Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра мережних технологій

Магістерська робота

освітній ступінь – магістр

на тему: «**ПОБУДОВА ХМАРНОГО ЗАСТОСУВАННЯ ДЛЯ ВЕБ-ОРІЄНТОВАНОГО ДОСТУПУ ДО ВІРТУАЛЬНИХ ТА ВІДДАЛЕНИХ РОБОЧИХ СТАНЦІЙ**»

Виконав: студент 2-го