позволяет безопасно и эффективно

переносить IP-адреса между различными экземплярами EC2 и регионами AWS, что является важным аспектом обеспечения гибкости и доступности при работе с облачной инфраструктурой.

Четвертое преимущество заключается в обеспечении безопасности. Использование EIP позволяет избежать блокировки IP-адресов в списке антиспам-фильтров или других систем безопасности, так как EIP остается постоянным и надежным, что повышает уровень защиты и предотвращает потенциальные проблемы с безопасностью в сети.

Выбор использования Elastic IP для распределенного веб-сервиса на AWS обоснован несколькими факторами, включая необходимость обеспечения безопасного и надежного соединения с постоянным доменным именем. При развертывании веб-сервиса в облаке AWS часто требуется использование SSL-сертификата для защищенной передачи данных между клиентами и сервером. Важно, чтобы веб-сервис обладал постоянным доменным именем, что обеспечивает удобство для пользователей, позволяя им легко идентифицировать и запомнить адрес сайта.

Поскольку при создании нового экземпляра EC2 каждый раз выдается новый Public IPv4-адрес, использование вручную созданного и постоянного Elastic IP address является эффективным решением. Это позволяет назначить один и тот же постоянный IPv4-адрес каждый раз при создании нового экземпляра EC2 через Terraform. Такой подход обеспечивает постоянство IP-адреса, что необходимо для создания А-записи в DNS домена на Cloudflare, обеспечивая постоянное доменное имя для веб-сервиса.

Таким образом, использование Elastic IP в данном контексте позволяет обеспечить надежную и устойчивую работу веб-сервиса, обеспечивая постоянное соединение с постоянным доменным именем, что повышает удобство использования и безопасность сервиса для его пользователей.

3.2.11 IAM Policy

С помощью IAM Policy можно управлять доступом в AWS, создавая политики и прикрепляя их к IAM-идентификаторам (пользователям, группам пользователей или ролям) или ресурсам AWS. Политика – это объект в AWS, который, будучи связанным с идентификатором или ресурсом, определяет их разрешения. AWS оценивает эти политики, когда принципал IAM (пользователь или роль) делает запрос. Разрешения в политиках определяют, будет ли запрос разрешен или отклонен. Большинство политик хранятся в AWS в виде документов JSON. AWS поддерживает шесть типов политик: политики на основе идентификации, политики на основе ресурсов, границы разрешений, организационные SCP, ACL и политики сессий.

Политики IAM определяют разрешения на действие независимо от метода, который использует пользователь для выполнения операции.

Например, если политика разрешает действие `GetUser`, то пользователь с такой политикой может получить информацию о пользователе из AWS Management Console, AWS CLI или AWS API.

При создании пользователя IAM предоставляется возможность выбора типа доступа – консольного или программного. Если разрешен доступ к консоли, пользователь IAM имеет права доступа войти в консоль, используя предоставленные учетные данные. В случае разрешенного программного доступа пользователь имеет права доступа использовать ключи доступа для работы с CLI или API. Эти два варианта предоставления доступа представляют собой различные методы взаимодействия с облачными ресурсами и предназначены для разных сценариев использования.

Распределенный веб-сервис IAM Policy на AWS (Identity and Access Management) играет важную роль в управлении доступом к ресурсам и сервисам облака. Выбор использования IAM Policy обусловлен необходимостью реализации распределенного веб-сервиса CloudWatch для мониторинга логирования, где необходима IAM Policy.

Основные преимущества использования IAM Policy:

1 Гранулированное управление доступом: IAM Policy позволяет создавать гранулированные политики доступа, определяющие, какие пользователи или роли могут выполнять какие операции над определенными ресурсами. Это обеспечивает принцип минимальных привилегий и улучшает безопасность инфраструктуры.

2 Гибкость и масштабируемость: IAM Policy позволяет настраивать политики доступа для различных типов ресурсов, включая EC2, S3, RDS и другие, а также для различных типов действий, таких как чтение, запись, удаление и т. д. Это обеспечивает гибкость в настройке доступа и масштабируемость при добавлении новых ресурсов и пользователей.

3 Интеграция с другими сервисами AWS: IAM Policy интегрируется с другими сервисами AWS, такими как S3, EC2, RDS и другими, обеспечивая возможность управления доступом к различным ресурсам и сервисам из единого интерфейса.

4 Мониторинг и аудит доступа: IAM Policy позволяет отслеживать и анализировать действия пользователей и ролей в рамках вашей учетной записи AWS, обеспечивая возможность мониторинга и аудита доступа для обеспечения безопасности и соответствия требованиям.

Использование IAM Policy в контексте дипломного проекта обеспечивает необходимый уровень безопасности и контроля доступа к ресурсам и сервисам облака AWS. Помимо этого, использование IAM Policy

необходимо для реализации распределенного веб-сервиса CloudWatch. Это позволяет эффективно управлять пользователями, ролями и политиками безопасности, обеспечивая соблюдение правил доступа и минимизацию рисков безопасности.

3.2.12 IAM Role

IAM Role – это идентификатор в IAM, который создается в учетной записи и обладает определенными разрешениями. Этот вид идентификатора похож на IAM User в том смысле, что также обладает политиками разрешений, определяющими его возможности в AWS. В отличие от IAM User, роль не привязана к конкретному пользователю, а предназначена для использования любым лицом. Кроме того, роль не обладает постоянными учетными данными, такими как пароль или ключи доступа. Вместо этого, при принятии роли пользователем предоставляются временные учетные данные безопасности для сеанса работы с данной ролью.

Роли могут быть использованы для передачи доступа к ресурсам AWS пользователям, веб-сервисам или службам, которые обычно не имеют прямого доступа к ним. Например, это включает предоставление доступа пользователям учетной записи AWS к ресурсам, либо предоставление доступа пользователям из одной учетной записи AWS к ресурсам из другой учетной записи. Также роли могут быть использованы для разрешения мобильным программным средствам использовать ресурсы AWS, избегая встраивания ключей AWS в исходный код программного средства.

Выбор использования распределенного веб-сервиса IAM Role на AWS обусловлен необходимостью обеспечения безопасного и гибкого управления доступом к ресурсам облака, особенно в контексте среды с множеством различных сервисов. IAM Role представляет собой эффективный механизм для предоставления временных учетных записей с ограниченными привилегиями, которые могут использоваться в определенные временные интервалы или автоматически для выполнения конкретных задач или операций.

В случае реализации сервиса логирования CloudWatch в инфраструктуре разворачиваемого веб-сервиса, необходимо реализовать соответствующие IAM Policy для обеспечения необходимых разрешений. После этого IAM Policy ассоциируется с созданной до этого IAM Role. Создание IAM Role позволяет эффективно управлять доступом к ресурсам облака и делегировать необходимые привилегии на выполнение операций с логами CloudWatch. Кроме того, использование IAM Role обеспечивает высокий уровень

безопасности за счет предоставления временных учетных записей, что снижает риск компрометации учетных данных и распространения привилегий.

Далее, на основе созданной IAM Role и связанной с ней IAM Policy, необходимо формировать IAM Instance Profile, который затем прикрепляется к создаваемому в инфраструктуре EC2. Этот шаг обеспечивает автоматическое предоставление необходимых привилегий экземпляру EC2 для выполнения операций с логами CloudWatch без необходимости явного указания учетных данных в конфигурации или в коде разрабатываемого программного средства и (или) веб-сервиса. Такой метод управления доступом упрощает процесс администрирования и повышает общий уровень безопасности инфраструктуры за счет сокращения возможных точек уязвимости и рисков безопасности.

3.2.13 CloudWatch

Amazon CloudWatch – это служба, которая отслеживает работу веб-сервиса, реагирует на изменения производительности, оптимизирует использование ресурсов и предоставляет информацию о состоянии операционной системы. Собирая данные по всем ресурсам AWS, CloudWatch обеспечивает видимость производительности всей системы и позволяет пользователям устанавливать сигналы тревоги, автоматически реагировать на изменения и получать единое представление о работоспособности системы (рисунок 3.7).

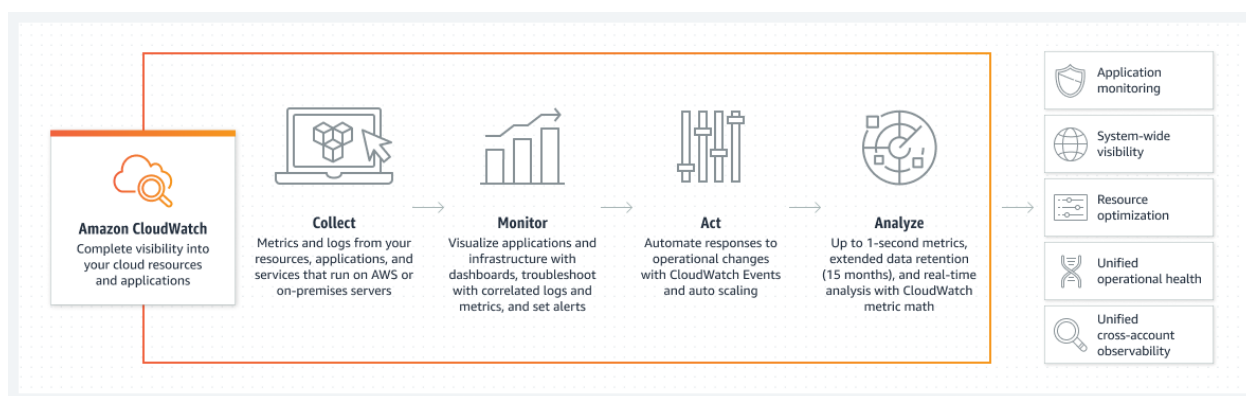


Рисунок 3.7 – Принцип работы Amazon CloudWatch [15]

Amazon CloudWatch собирает и визуализирует журналы, метрики и данные о событиях в реальном времени в виде автоматизированных панелей, чтобы упростить обслуживание инфраструктуры и веб-сервисов.

Выбор использования распределенного веб-сервиса CloudWatch на AWS обусловлен потребностью в мониторинге, анализе и управлении ресурсами облака для обеспечения их надежной и эффективной работы. CloudWatch предоставляет широкий спектр инструментов для сбора, отображения и анализа метрик, журналов и событий, что позволяет оперативно реагировать на изменения и проблемы в инфраструктуре.

Выбор использования CloudWatch обоснован несколькими факторами.

Во-первых, сервис обеспечивает мониторинг и анализ метрик для различных ресурсов облака, что позволяет оперативно отслеживать производительность и использование ресурсов, а также оптимизировать инфраструктуру.

Во-вторых, CloudWatch предоставляет возможность мониторинга журналов и событий, что позволяет выявлять проблемы и нештатные ситуации для оперативной реакции на них. Кроме того, сервис предоставляет инструменты для управления и оптимизации ресурсов, таких как автоматическое масштабирование или использование правил мониторинга для определенных событий.

Наконец, интеграция CloudWatch с другими сервисами AWS обеспечивает возможность автоматизации процессов мониторинга и управления ресурсами. Таким образом, использование CloudWatch обеспечивает надежный и эффективный мониторинг, анализ и управление ресурсами облака, что делает его важным компонентом для обеспечения безопасности, надежности и эффективности работы в облачной среде.

3.3 Контейнеризация и оркестрация с помощью Docker и Docker Compose

Docker представляет собой открытую платформу, предназначенную для разработки, доставки и запуска собранных образов. Его основное преимущество заключается в возможности изолировать контейнер от инфраструктуры, что обеспечивает быструю поставку программного обеспечения.

Основой Docker являются контейнеры, которые представляют собой слабо изолированную среду для упаковки и запуска программных компонентов. Контейнеры обеспечивают изоляцию и безопасность, что позволяет запускать множество контейнеров одновременно на одном хосте. Более того, контейнеры могут легко обмениваться во время выполнения, гарантируя, что все пользователи получают одинаковый контейнер, работающий в точности так же.

Docker предоставляет инструменты и платформу для управления жизненным циклом контейнеров. Разработчики могут создавать образы и связанные компоненты с помощью контейнеров, которые затем используются для распространения и тестирования программного окружения. После этого, собранный образ готово проекта развертывается в производственной среде в виде контейнера или оркестрированной службы, независимо от того, где находится производственная среда – на локальном сервере, в облачном окружении или в гибридной среде. Контейнеры также удобны для использования в рабочих процессах непрерывной интеграции и непрерывной доставки (CI/CD).

С помощью Docker также описывается процесс разработки программного обеспечения. Предположим, что есть следующий сценарий:

- 1 Разработчики создают код локально и обмениваются им с коллегами через контейнеры Docker.

- 2 Затем наработки кода передаются в тестовую среду с помощью Docker, где запускаются как автоматические, так и ручные тесты.

- 3 В случае обнаружения ошибок в процессе тестирования, исправления вносятся разработчиками в локальной среде, а затем переносятся в тестовую среду для проверки и утверждения.

- 4 После завершения тестирования обновленный код легко развертывается в производственной среде, что упрощает доставку исправлений клиентам.

Использование Docker позволяет эффективно использовать оборудование, так как Docker обеспечивает высокую плотность размещения и является эффективной альтернативой виртуальным машинам на базе гипервизора. Это особенно полезно для малых и средних развертываний, где требуется максимальное использование ресурсов.

Архитектура Docker основана на клиент-серверном принципе. Клиентский компонент Docker взаимодействует с демоном Docker, который отвечает за создание, запуск и управление контейнерами Docker. Присутствует возможность использования как локального соединения клиента Docker с демоном Docker на той же системе, так и подключение клиента Docker к удаленному демону Docker. Для обмена данными между клиентом Docker и демоном Docker используются REST API, сокеты UNIX или сетевой интерфейс (рисунок 3.8).

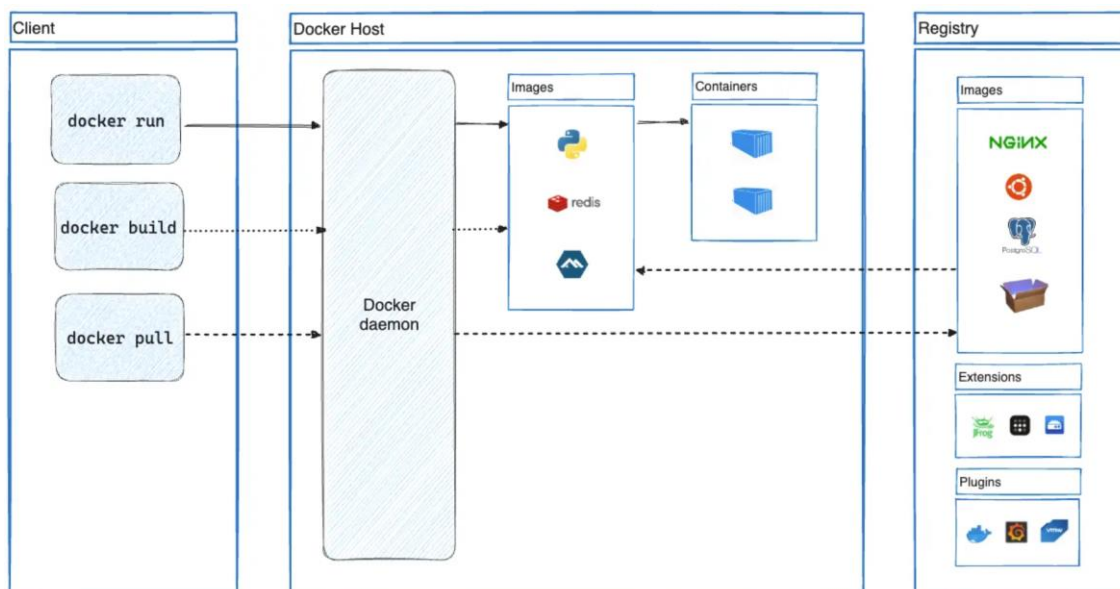


Рисунок 3.8 – Архитектура Docker [16]

Демон Docker (dockerd) прослушивает запросы API Docker и управляет объектами Docker, такими как образы, контейнеры, сети и тома. Демон взаимодействует с другими демонами для управления службами Docker.

Клиент Docker, известный как docker, представляет собой основной интерфейс для взаимодействия множества пользователей с Docker. Когда выполняются команды, такие как `docker run`, клиент передает их на исполнение демону Docker (dockerd). Этот процесс осуществляется с использованием API Docker. Клиент Docker может взаимодействовать с несколькими демонами.

Образ (Image) в Docker – это шаблон, который доступен только для чтения и содержит инструкции по созданию контейнера Docker. Зачастую образ основан на другом образе, к которому добавлены определенные дополнительные настройки. Например, если создать образ, основанный на образе `ubuntu`, с установленным веб-сервером Apache и статическим веб-сервисом, а также необходимыми настройками конфигурации для его функционирования.

Кроме того, Docker позволяет создавать собственные образы или использовать те, которые были созданы другими и опубликованы в реестре. Для создания собственного образа необходимо создать Dockerfile с простым синтаксисом, определяющим этапы, необходимые для создания и запуска образа. Каждая инструкция в Dockerfile создает слой в образе, который содержит определенный объем информации (например, `RUN`, `COPY`, `ADD`, `CMD`).

Когда происходит изменение Dockerfile и пересобирание образа, пересобираются только те слои, которые изменились. Именно это делает образы такими легкими, маленькими и быстрыми по сравнению с другими технологиями виртуализации.

Контейнер в Docker представляет собой запускаемый экземпляр образа. Создание, запуск, остановка, перемещение или удаление контейнера осуществляется с помощью Docker API или CLI. Контейнер можно подключить к одной или нескольким сетям, присоединить к нему хранилище или создать новый образ на основе его текущего состояния.

По умолчанию контейнер хорошо изолирован от других контейнеров и хост-машины. Управление степенью изоляции сети, хранилища и других базовых подсистем контейнера от других контейнеров или хост-машины возможно.

Контейнер определяется образом и параметрами конфигурации, предоставленными при его создании или запуске. При удалении контейнера все изменения его состояния, не сохраненные в постоянном хранилище, удаляются.

Docker разработан на языке программирования Go и использует возможности ядра Linux для обеспечения своей функциональности. Применение технологии пространств имен позволяет Docker создавать изолированные рабочие пространства, называемые контейнерами. При запуске контейнера Docker формирует набор пространств имен для его работы.

Docker Compose – это инструмент для определения и запуска многоконтейнерных проектов. Это ключ к оптимизации и повышению эффективности разработки и развертывания.

Compose упрощает управление всем стеком проекта, обеспечивая простое управление сервисами, сетями и томами в едином конфигурационном файле YAML. После создания этого файла, пользователь может запустить все службы с помощью одной команды.

Compose поддерживает работу в различных средах, включая продакшн, стейджинг, разработку, тестирование и процессы CI. Помимо этого, Compose предоставляет команды для управления жизненным циклом программного обеспечения или средства, включая запуск, остановку, восстановление служб, просмотр состояния запущенных служб, потоковый вывод журнала запущенных служб и выполнение разовых команд для службы.

Docker Compose основан на конфигурационном файле YAML, который часто называется `compose.yaml` или `docker-compose.yml`.

Модель Docker Compose. Компоненты проекта определяются как сервисы. Пример реализации `docker-compose.yml`:

```

version: '3.8'

services:
  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - wordpress:/var/www/html

  db:
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
    volumes:
      - db:/var/lib/mysql

volumes:
  wordpress:
  db:

```

Сервис – это абстрактная концепция, реализуемая на платформах путем запуска одного и того же образа контейнера и его конфигурации один или несколько раз.

Сервисы взаимодействуют друг с другом через сети. В спецификации Compose сеть представляет собой абстракцию возможностей платформы, позволяющую создавать IP-маршруты между контейнерами внутри сервисов, соединенных между собой.

Для хранения и обмена постоянными данными между сервисами используются тома. Спецификация описывает такие постоянные данные как высокоуровневое монтирование файловой системы с глобальными опциями.

Некоторые сервисы требуют конфигурационных данных, которые зависят от времени выполнения или платформы. Для этого в спецификации определена специальная концепция configs. Конфигурации можно сравнить с томами, поскольку тома представляют собой файлы, монтируемые в контейнер. Однако фактическое определение включает в себя отдельные ресурсы платформы и сервисы, которые абстрагируются этим типом.

Секрет – это особый вид конфигурационных данных для конфиденциальных данных, которые не должны быть открыты без учета соображений безопасности. Секреты предоставляются сервисам в виде файлов, монтируемых в их контейнеры, но ресурсы платформы для предоставления конфиденциальных данных достаточно специфичны, чтобы заслужить отдельное понятие и определение в спецификации Compose.

Проект – это отдельное развертывание спецификации готово проекта на платформе. Имя проекта, задаваемое с помощью атрибута `name` верхнего уровня, используется для объединения ресурсов в группы и изоляции их от других сервисов или других установок того же сервиса, специфицированного Compose, с различными параметрами.

Compose предлагает возможность задать пользовательское имя проекта и переопределить его, чтобы один и тот же файл `compose.yaml` можно было развернуть дважды на одной и той же инфраструктуре без изменений, просто передав другое имя. Пример присваивания имени в запускаемом `docker-compose.yml` (с помощью флага `-p`):

```
docker-compose -p PROJECT_NAME_BSUIR up -d
```

Выбор инструментов контейнеризации и оркестрации для реализации DevOps-технологий поддержки распределенных веб-сервисов на базе AWS с использованием Terraform обоснован рядом факторов.

Docker и Docker Compose являются широко используемыми инструментами в индустрии разработки программного обеспечения. Данные инструменты обеспечивают возможность контейнеризации готовых или разрабатываемых проектов, что обеспечивает консистентность окружений разработки, тестирования и производства.

3.4 Описание и обоснование использования CI/CD

Непрерывная интеграция (Continuous Integration, CI) и непрерывная доставка (Continuous Delivery, CD) представляют собой методологию, основанную на культуре, наборе принципов и практиках, обеспечивающих более частое и надежное развертывание изменений в программном обеспечении.

CI/CD являются ключевыми практиками в рамках DevOps и также относятся к agile-подходам. Автоматизация процесса развертывания позволяет разработчикам сосредоточиться на реализации бизнес-требований, а также на качестве кода и безопасности.

Непрерывная интеграция и непрерывная поставка являются составными частями более обширной методологии DevOps. CI и CD взаимодействуют между собой, стремясь к устранению сложностей, возникающих в процессе непрерывных инноваций. Эти два процесса тесно взаимосвязаны и совместно обеспечивают методологию DevOps.

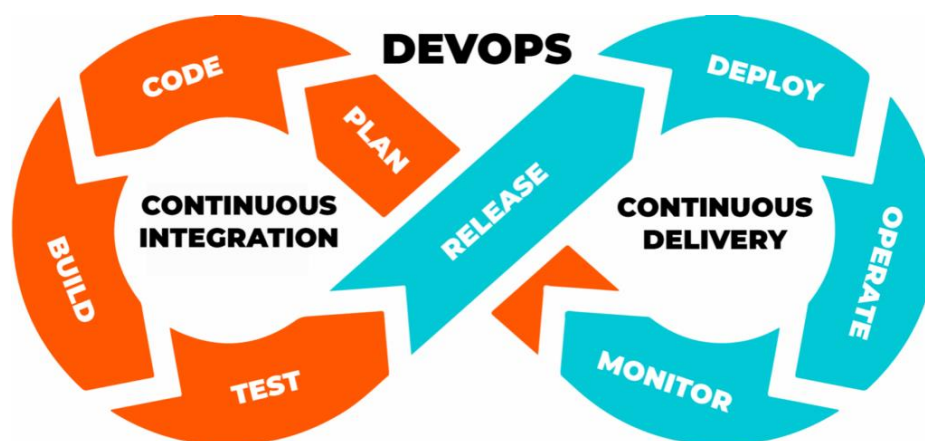


Рисунок 3.9 – DevOps и CI/CD [17]

Непрерывная интеграция (Continuous integration, CI) – это процесс разработки, заключающийся в автоматической сборке и выполнении модульных тестов после внесения изменений в исходный код. CI требует от команд разработчиков ежедневно по несколько раз интегрировать изменения кода в общий репозиторий исходного кода.

Основная цель непрерывной интеграции – создать последовательный, постоянный метод автоматической сборки и тестирования программного окружения или готового проекта, гарантирующий, что изменения, внесенные одним разработчиком, пригодны для использования во всей кодовой базе.

Непрерывная доставка (CD) – это продолжение непрерывной интеграции. Это процесс, в котором команды DevOps разрабатывают и поставляют полные части программного обеспечения в репозиторий – например, GitHub или реестр контейнеров – короткими, контролируемыми циклами. Непрерывная доставка делает релизы регулярными и предсказуемыми событиями для сотрудников DevOps и незаметными для конечных пользователей.

CI рассматривается как первый шаг, а CD – как второй для создания и развертывания кода. CI – это скорее подготовка кода к выпуску (сборка/тестирование), а CD – собственно выпуск кода (выпуск/развертывание).

Таким образом, в контексте данного дипломного проекта, применение непрерывной интеграции и непрерывной поставки (CI/CD) обосновано сразу несколькими аспектами.

Во-первых, развертывание и обновление изменений веб-сервиса с использованием распределенных веб-сервисов на платформе AWS требует некоторой степени автоматизации и контроля процесса. CI/CD позволяет автоматизировать сборку и развертывания веб-сервиса, обеспечивая быстрое и надежное внедрение изменений в инфраструктуру. Это особенно важно для эффективного управления проектом.

Во-вторых, в контексте DevOps методологии, внедрение CI/CD способствует ускорению цикла разработки, улучшению качества и уменьшению рисков. CI/CD позволяет разработчикам чаще и безопаснее вносить изменения в код, а также проводить тестирование и развертывание в автоматизированном режиме. Это особенно важно для проектов, где высокая скорость разработки и поставки необходима для конкурентного преимущества, именно поэтому это является технологией в мире DevOps.

Частота использования CI/CD зависит от конкретного проекта и его требований. Однако, в современной разработке программного обеспечения, CI/CD является стандартной практикой, которая широко применяется в индустрии. Это подтверждается популярностью инструментов CI/CD, таких как Jenkins, CircleCI, GitLab CI/CD и других, а также активным обсуждением этой темы в сообществе разработчиков и специалистов по DevOps.

Исходя из описания, изучения и анализа практики DevOps, применение CI/CD оправдано как с технической, так и с методологической точек зрения, и соответствует современным требованиям и практикам разработки и развертывания программного обеспечения.

3.5 Проектирование облачной инфраструктуры

Для инициирования процесса проектирования облачной инфраструктуры для веб-сервиса на платформе AWS предварительным этапом является загрузка значков архитектуры AWS с официального веб-ресурса. Этот шаг необходим для обеспечения ясного понимания распределения веб-сервисов AWS, их структуры и взаимосвязей. Визуальное представление архитектурных элементов и компонентов позволяет осуществить более эффективное планирование и развертывание инфраструктуры [18].

Следующим этапом является анализ требований веб-сервиса и его функциональных характеристик. На основе этого анализа определяются необходимые компоненты и сервисы AWS, которые будут использоваться в

инфраструктуре. Рассматриваются такие аспекты, как масштабируемость, доступность, безопасность и производительность.

После этого производится проектирование архитектуры облачной инфраструктуры, включающее в себя выбор подходящих сервисов AWS и определение их конфигурации. Распределение ресурсов, управление трафиком, резервное копирование данных и мониторинг производительности являются ключевыми аспектами этого этапа.

Далее осуществляется развертывание инфраструктуры на платформе AWS согласно спроектированной архитектуре. Процесс развертывания включает в себя создание необходимых ресурсов, настройку параметров и интеграцию компонентов с помощью Terraform.

После развертывания инфраструктуры производится ее тестирование для проверки работоспособности и соответствия требованиям. В случае обнаружения ошибок или несоответствий производится их исправление и повторное тестирование.

Наконец, после успешного завершения всех предыдущих этапов обеспечивается поддержка и мониторинг облачной инфраструктуры.

Спроектированная облачная инфраструктура представлена на рисунке 3.10.

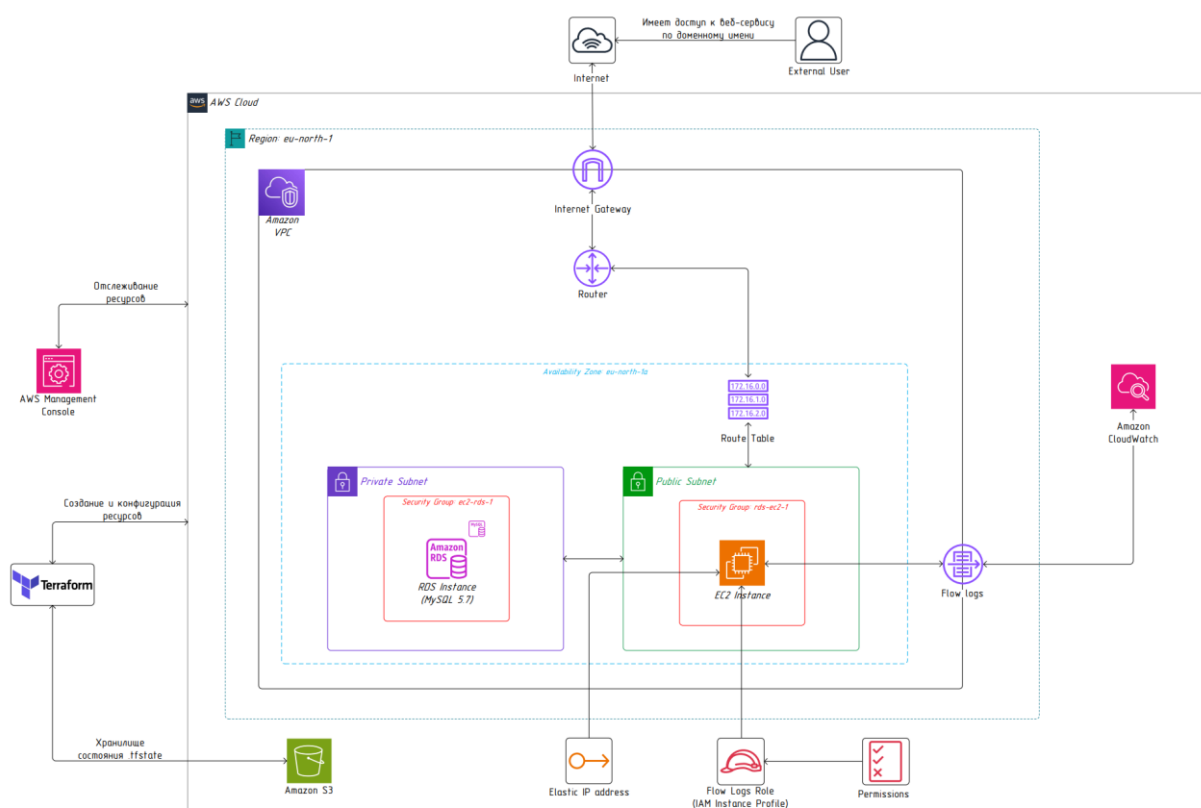


Рисунок 3.10 – Спроектированная облачная инфраструктура

AWS Cloud служит в качестве облачного провайдера, обеспечивающего доступность и масштабируемость. Выбранный регион – eu-north-1 – является местом размещения всех ресурсов инфраструктуры. В рамках этого региона создается Amazon VPC, основной элемент сетевой инфраструктуры, который обеспечивает изоляцию и безопасность ресурсов.

В состав Amazon VPC входят Internet Gateway и маршрутизатор, обеспечивающие подключение к внешнему интернету. VPC также содержит несколько зон доступности (availability zones), в данном случае eu-north-1a. В каждой зоне доступности находится Route Table и две подсети (Subnet) - Public и Private.

В Private Subnet размещается экземпляр базы данных RDS, защищенный Security Group (ec2-rds-1), который ограничивает доступ только из локальной сети Public Subnet для повышения безопасности. В Public Subnet развертывается экземпляр EC2, который обеспечивает взаимодействие с внешними клиентами и сервисами.

Для обеспечения постоянной доступности веб-сервиса, EC2 Instance привязывается к статическому IP-адресу с помощью Elastic IP, который создается вручную и ассоциируется с экземпляром при помощи Terraform.

Дополнительно в регионе настраивается IAM Policy для определения прав доступа и IAM Role для CloudWatch, обеспечивающие мониторинг и логирование веб-сервиса. IAM Instance Profile, связанный с EC2 Instance, обеспечивает безопасный доступ к другим AWS-ресурсам.

В итоге, пользователи имеют доступ к веб-сервису через интернет, а EC2 Instance безопасно соединяется с базой данных RDS. Создание и управление ресурсами осуществляется с помощью Terraform, а отслеживание производится через AWS Management Console. Файл .tfstate хранится в Amazon S3 в качестве надежного хранилища состояний.

Данный раздел служил основой для разработки графического материала под названием «Схема инфраструктуры AWS», который прилагается к дипломному проекту в качестве Приложения Б. Схема представляет собой визуализацию архитектуры облачной инфраструктуры, развернутой на платформе Amazon Web Services (AWS). Данный графический материал отражает компоненты инфраструктуры, их взаимосвязи и включает в себя описание ключевых сервисов и ресурсов, задействованных в проекте. Представленный графический материал служит для более наглядного представления инфраструктуры проекта, а также для обеспечения более глубокого понимания его структуры и компонентов.

4 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ОБЛАЧНОЙ ИНФРАСТРУКТУРЫ ДЛЯ ВЕБ-СЕРВИСА

4.1 Обзор разворачиваемого веб-сервиса и используемых библиотек

Веб-сервис WordPress представляет собой популярную платформу для создания и управления веб-сервисами и блогами. WordPress снован на языке программирования PHP и использует базу данных MySQL для хранения контента. WordPress предоставляет широкий набор функциональных возможностей, включая возможность создания и редактирования страниц и постов, управление пользователями и правами доступа, а также расширение функциональности с помощью плагинов и тем.

WordPress остается одной из самых популярных платформ CMS (Content Management System) для создания веб-сервисов, используемой более чем на 43% всех существующих веб-ресурсов. Однако, в отличие от некоторых других конструкторов веб-сервисов, WordPress не является универсальным решением. Эта платформа представляет собой систему управления контентом, требующую от пользователя некоторого уровня технических знаний для управления различными аспектами веб-сервиса, такими как хостинг и безопасность.

Научные данные и статистика о WordPress являются важным инструментом для понимания его роли и влияния в мире веб-разработки и онлайн-сообщества. Согласно последним исследованиям, проведенным в 2021 году, WordPress остается доминирующей платформой среди CMS, управляющей значительной частью интернет-ресурсов. Более конкретно, WordPress используется для более чем 43% всех сайтов в мире, подчеркивая его широкое распространение и популярность среди веб-разработчиков и пользователей.

Дополнительные данные указывают на то, что каждый день создается и запускается сотни тысяч новых веб-сервисов на платформе WordPress, что подчеркивает ее значительный вклад в развитие онлайн-пространства. Более того, сообщество WordPress состоит из миллионов разработчиков, дизайнеров и пользователей, которые активно взаимодействуют, обмениваются опытом и создают новые решения для улучшения функционала и производительности этой платформы.

Исследование использования WordPress и его компонентов, таких как версии и языки программирования, предоставляет важные данные о текущих тенденциях в веб-разработке и предпочтениях пользователей. Например, согласно последнему исследованию, версия 6.5 WordPress пользуется

наибольшей популярностью среди пользователей, составляя 52.5% от общего числа. В то время как версия 6.4 используется 16% пользователей, а версия 6.2 – 4.1%.

Относительно версий PHP, используемых при развертывании веб-сервиса, наблюдается следующая динамика: версия PHP 7.4 используется 42.3% пользователей, версия PHP 8.1 – 17%, а версия PHP 8.0 – 11.9%. Это связано с тем, что крупные компании, особенно те, которые работают с критически важными веб-сервисами и имеют сложные инфраструктуры, предпочитают оставаться на стабильных версиях PHP из-за особенностей их окружения и требований безопасности.

Что касается баз данных, наибольшей популярностью среди пользователей WordPress пользуется MySQL 5.7, которую использует 28.3% пользователей. MySQL 8.0 используется 14.3% пользователей, а MySQL 5.5 – 11.5% [19].

Эти данные помогают понять текущий пейзаж использования WordPress и его компонентов, а также выявить тенденции и предпочтения пользователей в веб-разработке.

Запуск веб-сервиса на WordPress требует от пользователя освоения основных принципов функционирования платформы, что может оказаться сложным, особенно для новичков. Однако, благодаря широкому выбору обучающих ресурсов, изучение WordPress доступно даже без опыта.

Преимущества WordPress включают полный контроль над каждым аспектом веб-сервиса и доступ к огромному количеству плагинов, предоставляющих дополнительные функциональные возможности. Это позволяет пользователям настраивать веб-сервис в соответствии с их уникальными потребностями и предпочтениями.

Кроме того, для дизайна веб-сервиса в WordPress доступны тысячи бесплатных и платных шаблонов, а также конструкторы страниц, такие как Elementor, которые позволяют создавать веб-сервисы без необходимости в программировании.

Несмотря на то, что использование WordPress требует времени и технических знаний, эта платформа остается привлекательным выбором благодаря своей гибкости и масштабируемости. С возможностью использования WordPress бесплатно, кроме расходов на хостинг, это является привлекательным вариантом для создания веб-сервисов с ограниченным бюджетом.

Таким образом, несмотря на свои технические сложности, WordPress остается лидером среди платформ для создания веб-сервисов благодаря своей гибкости, множеству доступных ресурсов и возможности адаптации под различные потребности пользователей.

Одной из ключевых особенностей WordPress является его простота использования и гибкость настройки. Это делает его популярным выбором как для небольших личных блогов, так и для крупных корпоративных веб-сервисов. Благодаря обширному сообществу разработчиков и пользователей, WordPress постоянно обновляется и совершенствуется, предлагая новые возможности и улучшения.

Для разработки и настройки веб-сервиса WordPress часто используются различные библиотеки и инструменты. Например, для управления версиями кода и совместной работы разработчиков может применяться система контроля версий Git, а для автоматизации процессов развертывания и управления инфраструктурой — инструменты DevOps, такие как Ansible или Terraform.

В области фронтенд разработки для создания пользовательского интерфейса и визуального оформления веб-сервиса могут использоваться различные фреймворки и библиотеки CSS и JavaScript, такие как Bootstrap, Foundation или React. Данные фреймворки и библиотеки предоставляют готовые компоненты и стили, упрощающие создание современных и адаптивных дизайнов.

Для оптимизации производительности и безопасности веб-сервиса WordPress могут применяться различные инструменты и плагины. Например, для кэширования контента и ускорения загрузки страниц используются плагины типа WP Super Cache или W3 Total Cache. Для обеспечения безопасности сайта применяются инструменты мониторинга и защиты, такие как Wordfence Security или Sucuri Security.

В целом, веб-сервис WordPress представляет собой мощное и гибкое средство для создания и управления веб-сервисами и блогами, которое активно используется множеством разработчиков и пользователей по всему миру. Его широкие возможности и обширный выбор инструментов делают его привлекательным выбором для различных проектов веб-разработки.

Этот раздел составлен с целью обзора разворачиваемого веб-сервиса и является основой для разработки графического материала под названием «Графический интерфейс веб-сервиса», который представлен в приложении Б. В представленном графическом материале описывается пользовательский интерфейс веб-сервиса с акцентом на визуальное представление его функциональных возможностей и основных элементов управления. Графический материал призван обеспечить более наглядное понимание внешнего вида и структуры веб-сервиса, а также дать пользователю представление о его интерфейсе и возможностях.

4.2 Реализация инфраструктуры в виде кода для облачного окружения

Для реализации инфраструктуры в облачном окружении было выбрано использование Terraform в связи с его возможностью представления инфраструктуры в виде кода (Infrastructure as Code) и автоматизации процесса развертывания. Это позволяет снизить риски и обеспечить консистентность и надежность инфраструктуры.

Реализация инфраструктуры в виде кода будет реализовываться с помощью официальных модулей от HashiCorp, таких как VPC и SG (Security Group Module). Данные модули обладают хорошей документацией и обеспечивают высокую степень гибкости и масштабируемости. Для обоснования использования модулей, необходимо привести пример создания VPC вместе с Public Subnet, EC2 Instance с ключом и Security Group, Route Table и Internet Gateway без использования модулей:

```
# Создание VPC
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
  enable_dns_support = true
  enable_dns_hostnames = true
  tags = {
    Name = "my_vpc"
  }
}

# Создание Internet Gateway
resource "aws_internet_gateway" "my_igw" {
  vpc_id = aws_vpc.my_vpc.id
}

# Создание Public Subnet
resource "aws_subnet" "public_subnet" {
  vpc_id            = aws_vpc.my_vpc.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = "eu-north-1a"
  map_public_ip_on_launch = true
  tags = {
    Name = "public_subnet"
  }
}

resource "aws_route_table" "public_route_table" {
  vpc_id = aws_vpc.my_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.my_igw.id
  }
}
```

```

tags = {
    Name = "public_route_table"
}
}

# Ассоциируем созданную маршрутную таблицу с Public Subnet
resource "aws_route_table_association" "public_route_table_association" {
    subnet_id      = aws_subnet.public_subnet.id
    route_table_id = aws_route_table.public_route_table.id
}

# Создание Security Group для EC2 Instance
resource "aws_security_group" "my_security_group" {
    name           = "my_security_group"
    description    = "Allow SSH and HTTP traffic"
    vpc_id        = aws_vpc.my_vpc.id

    ingress {
        from_port   = 22
        to_port     = 22
        protocol    = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }

    ingress {
        from_port   = 80
        to_port     = 80
        protocol    = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }

    egress {
        from_port   = 0
        to_port     = 0
        protocol    = "-1"
        cidr_blocks = ["0.0.0.0/0"]
    }
}

# Создание EC2 Instance
resource "aws_instance" "my_instance" {
    ami           = "ami-1234567890abcdef0"
    instance_type = "t2.micro"
    subnet_id     = aws_subnet.public_subnet.id
    key_name      = "my_key_pair_name"
    security_groups = [aws_security_group.my_security_group.name]
    tags = {
        Name = "my_instance"
    }
}

```

Этот код создает VPC с CIDR-блоком «10.0.0.0/16», Internet Gateway, публичную Subnet с CIDR-блоком «10.0.1.0/24», Route Table для этой Subnet с маршрутом в интернет через Internet Gateway, Security Group для EC2 Instance

с разрешенным трафиком по протоколам SSH и HTTP, и EC2 Instance внутри Public Subnet.

Создание явного инфраструктурного кода без использований модулей и count атрибутов не позволяет гибко масштабировать инфраструктуру и при добавлении нового ресурса приходится создавать новые строчки кода.

Необходимо рассмотреть тот же пример, но с использованием модуля VPC от HashiCorp:

```
data "aws_internet_gateway" "default" {
  filter {
    name     = "attachment.vpc-id"
    values   = [aws_default_vpc.get.id]
  }
}

module "vpc" {
  source = "./modules/vpc"

  create_vpc = true
  cidr       = "10.0.0.0/16"

  azs = ["eu-north-1a"]

  public_gateway_id      = data.aws_internet_gateway.default.id
  public_subnets        = ["10.0.1.0/24", "10.0.2.0/24"]
  public_subnet_names    = ["Public Subnet #1", "Public Subnet #2"]
  public_route_table_tags = { "Name" = "Route table of public subnet" }
  map_public_ip_on_launch = true
}

module "ec2_security_group" {
  source = "./modules/sg"

  name           = "EC2-sg"
  description    = "Open traffic from all IPv4"
  vpc_id         = module.vpc.vpc_id

  ingress_cidr_blocks = ["0.0.0.0/0"]
  ingress_rules       = ["http-80-tcp", "ssh-tcp"]

  egress_cidr_blocks = ["0.0.0.0/0"]
  egress_rules       = ["all-all"]
}

resource "aws_instance" "my_instance" {
  ami           = "ami-1234567890abcdef0"
  instance_type = "t2.micro"
  subnet_id    = module.vpc.public_subnets[0]
  key_name     = "my_key_pair_name"
  security_groups = [module.ec2_security_group.security_group_id]
  tags = {
    Name = "my_instance"
  }
}
```

}

Использование модулей Terraform обеспечивает более читабельный и модульный код, что позволяет разрабатывать инфраструктуру более удобную для поддержки и расширения. В приведенных примерах видно, что модули `vpc` и `security-group` упрощают создание соответствующих ресурсов, а также их настройку и связывание между собой. Это позволяет сосредоточиться на основных задачах развертывания инфраструктуры, а не на деталях каждого отдельного ресурса.

Более того, использование модулей позволяет создавать множество экземпляров одного и того же ресурса, таких как `Public Subnet`, `Private Subnet` и другие, без необходимости явного дублирования кода или использования сложных конструкций типа `count`. Это упрощает разработку инфраструктуры как кода и уменьшает время, затраченное на тестирование и отладку.

Использование модулей Terraform является рациональным решением в рамках данного дипломного проекта, что обосновывается несколькими причинами. Во-первых, модули позволяют организовать и структурировать код, делая его более понятным и управляемым, особенно при работе с крупными проектами. Во-вторых, модули обеспечивают повторное использование кода, что уменьшает дублирование и упрощает поддержку и обновление инфраструктуры. В-третьих, использование модулей способствует созданию абстракций и стандартизации конфигурации, что повышает уровень абстракции и упрощает внесение изменений.

В спроектированной облачной инфраструктуре предусмотрена система логирования на основе `CloudWatch Log Group`. Для этого создается `IAM Policy`, которая привязывается к соответствующей `IAM Role`. Результатом является `Flow Log Role`, которая затем связывается с созданным `IAM Instance Profile`. После этого необходимо ассоциировать `IAM Instance Profile` с `EC2 Instance`, который требуется логировать.

Для реализации данного функционала необходимо разработать модуль `cloudwatch`, который будет включать в себя определение ресурсов для настройки системы логирования `CloudWatch`. Этот модуль будет содержать конфигурацию для создания `CloudWatch Log Group`, `IAM Policy`, `IAM Role` и `IAM Instance Profile`. После этого модуль будет использоваться для развертывания системы логирования в облачной инфраструктуре.

Реализация манифеста `main.tf` модуля `modules/cloudwatch` выглядит следующим образом:

```
resource "aws_cloudwatch_log_group" "log_group" {  
  name = lookup(var.names, "cloudwatch_log_group", "unspecified-name")  
}
```

```

    tags = merge(var.tags, lookup(var.tags_for_resource,
"aws_cloudwatch_log_group", {}))
}

resource "aws_iam_policy" "flow_log_policy" {
    name = lookup(var.names, "iam_policy", "unspecified-name")
    path = "/"

    policy = var.resource_policy_json
}

resource "aws_iam_role" "flow_log_role" {
    name          = lookup(var.names, "iam_role", "unspecified-name")
    assume_role_policy = var.trust_policy_json
}

resource "aws_iam_policy_attachment" "attach_resource_policy_to_role" {
    name          = lookup(var.names, "iam_policy", "unspecified-name")
    policy_arn    = aws_iam_policy.flow_log_policy.arn
    roles         = [aws_iam_role.flow_log_role.name]
}

resource "aws_flow_log" "flow_log_parameters" {
    iam_role_arn    = aws_iam_role.flow_log_role.arn
    log_destination = aws_cloudwatch_log_group.log_group.arn
    traffic_type    = "ALL"
    subnet_id       = var.subnet_id
}

resource "aws_iam_instance_profile" "FlowLogInstanceProfile" {
    name = lookup(var.names, "instance_profile", "unspecified-name")
    role = aws_iam_role.flow_log_role.name
}

```

Реализация манифеста `outputs.tf` модуля `modules/cloudwatch` выглядит следующим образом:

```

output "IAMInstanceProfileRoleName" {
    value = aws_iam_instance_profile.FlowLogInstanceProfile.name
}

```

Вывод `IAMInstanceProfileRoleName` необходим для получения имени в главном манифесте Terraform при имплементации модуля.

Переменные манифеста `variables.tf` модуля `modules/cloudwatch` выглядят следующим образом:

```

variable "tags" {
    description = "A map of tags to assign to resources"
    type        = map(string)
    default     = {}
}

```



```

variable "tags_for_resource" {
  description = "A nested map of tags to assign to specific resource types"
  type        = map(map(string))
  default = {
    aws_cloudwatch_log_group = {
      "Description" = "Created using the cloudwatch module by khomenokkg"
    }
  }
}

variable "names" {
  description = "A nested map of tags to assign to specific resource types"
  type        = map(string)
  default = {
    cloudwatch_log_group = "CloudWatch-LogGroup"
    iam_policy            = "Policy-LogGroup"
    iam_role              = "Role-LogGroup"
    instance_profile      = "FlowLogInstanceProfile"
  }
}

variable "subnet_id" {
  type    = string
  default = ""
}

variable "resource_policy_json" {
  default = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Resource": "*"
    }
  ]
}
EOF
}

variable "trust_policy_json" {
  default = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "vpc-flow-logs.amazonaws.com"
      }
    }
  ]
}
EOF
}

```

```

    },
    "Action": "sts:AssumeRole"
  }
]
}
EOF
}

```

Таким образом, был реализован модуль CloudWatch. Модуль cloudwatch реализует инфраструктуру для системы логирования с использованием сервиса Amazon CloudWatch. Файл main.tf определяет ресурсы, необходимые для создания и настройки системы логирования. Были задействованы следующие ресурсы провайдера AWS:

- aws_cloudwatch_log_group создает группу журналов в Amazon CloudWatch с указанным именем и тегами;
- aws_iam_policy определяет политику безопасности IAM для разрешения действий над журналами CloudWatch;
- aws_iam_role создает роль IAM, которая будет использоваться для доступа к журналам CloudWatch;
- aws_iam_policy_attachment привязывает политику безопасности к роли IAM;
- aws_flow_log настраивает журнал потока VPC для отправки событий трафика в созданную группу журналов CloudWatch;
- aws_iam_instance_profile создает профиль экземпляра IAM, который привязывается к роли IAM и используется для автоматической аутентификации на инстансе EC2.

Файл outputs.tf определяет выходные значения, которые могут быть использованы другими модулями или внешними программами.

Файл variables.tf содержит переменные, используемые в модуле, такие как теги, идентификатор подсети и политики безопасности в формате JSON.

Модуль cloudwatch был реализован для упрощения создания и настройки инфраструктуры логирования с использованием сервиса Amazon CloudWatch. Это позволяет развернуть все необходимые ресурсы с помощью единой конфигурации и предоставляет удобные выходные данные для дальнейшего использования.

Далее необходимо перейти к реализации облачной инфраструктуры в виде кода с использованием модулей от HashiCorp – VPC и Security Group, а также лично разработанным модулем cloudwatch.

Для реализации спроектированной облачной инфраструктуры в виде кода, главный файл-манифест Terraform main.tf выглядит следующим образом:

```

provider "aws" {
  region = var.aws_region
}

data "aws_vpc" "default" {
  default = true
}

data "aws_security_group" "default" {
  name     = "default"
  vpc_id = data.aws_vpc.default.id
}

data "aws_internet_gateway" "default" {
  filter {
    name     = "attachment.vpc-id"
    values = [aws_default_vpc.get.id]
  }
}

data "aws_secretsmanager_secret_version" "db_credentials" {
  secret_id = "db-credentials"
}

locals {
  db_credentials =
  jsondecode(data.aws_secretsmanager_secret_version.db_credentials.secret_string)
}

resource "aws_default_vpc" "get" {
  tags = {
    Name = "Default VPC"
  }
}

module "vpc" {
  source = "../modules/vpc"

  create_vpc = false

  azs = ["eu-north-1a"]

  public_gateway_id      = data.aws_internet_gateway.default.id
  public_vpc_id          = aws_default_vpc.get.id
  public_subnets        = ["172.31.49.0/24"]
  public_subnet_names    = ["Public Subnet #1"]
  public_route_table_tags = { "Name" = "Route table of public subnet" }
  map_public_ip_on_launch = true

  private_vpc_id         = aws_default_vpc.get.id
  private_subnets       = ["172.31.50.0/24"]
  private_subnet_names   = ["Private Subnet #1"]
  private_route_table_tags = { "Name" = "Route table of private subnet" }
}

```

```

module "ec2_rds" {
  source = "../modules/sg"

  name          = "ec2-rds-1"
  description   = "Open traffic from all IPv4"
  vpc_id        = module.vpc.default_vpc_id

  ingress_cidr_blocks = ["0.0.0.0/0"]
  ingress_rules       = ["all-all"]

  egress_cidr_blocks = ["0.0.0.0/0"]
  egress_rules       = ["all-all"]
}

module "rds_ec2" {
  source = "../modules/sg"

  name          = "rds-ec2-1"
  description   = "Open inbound traffic from EC2 to RDS"
  vpc_id        = module.vpc.default_vpc_id

  ingress_cidr_blocks = ["172.31.49.0/24"]
  ingress_rules       = ["mysql-tcp"]
}

module "db_computed_source_sg" {
  source = "../modules/sg"

  vpc_id = module.vpc.default_vpc_id

  name          = "RDS-EC2-1"
  ingress_cidr_blocks = ["172.31.49.0/24"]
  egress_cidr_blocks = ["0.0.0.0/0"]
  egress_rules       = ["mysql-tcp"]

  computed_ingress_with_source_security_group_id = [
    {
      rule          = "mysql-tcp"
      source_security_group_id = module.ec2_rds.security_group_id
    }
  ]
  number_of_computed_ingress_with_source_security_group_id = 1
}

data "aws_ami" "getLatestUbuntu" {
  most_recent = true

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
  }

  owners = ["099720109477"]
}

resource "aws_eip_association" "web-srv-eip" {

```

```

    instance_id    = aws_instance.public_instance.id
    allocation_id = "eipalloc-011244477dcc86e94"
}

resource "aws_key_pair" "public_instance_key" {
    key_name     = var.key_name
    public_key   = var.public_key
}

module "cloudwatch" {
    source      = "./modules/cloudwatch"
    subnet_id  = module.vpc.public_subnets[0]
}

resource "aws_instance" "public_instance" {
    ami          = data.aws_ami.getLatestUbuntu.id
    instance_type = "t3.micro"

    key_name          = aws_key_pair.public_instance_key.key_name
    security_groups   = [module.ec2_rds.security_group_id]
    subnet_id        = module.vpc.public_subnets[0]
    availability_zone = "eu-north-1a"
    iam_instance_profile = module.cloudwatch.IAMInstanceProfileRoleName

    connection {
        type      = "ssh"
        host      = self.public_ip
        user      = "ubuntu"
        private_key = var.private_key
        timeout    = "3m"
    }

    tags = {
        Name = "Instance of T3.Micro in Public subnet"
    }
}

resource "aws_db_instance" "rds_instance" {
    vpc_security_group_ids = [module.db_computed_source_sg.security_group_id]
    instance_class          = "db.t3.micro"
    multi_az                = false
    publicly_accessible     = false
    availability_zone       = "eu-north-1a"
    identifier              = var.db_identifier

    engine          = "mysql"
    engine_version  = "5.7"
    allocated_storage = 10
    db_name         = local.db_credentials.dbname

    username = local.db_credentials.username
    password = local.db_credentials.password

    parameter_group_name = "default.mysql5.7"
    skip_final_snapshot  = true
}

```

Для обеспечения безопасного хранения секретных данных подключения к базе данных был выбран сервис Amazon Secrets Manager. Доступ к данным осуществляется исключительно через Terraform и AWS API. Файл .tfstate также хранится удаленно.

Terraform получает доступ к данным, хранящимся в Amazon Secrets Manager с использованием ресурса `aws_secretsmanager_secret_version`. Для этого в конфигурации Terraform указывается идентификатор секрета, содержащего данные о подключении к базе данных. После получения секрета, его содержимое декодируется из JSON формата с помощью функции `jsondecode()`, и затем используется для настройки ресурса `aws_db_instance`, который представляет собой экземпляр базы данных RDS.

Помимо этого, вывод некоторых данных (публичный IPv4-адрес EC2, доменное имя для подключения к RDS, имя базы данных, имя пользователя базы данных, пароль базы данных) также скрываются как чувствительные и секретные данные (представлен файл манифеста `outputs.tf`):

```
output "instance_public_ip" {
  value      = aws_eip_association.web-srv-eip.public_ip
  sensitive = true
}

output "credentials_dbhost" {
  value      = aws_db_instance.rds_instance.address
  sensitive = true
}

output "credentials_dbname" {
  value      = local.db_credentials.dbname
  sensitive = true
}

output "credentials_dbusername" {
  value      = local.db_credentials.username
  sensitive = true
}

output "credentials_dbpassword" {
  value      = local.db_credentials.password
  sensitive = true
}
```

В заключение, стоит отметить, что в данном дипломном проекте была реализована инфраструктура как код в рамках дипломного проекта по применению DevOps технологий для поддержки распределенных веб-сервисов на AWS с использованием Terraform.

Дополнительно, данный раздел, в котором подробно описывается практическая реализация инфраструктуры в виде кода, послужил основой для

разработки двух графических материалов. Первый материал представляет собой IDEF0 диаграмму декомпозиции, где изображен начальный процесс под названием «Процесс проектирования и создания инфраструктуры как кода для AWS». Этот материал представлен в приложении Б и предназначен для наглядного представления структуры и этапов процесса создания инфраструктуры.

Второй материал представляет собой графическое изображение «Структуры манифеста Terraform», которое приведено в приложении Б. Данный материал дает обзор основных компонентов и структуры файлов манифеста Terraform, что помогает визуально представить организацию кода и его логическую структуру.

4.3 Настройка непрерывной интеграции и доставки (CI/CD)

Для настройки непрерывной интеграции и доставки (CI/CD) будет использован набор инструментов, включающий GitHub Actions, Terraform и docker-compose с контейнерами nginx, certbot и wordpress.

GitHub Actions будет использоваться для автоматизации процессов CI/CD. С его помощью будет настроено автоматическое тестирование и сборка веб-сервиса при каждом обновлении кодовой базы в репозитории. Также будут настроены действия для автоматического развертывания веб-сервиса на целевом сервере.

Terraform будет использован для определения инфраструктуры в виде кода, что позволит управлять инфраструктурой как программным обеспечением. С его помощью будут настроены ресурсы AWS, такие как виртуальные машины, сетевые настройки и другие необходимые компоненты.

Docker-compose будет использоваться для локального развертывания и тестирования веб-сервиса в контейнерах, позволяя легко запускать и масштабировать контейнеризованные сервисы, такие как nginx, certbot и wordpress, на локальной машине для разработки и отладки.

Таким образом, интеграция GitHub Actions с Terraform и docker-compose обеспечит автоматизацию процесса разработки, тестирования и развертывания веб-сервиса, что позволит повысить эффективность и надежность разработки и доставки программного обеспечения.

Для реализации непрерывной интеграции и доставки необходимо сконфигурировать файлы для автоматизации развертывания веб-сервиса. Для начала, необходимо сконфигурировать docker-compose с сервисами nginx, certbot и wordpress для 80 порта HTTP, а также конфигурацию для веб-сервера nginx.

Конфигурация docker-compose.yml для сервисов WordPress, nginx на 80 порту HTTP и certbot:

```
version: "3.9"

services:
  wordpress:
    container_name: wordpress
    image: wordpress:php8.2-fpm
    restart: always
    env_file:
      - .env
    environment:
      WORDPRESS_DB_HOST: ${CREDENTIALS_DBHOST}
      WORDPRESS_DB_USER: ${CREDENTIALS_DBUSERNAME}
      WORDPRESS_DB_PASSWORD: ${CREDENTIALS_DBPASSWORD}
      WORDPRESS_DB_NAME: ${CREDENTIALS_DBNAME}
    volumes:
      - wordpress_data:/var/www/html

  certbot:
    container_name: certbot
    image: certbot/certbot:latest
    command: certonly --webroot --webroot-path=/var/www/html --email
khomenokkg@gmail.com --agree-tos -d wordpress.diplomadomain.online
    volumes:
      - certbot-etc:/etc/letsencrypt
      - wordpress_data:/var/www/html

  web:
    container_name: webnginx
    image: nginx:latest
    volumes:
      - wordpress_data:/var/www/html
      - ./nginx:/etc/nginx/conf.d
      - certbot-etc:/etc/letsencrypt
    ports:
      - "80:80"
    restart: always

volumes:
  wordpress_data:
  certbot-etc:
```

Конфигурация веб-сервера nginx для его прослушивания на 80 порту HTTP (default.conf):

```
server {
    listen 80;
    listen [::]:80;

    server_name wordpress.diplomadomain.online;

    index index.php index.html index.htm;
```



```

root /var/www/html;

location ~ /\.well-known/acme-challenge {
    allow all;
    root /var/www/html;
}

location / {
    try_files $uri $uri/ /index.php$is_args$args;
}

location ~ /\.php$ {
    try_files $uri =404;
    fastcgi_split_path_info ^(.+\.(php))(/.+)$;
    fastcgi_pass wordpress:9000;
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_param                                SCRIPT_FILENAME
$document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
}

location ~ /\.ht {
    deny all;
}

location = /favicon.ico {
    log_not_found off; access_log off;
}
location = /robots.txt {
    log_not_found off; access_log off; allow all;
}
location ~* \.(css|gif|ico|jpeg|jpg|js|png)$ {
    expires max;
    log_not_found off;
}
}

```

Файл `docker-compose.yml` содержит описание всех контейнеров, их настроек и зависимостей. В данном случае, файл содержит описание трех сервисов: `wordpress`, `certbot`, и `web`.

Сервис `wordpress` запускает контейнер с WordPress, используя образ `wordpress:php8.2-fpm`. Более того, сервис указывает на файл `.env` для загрузки переменных окружения, таких как адрес и учетные данные базы данных WordPress. Также, сервис определяет том `wordpress_data`, который используется для сохранения данных WordPress.

Сервис `certbot` запускает контейнер с Certbot, который используется для получения SSL-сертификата от Let's Encrypt. Команда `certonly` указывает, что нужно получить сертификат, но не устанавливать его автоматически. Параметры команды указывают путь к веб-корню и адрес электронной почты

для уведомлений об обновлениях сертификата. Также, сервис определяет том certbot-etc для сохранения конфигурационных файлов Certbot.

Сервис web запускает контейнер с Nginx, который используется в качестве веб-сервера для WordPress. Для разворачивания веб-сервиса необходимо определить тома, – wordpress_data и certbot-etc, а также локальный файл конфигурации Nginx из каталога ./nginx. Порт 80 контейнера привязывается к порту 80 хоста, чтобы обеспечить доступ к веб-серверу через HTTP.

Вместе с файлом docker-compose.yml, используется конфигурационный файл default.conf для Nginx, который определяет настройки веб-сервера, включая проксирование запросов к WordPress. При первом запуске контейнера Certbot, сервис использует порт 80 для выполнения проверки владения доменным именем и запроса SSL-сертификата. После получения сертификата и завершения первоначальной настройки, необходимо остановить контейнеры, изменить конфигурацию для прослушивания порта 443 и перенаправления трафика через SSL. Затем контейнеры можно снова запустить, чтобы обеспечить безопасное соединение с веб-сервером через HTTPS.

Docker-compose для разворачиваемого веб-сервиса с прослушиванием 443 порта HTTPS отличается сервисом nginx с настроенным портом 443 и выглядит следующим образом:

```
web:
  container_name: webnginx
  image: nginx:latest
  volumes:
    - wordpress_data:/var/www/html
    - ./nginx:/etc/nginx/conf.d
    - certbot-etc:/etc/letsencrypt
  ports:
    - "80:80"
    - "443:443"
  restart: always
```

Конфигурационный файл с настройками веб-сервера Nginx для обеспечения прослушивания порта 443 с использованием протокола HTTPS:

```
server {
    listen 80;
    listen [::]:80;

    server_name wordpress.diplomadomain.online;

    location ~ /.well-known/acme-challenge {
        allow all;
        root /var/www/html;
```

```

    }

    location / {
        rewrite ^ https://$host$request_uri? permanent;
    }
}

server {
    listen 443 ssl http2;
    server_name wordpress.diplomadomain.online;

    index index.php index.html index.htm;

    root /var/www/html;

    server_tokens off;

    ssl_certificate
/etc/letsencrypt/live/wordpress.diplomadomain.online/fullchain.pem;
    ssl_certificate_key
/etc/letsencrypt/live/wordpress.diplomadomain.online/privkey.pem;

    include /etc/nginx/conf.d/options-ssl-nginx.conf;

    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header Referrer-Policy "no-referrer-when-downgrade" always;
    add_header Content-Security-Policy "default-src * data: 'unsafe-eval'
'unsafe-inline'" always;
    # add_header Strict-Transport-Security "max-age=31536000;
includeSubDomains; preload" always;
    # enable strict transport security only if you understand the
implications

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ /\.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.(php))(/.+)$;
        fastcgi_pass wordpress:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }

    location ~ /\.ht {
        deny all;
    }

    location = /favicon.ico {
        log_not_found off; access_log off;

```

```

    }
    location = /robots.txt {
        log_not_found off; access_log off; allow all;
    }
    location ~* \.(css|gif|ico|jpeg|jpg|js|png)$ {
        expires max;
        log_not_found off;
    }
}

```

Представленный конфигурационный файл nginx определяет параметры для обработки запросов на двух портах, 80 и 443. При обращении к порту 80 сервер принимает запросы для домена `wordpress.diplomadomain.online`, а также обрабатывает запросы к директории `/.well-known/acme-challenge`, что позволяет сервису Certbot выполнять проверку владения доменом при запросе SSL-сертификата. Все остальные запросы с порта 80 автоматически перенаправляются на HTTPS для обеспечения безопасного соединения. На порту 443 сервер также обслуживает запросы для указанного домена, используя SSL-сертификат и приватный ключ для создания защищенного соединения. Дополнительные HTTP-заголовки, такие как `X-Frame-Options`, `X-XSS-Protection`, и `Content-Security-Policy`, добавляются для усиления безопасности сервера. Кроме того, в конфигурации определены правила обработки запросов к PHP-скриптам и статическим файлам, а также настройки для специальных файлов типа `.htaccess`, `favicon.ico`, и `robots.txt`.

Помимо этого, для правильной настройки SSL необходимо загрузить специальный конфигурационный файл для SSL:

```

curl -sSL0 https://raw.githubusercontent.com/certbot/certbot/master/certbot-nginx/certbot_nginx/_internal/tls_configs/options-ssl-nginx.conf

```

В рамках настройки непрерывной интеграции и доставки следует создать структуру каталогов, начиная с корневого каталога проекта. В нем создается директория «`terraform`», где будет содержаться проект, реализующий инфраструктуру для веб-сервиса. Важно учитывать, что конфиденциальные данные, такие как ссылка для подключения к RDS, имя пользователя и пароль для базы данных, следует хранить в Amazon Secrets Manager и получать через Terraform, с указанием соответствующего атрибута «`sensitive = true`».

Далее создаются две директории «`wp-http`» и «`wp-https`» в корневом каталоге. Внутри каждой из них располагается директория «`nginx`», содержащая конфигурационный файл для веб-сервера. В «`wp-http`» размещается файл «`docker-compose.yml`» с сервисами WordPress, Certbot и

Nginx, настроенными на прослушивание 80 порта. Внутри «nginx/default.conf» хранится конфигурация веб-сервера для работы на порту 80.

Аналогично, в директории «wp-https» размещается файл «docker-compose.yml» с теми же сервисами, настроенными на прослушивание 443 порта. Внутри «nginx/default.conf» содержится конфигурация веб-сервера для работы на порту 443 с SSL-настройками.

Итоговая структура репозитория представлена на рисунке 4.1.

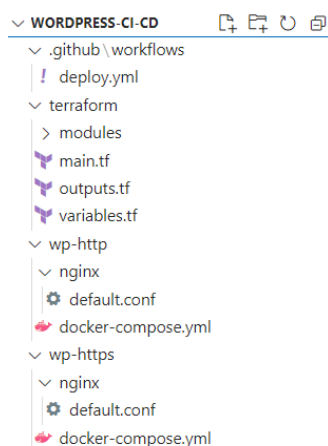


Рисунок 4.1 – Итоговая структура репозитория

В конечном итоге необходимо создать приватный репозиторий на GitHub, где происходит настройка GitHub Secrets в GitHub Actions на этапе настройки и описания пайплайна CI/CD. Эти секреты будут использоваться в процессе CI/CD для безопасного доступа к конфиденциальным данным и настроек веб-сервера.

Для настройки GitHub Secrets and variables необходимо перейти в настройки приватного репозитория, кликнув «Settings», далее необходимо перейти в категорию «Security» и выбрать раздел «Secrets and variables», после чего перейти в «Actions» (рисунок 4.2).

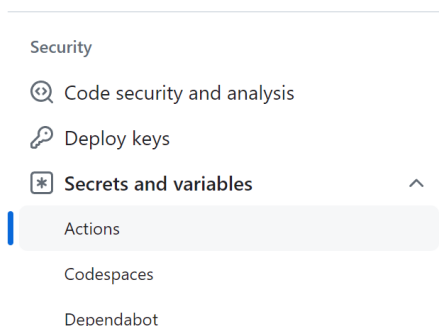


Рисунок 4.2 – Расположение настройки «GitHub Secrets and Variables»

Далее необходимо нажать кнопку «New repository secret» и настроить такие переменные окружения, как `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SSH_KEY_PRIVATE`, `AWS_SSH_KEY_PUBLIC`, `AWS_TF_STATE_BUCKET_NAME` и `GH_ACCESS_TOKEN`.

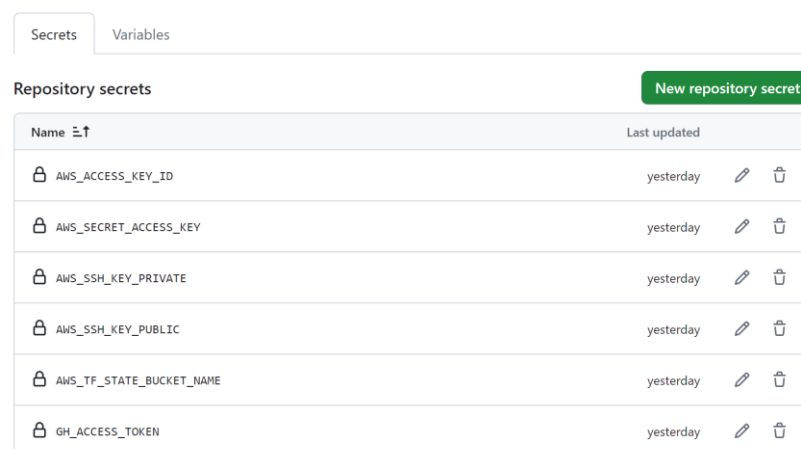
Для настройки переменных окружения `AWS_ACCESS_KEY_ID` и `AWS_SECRET_ACCESS_KEY`, необходимо зайти в консоль AWS и перейти в раздел IAM (Identity and Access Management).

Переменные окружения `AWS_SSH_KEY_PRIVATE` и `AWS_SSH_KEY_PUBLIC` необходимо сгенерировать с помощью команды `ssh-keygen`, после чего добавить значения в GitHub Secrets.

`AWS_TF_STATE_BUCKET_NAME` – это имя Amazon S3, где будет храниться файл `.tfstate`.

`GH_ACCESS_TOKEN` – это персональный токен доступа GitHub, который используется для авторизации в GitHub при работе с репозиторием, данный токен необходимо создать в настройках профиля пользователя, который владеет приватным репозиторием.

Настроенные переменные Secrets для GitHub Actions должны выглядеть следующим образом (рисунок 4.2).













Secrets	Variables
Repository secrets	
New repository secret	
Name ↕	Last updated
 <code>AWS_ACCESS_KEY_ID</code>	yesterday  
 <code>AWS_SECRET_ACCESS_KEY</code>	yesterday  
 <code>AWS_SSH_KEY_PRIVATE</code>	yesterday  
 <code>AWS_SSH_KEY_PUBLIC</code>	yesterday  
 <code>AWS_TF_STATE_BUCKET_NAME</code>	yesterday  
 <code>GH_ACCESS_TOKEN</code>	yesterday  

Рисунок 4.2 – Настроенные переменные окружения в GitHub Action Secrets

После настройки переменных окружения в GitHub Actions Secrets необходимо поместить все разработанные конфигурационные файлы в репозиторий и сделать Initial commit с последующим push'ем в репозиторий. Ниже представлена примерная реализация:

```
git config --global user.email "emailaddress@domain.com"
git config --global user.name "username"
```

```
git clone $GH_ACCESS_TOKEN@github.com:$USER/$REPOSITORY.git
cd $REPOSITORY
cp -R path/to/confs/* .
git                                remote                                add                                origin
https://$GH_ACCESS_TOKEN@github.com:$USER/$REPOSITORY.git
git commit -m "Initial commit"
git push
```

Далее необходимо создать директорию `.github`, после чего перейти в данный каталог и создать директорию `workflows`, внутри которой необходимо создать файл `deploy.yml`. Ниже представлена реализация:

```
mkdir -p .github/workflow/ && cd $_ && touch deploy.yml
```

После завершения настройки конфигурационных файлов и переменных окружения в GitHub Actions, а также выполнения Initial Commit, который включал в себя все необходимые конфигурационные файлы, необходимо начать написание пайплайна GitHub Actions. Пайплайн будет описан в файле `deploy.yml`, который находится по следующему пути в вашем репозитории: `~$GITREPO/.github/workflows/deploy.yml`.

Пайплайн GitHub Actions будет определять последовательность шагов, которые будут выполняться при каждом обновлении репозитория. Эти шаги могут включать такие действия, как сборка, тестирование, развертывание и другие автоматизированные задачи.

Пользователи GitHub используют в пайплайне различные директивы и действия GitHub Actions, такие как `on`, `jobs`, `steps` и другие, чтобы определить условия запуска пайплайна, настроить задачи и указать команды для выполнения в рамках каждого шага.

После написания пайплайна GitHub Actions необходимо его сохранить в файле `deploy.yml` в директории `.github/workflows` приватного репозитория.

Написанный пайплайн нацелен на реализацию непрерывной интеграции и доставки (CI/CD) инфраструктуры и веб-сервиса WordPress с использованием инструментов Terraform и Docker. Пайплайн состоит из двух основных задач: «`infrastructure-deployment`» и «`deploy-app`».

Задача «`infrastructure-deployment`» выполняется на Runner, предоставляемым GitHub, с ОС `ubuntu-latest` и состоит из нескольких этапов:

- `checkout repository`: извлечение репозитория для выполнения задачи;
- `setup Terraform`: подготовка среды Terraform;
- `initializing Terraform`: инициализация Terraform с указанием конфигурации удаленного хранилища состояния;
- `planning Terraform`: планирование инфраструктуры и создание файла плана;

- terraform Apply: применение плана, что приводит к развертыванию инфраструктуры в AWS;
- getting EC2 Public IP: получение публичного IP-адреса EC2-экземпляра;
- getting RDS credentials: получение учетных данных для подключения к базе данных RDS;

Задача «deploy-app» также выполняется на Runner от GitHub – ubuntu-latest и зависит от завершения задачи «infrastructure-deployment». Необходимо описать некоторые этапы, которые включает в себя задача:

- checkout: извлечение репозитория для развертывания веб-сервиса.
- setting environment variables: установка переменных окружения на основе результатов задачи «infrastructure-deployment».
- connect to EC2 instance via ssh: установка соединения с экземпляром EC2 по SSH и выполнение необходимых действий на сервере, включая установку Docker и Docker Compose, загрузку конфигурационных файлов и запуск контейнеров.

В целом, пайплайн автоматизирует процесс развертывания инфраструктуры и веб-сервиса WordPress, начиная с создания необходимых ресурсов в облаке AWS с использованием Terraform и заканчивая развертыванием веб-сервиса на EC2-экземпляре с помощью Docker Compose и необходимых конфигурационных файлов.

После того, как в ветку main попадают новые изменения, то запускается GitHub Action (Workflow) и начинается развертывание инфраструктуры и веб-сервиса, при этом каждый этап отслеживается в GitHub Actions (рисунок 4.3).

Event	Status	Branch	Actor
Finally implementing CI/CD and destroy infrastructure Implementing CI/CD with Terraform #24: Commit b0caaf3 pushed by kh0mika	Failed (Red X)	main	...
Testing CI/CD with security credentials Implementing CI/CD with Terraform #23: Commit 3177643 pushed by kh0mika	Success (Green Checkmark)	main	...
Update deploy.yml Implementing CI/CD with Terraform #13: Commit 494f2af pushed by kh0mika	Success (Green Checkmark)	main	...
Added CloudWatch to infrastructure, replace random rds identifier to ... Implementing CI/CD with Terraform #12: Commit 5313859 pushed by kh0mika	Warning (Yellow Triangle)	main	...

Рисунок 4.3 – Отслеживание работы GitHub Actions (Workflows)

Необходимость разработки мануального Workflow для уничтожения инфраструктуры на платформе Amazon Web Services (AWS) возникает в контексте оптимизации процесса развертывания и обновления веб-сервиса. Добавление операции уничтожения инфраструктуры перед созданием новой в

deploy.yml приводит к нежелательной пересборке инфраструктуры после каждого обновления параметров веб-сервиса. Это повышает время развертывания, что не соответствует требованиям эффективности и производительности.

Мануальный Workflow, описываемый в файле destroy.yml, является специально созданным механизмом, позволяющим уничтожать инфраструктуру на AWS по требованию пользователя. Данный Workflow активируется нажатием специальной кнопки в веб-интерфейсе GitHub Actions.

Итоговая реализация мануального Workflow (.github/workflows/destroy.yml) представлена ниже:

```
# Author: Kiryl Homenok
name: Manually destroy the infrastructure
on: workflow_dispatch

env:
  AWS_ACCESS_KEY_ID: ${ secrets.AWS_ACCESS_KEY_ID }
  AWS_SECRET_ACCESS_KEY: ${ secrets.AWS_SECRET_ACCESS_KEY }
  TF_STATE_BUCKET_NAME: ${ secrets.AWS_TF_STATE_BUCKET_NAME }
  PRIVATE_SSH_KEY: ${ secrets.AWS_SSH_KEY_PRIVATE }
  PUBLIC_SSH_KEY: ${ secrets.AWS_SSH_KEY_PUBLIC }

jobs:
  destroy-infrastructure:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2
      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v1
        with:
          terraform_wrapper: false
      - name: Initializing Terraform
        id: init
        run: terraform init -backend-config="bucket=$TF_STATE_BUCKET_NAME" -
backend-config="region=eu-north-1"
        working-directory: ./terraform
      - name: Destroy Infrastructure
        id: destroy
        run: |-
          terraform destroy --auto-approve \
            -var="public_key=$PUBLIC_SSH_KEY" \
            -var="private_key=$PRIVATE_SSH_KEY" \
            -var="key_name=wordpresscid"
        working-directory: ./terraform
```

В ходе разработки практической части дипломного проекта, ориентированного на применение DevOps технологий для поддержки распределенных веб-сервисов на платформе AWS с использованием Terraform, были осуществлены следующие шаги. Сначала была создана

инфраструктура как код с применением Terraform, включая конфигурационные файлы для сервера Nginx, настроенного на прослушивание портов 80 и 443 веб-сервиса. Далее было приобретено доменное имя и зарегистрировано на Cloudflare, после чего была добавлена А-запись, обеспечивающая постоянный доступ к веб-сервису через Elastic IP Address.

После завершения этапа инфраструктуры были разработаны конфигурационные файлы docker-compose для развертывания веб-сервиса, автоматически получающие SSL-сертификат для доменного имени. Далее был создан приватный репозиторий, в котором были настроены CI/CD процессы: инициализация конфигурационных файлов, последующий push в репозиторий, а также настройка переменных окружения с использованием GitHub Actions Secrets. Затем был написан пайплайн Workflow для автоматизации создания инфраструктуры и деплоя веб-сервиса, а также разработан мануальный Workflow для уничтожения инфраструктуры.

В рамках обеспечения безопасности и конфиденциальности были приняты следующие меры: получение SSL-сертификата, ограничение доступности экземпляра базы данных только из локальной сети виртуальной машины, а также скрытие конфиденциальных данных с использованием AWS Secret Manager и GitHub Actions Secrets.

С учетом неэффективности использования Dockerfile для сборки WordPress, было принято решение развертывать веб-сервис с использованием docker-compose. Заказчик или другие уполномоченные лица, желающие автоматизировать развертывание видоизмененного веб-сервиса WordPress, должны изменить исходный код WordPress и собрать его с помощью Dockerfile. Полученный образ с изменениями в коде следует загрузить на Docker Hub, после чего можно заменить тип образа в конфигурационном файле docker-compose.yml. Таким образом, возможно использование как стандартного WordPress, так и модифицированной версии, адаптированной под специфические требования веб-сервиса.

Данный раздел, посвященный настройке непрерывной интеграции и доставки (CI/CD), послужил основой для создания графического материала под названием «UML диаграмма развертывания веб-сервиса». Этот материал представлен в приложении Б и разработан для наглядного обозначения структуры и процесса развертывания веб-сервиса в рамках инфраструктуры, реализованной с использованием непрерывной интеграции и доставки.

Разработанные файлы: Workflow для создания инфраструктуры и деплоя веб-сервиса (.github/workflows/deploy.yml), а также мануальный Workflow для уничтожения инфраструктуры (.github/workflows/destroy.yml), будут представлены в приложении А – «Листинги программного кода», поскольку его размер значительный.

4.4 Описание технологий разворачивания распределенного Web-сервиса с использованием Terraform

Для развертывания распределенного веб-сервиса был использован комплекс DevOps-технологий.

Во-первых, технология инфраструктуры как кода (IaC) с использованием Terraform: инфраструктура была описана в виде кода и официальных модулей от HashiCorp, таких как VPC, security_group, а также разработанного модуля cloudwatch. Terraform позволил автоматизировать процесс создания и управления ресурсами AWS.

Во-вторых, технология контейнеризации и оркестрации с помощью docker-compose: для упрощения развертывания и масштабирования веб-сервиса был использован Docker Compose, который обеспечивает среду для упаковки веб-сервиса и его зависимостей в контейнеры, а также управляет контейнерами.

В-третьих, технология непрерывной интеграция и непрерывной доставки (CI/CD): были настроены процессы непрерывной интеграции и непрерывной доставки с использованием GitHub Actions и Workflows, что позволяет автоматизировать сборку и развертывание веб-сервиса при каждом обновлении.

В-четвертых, технология управления версиями и управления кодом с использованием Git: для управления исходным кодом и его версиями была использована система контроля версий Git, что позволяет эффективно отслеживать изменения и вносить исправления при необходимости.

И наконец, были приняты меры по обеспечению безопасности и конфиденциальности данных: получение доменного имени, регистрация на Cloudflare, автоматизация получения SSL-сертификатов. Ограничение доступа к базе данных (доступ разрешен исключительно из локальной сети виртуальной машины, которая выступает в качестве сервера для веб-сервиса), это позволяет ограничить нежелательный доступ к базе данных. Помимо этого, на сервере EC2 установлен PEM-ключ, который невозможно взломать, поэтому доступ к базе данных невозможен извне. Хранение конфиденциальным данных (доменное имя для подключения к базе данных, имя базы данных, имя пользователя и используемый пароль) - хранятся в виде JSON в AWS Secrets Manager, получение данных происходит исключительно через Terraform и AWS API, а хранения файла состояния также находится в облаке Amazon S3. Помимо этого, выводы этих данных в Terraform помечены, как sensitive data. Иные данные, такие как ключи доступа к AWS, Private и Public SSH KEY хранятся в GitHub Actions Secrets.

ЗАКЛЮЧЕНИЕ

В ходе дипломного проектирования были разработаны и использованы DevOps-технологии поддержки распределенных Web-сервисов для AWS с использованием Terraform.

Достижение поставленных целей в начале дипломного проектирования происходило в несколько этапов: первым этапом выполнения дипломного проекта стал анализ исходных данных для последующей разработки и применения DevOps-технологий. Во время проектирования были изучены следующие технологии: инфраструктура как код (IaC), контейнеризация и оркестрация с помощью Docker и Docker Compose, непрерывная интеграция и доставка (CI/CD), Git, а также технологии обеспечения безопасности и конфиденциальности данных.

В Visual Studio Code происходила разработка инфраструктурной части для облачного окружения, конфигурационных файлов для контейнеризации и оркестрации окружения веб-сервиса, настройка непрерывной интеграции и доставки. Реализация технологии непрерывной интеграции и доставки (CI/CD) реализовывалась на GitHub с использованием GitHub Actions, Workflows и GitHub Actions Secret.

Для отражения бизнес-процесса проектирования и создания инфраструктуры как кода была реализована «IDEF0 диаграмма декомпозиции». Для представления архитектурного развертывания веб-сервиса был спроектирован графический материал «UML диаграмма развертывания». Для четкого представления структуры и организации кода был спроектирован графический материал «Структура манифеста Terraform». Для отслеживания инфраструктуры облачного окружения был спроектирован графический материал «Схема инфраструктуры AWS». Для наглядного представления внешнего вида и функционала был разработан графический материал «Графический интерфейс веб-сервиса».

Таким образом, в рамках данного дипломного проекта были успешно достигнуты поставленные цели: были разработаны и применены DevOps-технологии поддержки распределенных веб-сервисов для AWS с использованием Terraform.

Проект успешно завершен, результаты могут быть внедрены для улучшения процессов разработки, эксплуатации и внедрение процессов автоматизации развертывания веб-сервиса на платформе AWS с применением DevOps-технологий.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- [1] Top 10 Cloud Provider Comparison 2023 [Электронный ресурс]. – Режим доступа : <https://dev.to/dkechag/cloud-vm-performance-value-comparison-2023-perl-more-1kpp>. – Дата доступа : 27.03.2024.
- [2] 11 Top Cloud Service Providers Globally In 2024 [Электронный ресурс]. – Режим доступа : <https://www.cloudzero.com/blog/cloud-service-providers/>. – Дата доступа : 27.03.2024.
- [3] Top 10 AWS Services for Data Engineering Projects [Электронный ресурс]. – Режим доступа : <https://www.projectpro.io/article/aws-services-for-data-engineering/644>. – Дата доступа : 27.03.2024.
- [4] Top Cloud Service Providers in 2021: AWS, Microsoft Azure and Google Cloud Platform | CloudThat [Электронный ресурс]. – Режим доступа : <https://www.cloudthat.com/resources/blog/top-cloud-service-providers-in-2021-aws-microsoft-azure-and-google-cloud-platform>. – Дата доступа : 27.03.2024.
- [5] What is Azure – Microsoft Cloud Services | Microsoft Azure [Электронный ресурс]. – Режим доступа : <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure>. – Дата доступа : 27.03.2024.
- [6] Top 10 Cloud Service Providers Globally in 2024 – Dgtl Infra [Электронный ресурс]. – Режим доступа : <https://dgtlinfra.com/top-cloud-service-providers/>. – Дата доступа : 27.03.2024.
- [7] Alibaba Cloud DevOps Pipeline (Flow): Enterprise Continuous Delivery Tool [Электронный ресурс]. – Режим доступа : https://www.alibabacloud.com/en/product/apsara-deveops/flow?_p_lc=1. – Дата доступа : 28.03.2024.
- [8] DevOps Capability Improvement Model – Alibaba DevOps Practice Part 26 [Электронный ресурс]. – Режим доступа : https://www.alibabacloud.com/blog/devops-capability-improvement-model---alibaba-devops-practice-part-26_598658. – Дата доступа : 28.09.2024.
- [9] What is Terraform | Terraform | HashiCorp Developer [Электронный ресурс]. – Режим доступа : <https://developer.hashicorp.com/terraform/intro>. – Дата доступа : 01.04.2024.
- [11] Логически изолированное виртуальное частное облако – Цены на Amazon VPC – Amazon Web Services [Электронный ресурс]. – Режим доступа : <https://aws.amazon.com/ru/vpc/features/>. – Дата доступа : 02.04.2024.
- [12] What is Amazon EC2? – Amazon Elastic Compute Cloud [Электронный ресурс]. – Режим доступа :

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>. – Дата доступа : 02.04.2024.

[13] Cloud Object Storage – Amazon S3 – AWS [Электронный ресурс]. – Режим доступа : https://aws.amazon.com/s3/?nc1=h_ls. – Дата доступа : 02.04.2024.

[14] Managed SQL Database – Amazon Relational Database Service – AWS [Электронный ресурс]. – Режим доступа : https://aws.amazon.com/rds/?nc1=h_ls. – Дата доступа : 03.04.2024.

[15] APM Tool – Amazon CloudWatch – AWS [Электронный ресурс]. – Режим доступа : <https://aws.amazon.com/cloudwatch/>. – Дата доступа : 02.04.2024.

[16] Docker overview | Docker Docs [Электронный ресурс]. – Режим доступа : <https://docs.docker.com/get-started/overview/>. – Дата доступа : 02.04.2024.

[17] What's the Difference Between CI/CD and DevOps? [Электронный ресурс]. – Режим доступа : <https://www.navisite.com/blog/insights/ci-cd-vs-devops/>. – Дата доступа : 02.04.2024.

[18] AWS Architecture Icons [Электронный ресурс]. – Режим доступа : <https://aws.amazon.com/architecture/icons/>. – Дата доступа : 03.04.2024.

[19] Stats – WordPress.org [Электронный ресурс]. – Режим доступа : <https://wordpress.org/about/stats/>. – Дата доступа : 03.04.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинги программного кода

```
# Author: Kiryl Homenok
# File: ../github/workflows/deploy.yml

name: Implementing CI/CD with Terraform
on:
  push:
    branches:
      - main

env:
  AWS_ACCESS_KEY_ID: ${ secrets.AWS_ACCESS_KEY_ID }
  AWS_SECRET_ACCESS_KEY: ${ secrets.AWS_SECRET_ACCESS_KEY }
  TF_STATE_BUCKET_NAME: ${ secrets.AWS_TF_STATE_BUCKET_NAME }
  PRIVATE_SSH_KEY: ${ secrets.AWS_SSH_KEY_PRIVATE }
  PUBLIC_SSH_KEY: ${ secrets.AWS_SSH_KEY_PUBLIC }
  GH_ACCESS_TOKEN: ${ secrets.GH_ACCESS_TOKEN }
  AWS_REGION: eu-north-1

jobs:
  infrastructure-deployment:
    runs-on: ubuntu-latest
    outputs:
      SERVER_PUBLIC_IP: ${ steps.set-ip.outputs.instance_public_ip }
      CREDENTIALS_DBHOST: ${ steps.get-credentials-
dbhost.outputs.credentials_dbhost }
      CREDENTIALS_DBNAME: ${ steps.get-credentials-
dbname.outputs.credentials_dbname }
      CREDENTIALS_DBUSERNAME: ${ steps.get-credentials-
dbusername.outputs.credentials_dbusername }
      CREDENTIALS_DBPASSWORD: ${ steps.get-credentials-
dbpassword.outputs.credentials_dbpassword }
    steps:
      - name: Waiting for approve deploy
        uses: trstringer/manual-approval@v1.9.0
        timeout-minutes: 15
        with:
          secret: ${ secrets.GH_ACCESS_TOKEN }
          approvers: kh0mka
          minimum-approvals: 1
          issue-title: "[CI/CD] Build infrastructure and deploy web service on
AWS"
          issue-body: "Please approve or deny the launch"
          exclude-workflow-initiator-as-approver: false
      - name: Checkout repository
        uses: actions/checkout@v2
      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v1
        with:
          terraform_wrapper: false
      - name: Initializing Terraform
```

```

    id: init
    run: terraform init -backend-config="bucket=$TF_STATE_BUCKET_NAME" -
backend-config="region=eu-north-1"
    working-directory: ./terraform
- name: Planning Terraform
  id: plan
  run: |-
    terraform plan \
      -var="public_key=$PUBLIC_SSH_KEY" \
      -var="private_key=$PRIVATE_SSH_KEY" \
      -var="key_name=wordpresscid" \
      -out=PLAN
    working-directory: ./terraform
- name: Terraform Apply
  id: apply
  run: terraform apply PLAN
  working-directory: ./terraform

- name: Trying to get EC2 Public IP
  id: set-ip
  run: |
    ip=$(terraform output instance_public_ip | tr -d '[],' | tr -d
'[:space:]')
    echo "instance_public_ip=$ip" >> $GITHUB_OUTPUT
  working-directory: ./terraform

- name: Getting RDS hostname
  id: get-credentials-dbhost
  run: |
    dbhost=$(terraform output credentials_dbhost)
    dbhost=$(echo $dbhost | sed 's/"//g')
    echo "credentials_dbhost=$dbhost" >> $GITHUB_OUTPUT
  working-directory: ./terraform

- name: Getting RDS database name
  id: get-credentials-dbname
  run: |
    dbname=$(terraform output credentials_dbname)
    dbname=$(echo $dbname | sed 's/"//g')
    echo "credentials_dbname=$dbname" >> $GITHUB_OUTPUT
  working-directory: ./terraform

- name: Getting RDS username
  id: get-credentials-dbusername
  run: |
    dbusername=$(terraform output credentials_dbusername)
    dbusername=$(echo $dbusername | sed 's/"//g')
    echo "credentials_dbusername=$dbusername" >> $GITHUB_OUTPUT
  working-directory: ./terraform

- name: Getting RDS password
  id: get-credentials-dbpassword
  run: |
    dbpassword=$(terraform output credentials_dbpassword)
    dbpassword=$(echo $dbpassword | sed 's/"//g')
    echo "credentials_dbpassword=$dbpassword" >> $GITHUB_OUTPUT

```



```

    working-directory: ./terraform
  deploy-app:
    runs-on: ubuntu-latest
    needs: infrastructure-deployment
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Setting SERVER_PUBLIC_IP as env
        run: echo SERVER_PUBLIC_IP=${{ needs.infrastructure-
deployment.outputs.SERVER_PUBLIC_IP }} >> $GITHUB_ENV

      - name: Setting CREDENTIALS_DBHOST as env
        run: echo CREDENTIALS_DBHOST=${{ needs.infrastructure-
deployment.outputs.CREDENTIALS_DBHOST }} >> $GITHUB_ENV

      - name: Setting CREDENTIALS_DBNAME as env
        run: echo CREDENTIALS_DBNAME=${{ needs.infrastructure-
deployment.outputs.CREDENTIALS_DBNAME }} >> $GITHUB_ENV

      - name: Setting CREDENTIALS_DBUSERNAME as env
        run: echo CREDENTIALS_DBUSERNAME=${{ needs.infrastructure-
deployment.outputs.CREDENTIALS_DBUSERNAME }} >> $GITHUB_ENV

      - name: Setting CREDENTIALS_DBPASSWORD as env
        run: echo CREDENTIALS_DBPASSWORD=${{ needs.infrastructure-
deployment.outputs.CREDENTIALS_DBPASSWORD }} >> $GITHUB_ENV

      - name: Connect to EC2 instance via ssh
        env:
          AWS_DEFAULT_REGION: eu-north-1
        uses: appleboy/ssh-action@master
        with:
          host: ${ env.SERVER_PUBLIC_IP }
          username: ubuntu
          key: ${ env.PRIVATE_SSH_KEY }
          envs:
PRIVATE_SSH_KEY,AWS_ACCESS_KEY_ID,AWS_SECRET_ACCESS_KEY,AWS_DEFAULT_REGION,AWS
_REGION,GH_ACCESS_TOKEN,CREDENTIALS_DBHOST,CREDENTIALS_DBNAME,CREDENTIALS_DB
USERNAME,CREDENTIALS_DBPASSWORD
          script: |-
            sudo apt update
            sudo apt install docker.io -y
            sudo apt install docker-compose -y
            git clone https://$GH_ACCESS_TOKEN@github.com/kh0mka/wordpress-ci-
cd.git
            curl -sSLo wordpress-ci-cd/wp-https/nginx/options-ssl-nginx.conf
https://raw.githubusercontent.com/certbot/certbot/master/certbot-
nginx/certbot_nginx/_internal/tls_configs/options-ssl-nginx.conf
            cd wordpress-ci-cd/wp-http/
            echo "CREDENTIALS_DBHOST=$CREDENTIALS_DBHOST" > .env
            echo "CREDENTIALS_DBNAME=$CREDENTIALS_DBNAME" >> .env
            echo "CREDENTIALS_DBUSERNAME=$CREDENTIALS_DBUSERNAME" >> .env
            echo "CREDENTIALS_DBPASSWORD=$CREDENTIALS_DBPASSWORD" >> .env
            sudo docker-compose up -d
            sleep 5
            sudo docker-compose stop

```

```

        cp -R ../wp-https/* .
        sudo docker-compose up -d

# Author: Kiryl Homenok
# File: ../github/workflows/destroy.yml

name: Manually destroy the infrastructure
on: workflow_dispatch

env:
  AWS_ACCESS_KEY_ID: ${ secrets.AWS_ACCESS_KEY_ID }
  AWS_SECRET_ACCESS_KEY: ${ secrets.AWS_SECRET_ACCESS_KEY }
  TF_STATE_BUCKET_NAME: ${ secrets.AWS_TF_STATE_BUCKET_NAME }
  PRIVATE_SSH_KEY: ${ secrets.AWS_SSH_KEY_PRIVATE }
  PUBLIC_SSH_KEY: ${ secrets.AWS_SSH_KEY_PUBLIC }

jobs:
  destroy-infrastructure:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2
      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v1
        with:
          terraform_wrapper: false
      - name: Initializing Terraform
        id: init
        run: terraform init -backend-config="bucket=$TF_STATE_BUCKET_NAME" -
backend-config="region=eu-north-1"
        working-directory: ./terraform
      - name: Destroy Infrastructure
        id: destroy
        run: |-
          terraform destroy --auto-approve \
            -var="public_key=$PUBLIC_SSH_KEY" \
            -var="private_key=$PRIVATE_SSH_KEY" \
            -var="key_name=wordpressci-cd"
        working-directory: ./terraform

# Author: Kiryl Homenok
# File: ../terraform/main.tf

terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
  backend "s3" {
    key = "wordpress-ci-cd"
    bucket = "wordpress-backend-bucket-s3"
    region = "eu-north-1"
  }
}

```

```

provider "aws" {
  region = var.aws_region
}

data "aws_vpc" "default" {
  default = true
}

data "aws_security_group" "default" {
  name     = "default"
  vpc_id = data.aws_vpc.default.id
}

data "aws_internet_gateway" "default" {
  filter {
    name     = "attachment.vpc-id"
    values = [aws_default_vpc.get.id]
  }
}

data "aws_secretsmanager_secret_version" "db_credentials" {
  secret_id = "db-credentials"
}

locals {
  db_credentials =
  jsondecode(data.aws_secretsmanager_secret_version.db_credentials.secret_string)
}

resource "aws_default_vpc" "get" {
  tags = {
    Name = "Default VPC"
  }
}

module "vpc" {
  source = "../modules/vpc"

  create_vpc = false

  azs = ["eu-north-1a"]

  public_gateway_id      = data.aws_internet_gateway.default.id
  public_vpc_id          = aws_default_vpc.get.id
  public_subnets        = ["172.31.49.0/24"]
  public_subnet_names    = ["Public Subnet #1"]
  public_route_table_tags = { "Name" = "Route table of public subnet" }
  map_public_ip_on_launch = true

  private_vpc_id         = aws_default_vpc.get.id
  private_subnets       = ["172.31.50.0/24"]
  private_subnet_names   = ["Private Subnet #1"]
  private_route_table_tags = { "Name" = "Route table of private subnet" }
}

```

```

module "ec2_rds" {
    source = "./modules/sg"

    name          = "ec2-rds-1"
    description    = "Open traffic from all IPv4"
    vpc_id        = module.vpc.default_vpc_id

    ingress_cidr_blocks = ["0.0.0.0/0"]
    ingress_rules       = ["all-all"]

    egress_cidr_blocks = ["0.0.0.0/0"]
    egress_rules       = ["all-all"]
}

module "rds_ec2" {
    source = "./modules/sg"

    name          = "rds-ec2-1"
    description    = "Open inbound traffic from EC2 to RDS"
    vpc_id        = module.vpc.default_vpc_id

    ingress_cidr_blocks = ["172.31.49.0/24"]
    ingress_rules       = ["mysql-tcp"]
}

module "db_computed_source_sg" {
    source = "./modules/sg"

    vpc_id = module.vpc.default_vpc_id

    name          = "RDS-EC2-1"
    ingress_cidr_blocks = ["172.31.49.0/24"]
    egress_cidr_blocks = ["0.0.0.0/0"]
    egress_rules       = ["mysql-tcp"]

    computed_ingress_with_source_security_group_id = [
        {
            rule          = "mysql-tcp"
            source_security_group_id = module.ec2_rds.security_group_id
        }
    ]
    number_of_computed_ingress_with_source_security_group_id = 1
}

data "aws_ami" "getLatestUbuntu" {
    most_recent = true

    filter {
        name     = "name"
        values   = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
    }

    owners = ["099720109477"]
}

```

```

resource "aws_eip_association" "web-srv-eip" {
  instance_id    = aws_instance.public_instance.id
  allocation_id = "eipalloc-0ccfea0274067619e"
}

resource "aws_key_pair" "public_instance_key" {
  key_name    = var.key_name
  public_key  = var.public_key
}

module "cloudwatch" {
  source      = "./modules/cloudwatch"
  subnet_id  = module.vpc.public_subnets[0]
}

resource "aws_instance" "public_instance" {
  ami          = data.aws_ami.getLatestUbuntu.id
  instance_type = "t3.micro"

  key_name          = aws_key_pair.public_instance_key.key_name
  security_groups   = [module.ec2_rds.security_group_id]
  subnet_id        = module.vpc.public_subnets[0]
  availability_zone = "eu-north-1a"
  iam_instance_profile = module.cloudwatch.IAMInstanceProfileRoleName

  connection {
    type      = "ssh"
    host      = self.public_ip
    user      = "ubuntu"
    private_key = var.private_key
    timeout   = "3m"
  }

  tags = {
    Name = "Instance of T3.Micro in Public subnet"
  }
}

resource "aws_db_instance" "rds_instance" {
  vpc_security_group_ids = [module.db_computed_source_sg.security_group_id]
  instance_class          = "db.t3.micro"
  multi_az               = false
  publicly_accessible     = false
  availability_zone       = "eu-north-1a"
  identifier              = var.db_identifier

  engine          = "mysql"
  engine_version  = "5.7"
  allocated_storage = 10
  db_name         = local.db_credentials.dbname

  username = local.db_credentials.username
  password = local.db_credentials.password

  parameter_group_name = "default.mysql5.7"
  skip_final_snapshot  = true
}

```

```

}

# Author: Kiryl Homenok
# File: ./terraform/outputs.tf

output "instance_public_ip" {
  value      = aws_eip_association.web-srv-eip.public_ip
  sensitive = true
}

output "credentials_dbhost" {
  value      = aws_db_instance.rds_instance.address
  sensitive = true
}

output "credentials_dbname" {
  value      = local.db_credentials.dbname
  sensitive = true
}

output "credentials_dbusername" {
  value      = local.db_credentials.username
  sensitive = true
}

output "credentials_dbpassword" {
  value      = local.db_credentials.password
  sensitive = true
}

# Author: Kiryl Homenok
# File: ./terraform/variables.tf

variable "key_name" {
  type = string
}

variable "public_key" {
  type = string
}

variable "private_key" {
  type = string
}

variable "aws_region" {
  type      = string
  default = "eu-north-1"
}

variable "db_identifier" {
  type      = string
  default = "wordpress-db-rds-connection-url"
}

```

```

# Author: Kiryl Homenok
# File: ./terraform/modules/cloudwatch/main.tf

resource "aws_cloudwatch_log_group" "log_group" {
  name = lookup(var.names, "cloudwatch_log_group", "unspecified-name")
  tags = merge(var.tags, lookup(var.tags_for_resource,
"aws_cloudwatch_log_group", {}))
}

resource "aws_iam_policy" "flow_log_policy" {
  name = lookup(var.names, "iam_policy", "unspecified-name")
  path = "/"

  policy = var.resource_policy_json
}

resource "aws_iam_role" "flow_log_role" {
  name = lookup(var.names, "iam_role", "unspecified-name")
  assume_role_policy = var.trust_policy_json
}

resource "aws_iam_policy_attachment" "attach_resource_policy_to_role" {
  name = lookup(var.names, "iam_policy", "unspecified-name")
  policy_arn = aws_iam_policy.flow_log_policy.arn
  roles = [aws_iam_role.flow_log_role.name]
}

resource "aws_flow_log" "flow_log_parameters" {
  iam_role_arn = aws_iam_role.flow_log_role.arn
  log_destination = aws_cloudwatch_log_group.log_group.arn
  traffic_type = "ALL"
  subnet_id = var.subnet_id
}

resource "aws_iam_instance_profile" "FlowLogInstanceProfile" {
  name = lookup(var.names, "instance_profile", "unspecified-name")
  role = aws_iam_role.flow_log_role.name
}

# Author: Kiryl Homenok
# File: ./terraform/modules/cloudwatch/outputs.tf

output "IAMInstanceProfileRoleName" {
  value = aws_iam_instance_profile.FlowLogInstanceProfile.name
}

# Author: Kiryl Homenok
# File: ./terraform/modules/cloudwatch/variables.tf

variable "tags" {
  description = "A map of tags to assign to resources"
  type = map(string)
  default = {}
}

variable "tags_for_resource" {

```

```

description = "A nested map of tags to assign to specific resource types"
type        = map(map(string))
default = {
  aws_cloudwatch_log_group = {
    "Description" = "Created using the cloudwatch module by khomenokkg"
  }
}

variable "names" {
  description = "A nested map of tags to assign to specific resource types"
  type        = map(string)
  default = {
    cloudwatch_log_group = "CloudWatch-LogGroup"
    iam_policy            = "Policy-LogGroup"
    iam_role              = "Role-LogGroup"
    instance_profile      = "FlowLogInstanceProfile"
  }
}

variable "subnet_id" {
  type    = string
  default = ""
}

variable "resource_policy_json" {
  default = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Resource": "*"
    }
  ]
}
EOF
}

variable "trust_policy_json" {
  default = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "vpc-flow-logs.amazonaws.com"
      },

```



```

        "Action": "sts:AssumeRole"
    }
]
}
EOF
}

# Author: Kiryl Homenok
# File: ./wp-https/docker-compose.yml

version: "3.9"

services:
  wordpress:
    container_name: wordpress
    image: wordpress:php8.2-fpm
    restart: always
    env_file:
      - .env
    environment:
      WORDPRESS_DB_HOST: ${CREDENTIALS_DBHOST}
      WORDPRESS_DB_USER: ${CREDENTIALS_DBUSERNAME}
      WORDPRESS_DB_PASSWORD: ${CREDENTIALS_DBPASSWORD}
      WORDPRESS_DB_NAME: ${CREDENTIALS_DBNAME}
    volumes:
      - wordpress_data:/var/www/html

  certbot:
    container_name: certbot
    image: certbot/certbot:latest
    command: certonly --webroot --webroot-path=/var/www/html --email
khomenokkg@gmail.com --agree-tos -d wordpress.diplomadomain.online
    volumes:
      - certbot-etc:/etc/letsencrypt
      - wordpress_data:/var/www/html

  web:
    container_name: webnginx
    image: nginx:latest
    volumes:
      - wordpress_data:/var/www/html
      - ./nginx:/etc/nginx/conf.d
      - certbot-etc:/etc/letsencrypt
    ports:
      - "80:80"
      - "443:443"
    restart: always

volumes:
  wordpress_data:
  certbot-etc:

```

ПРИЛОЖЕНИЕ Б
(обязательное)
Графический материал, поясняющий используемые DevOps-
технологии

Рисунок Б.2 – UML диаграмма развертывания

Структура манифеста Terraform

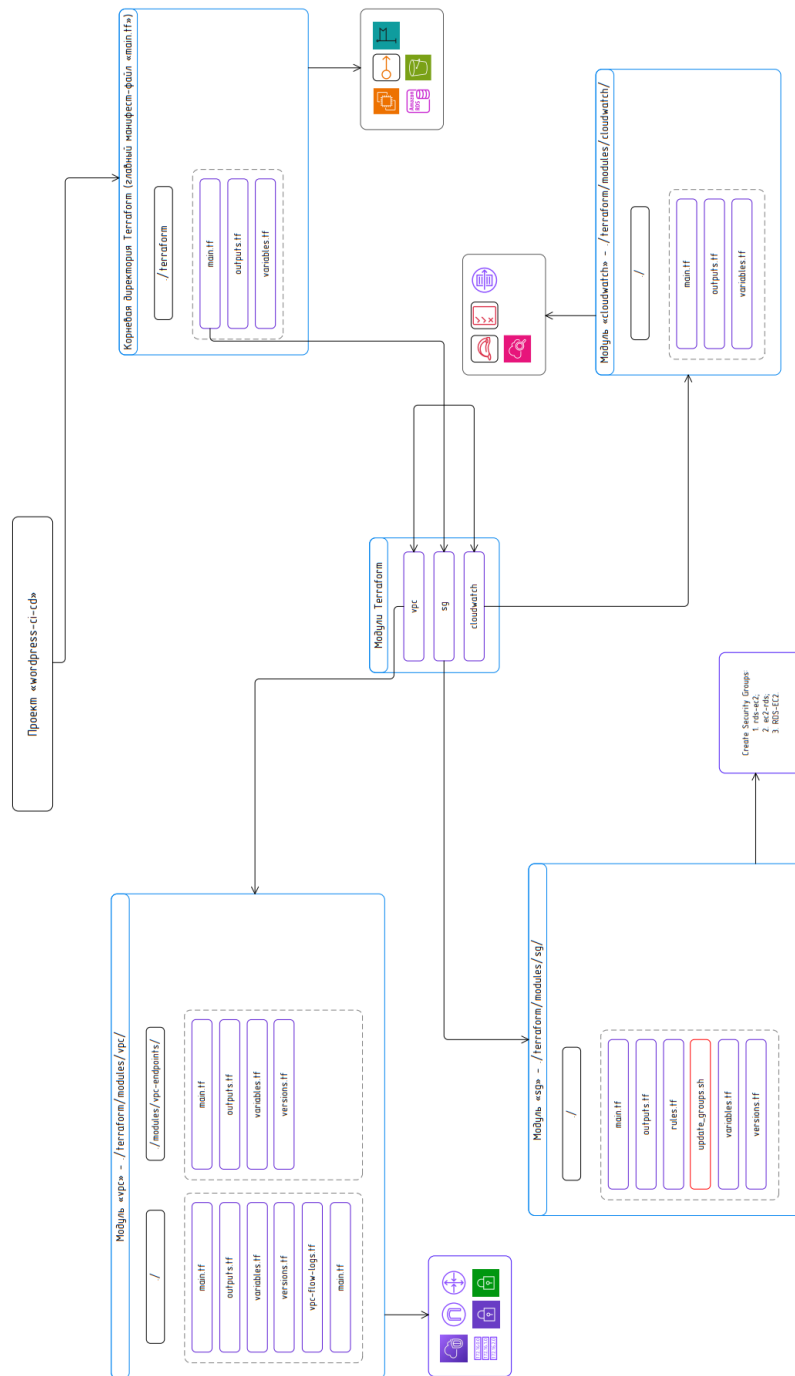


Рисунок Б.3 – Структура манифеста Terraform

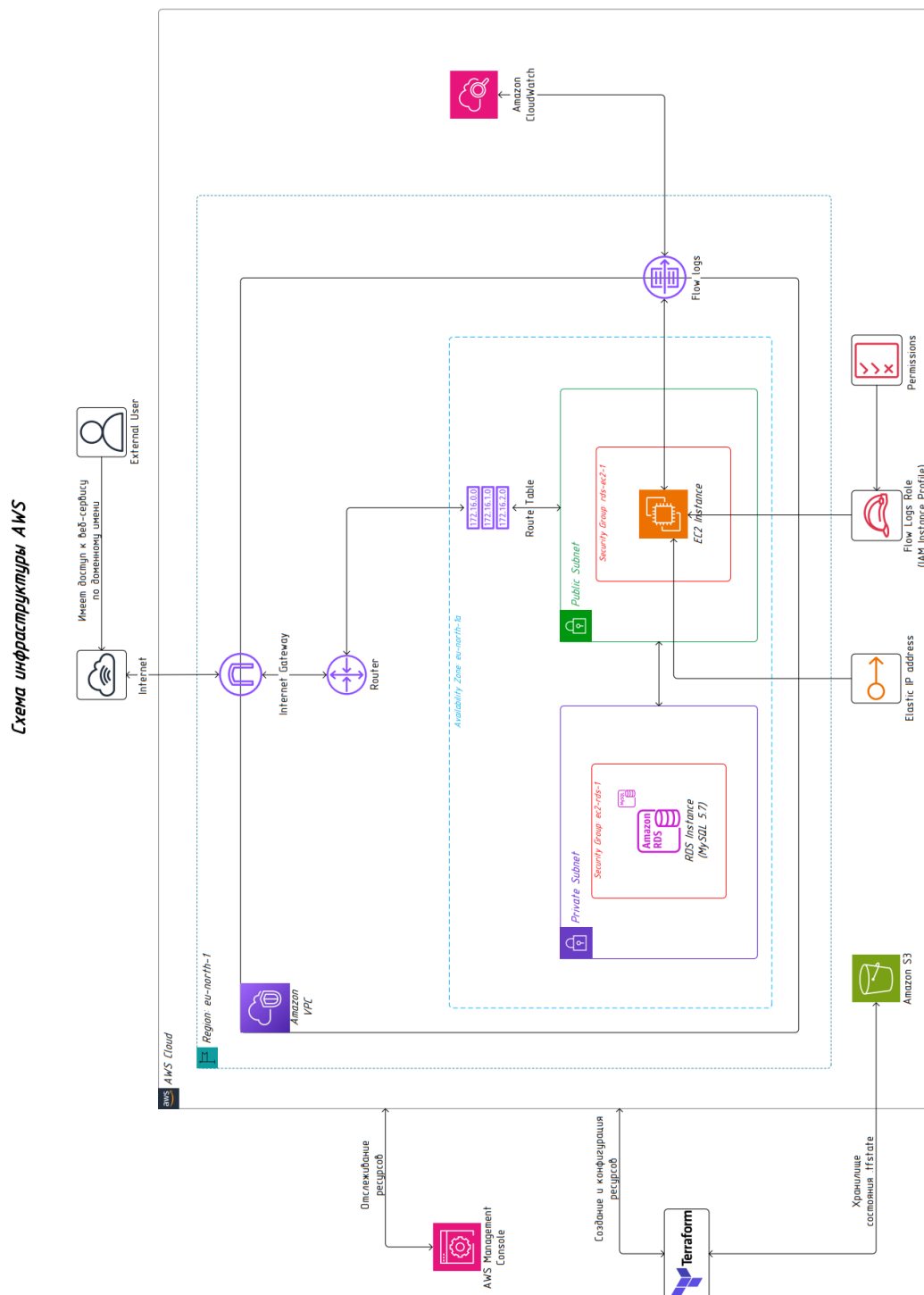


Рисунок Б.4 – Схема инфраструктуры AWS

Графический интерфейс веб-сервиса

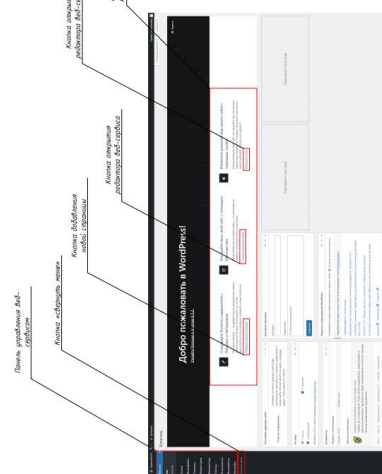


Рисунок 1 – Главная страница консоли WordPress

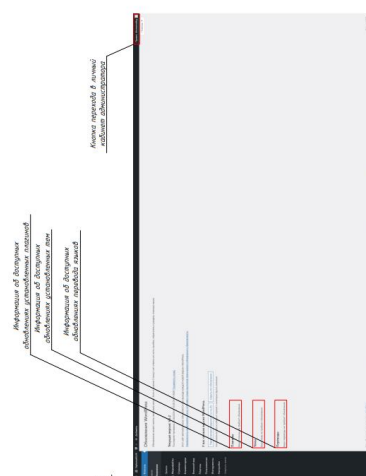


Рисунок 2 – Страница обновлений WordPress

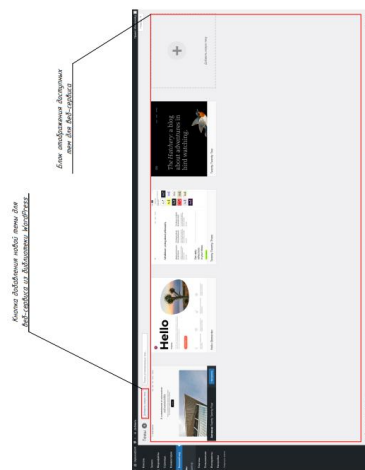


Рисунок 3 – Страница управления темами веб-сервиса

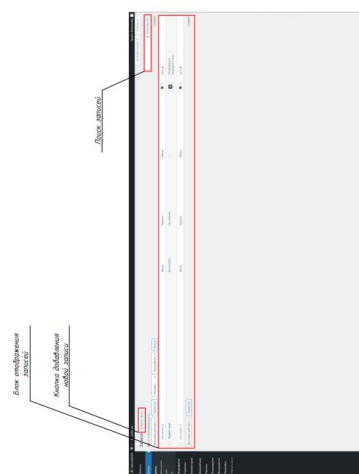


Рисунок 4 – Страница управления записями (блог)



Рисунок 4 – Главная страница веб-сервиса (CMS)

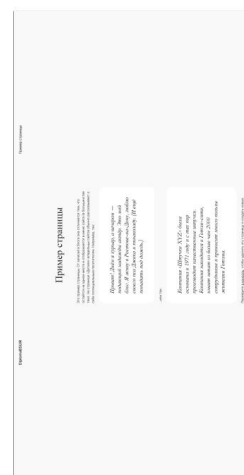


Рисунок 6 – Страница «Пример страницы»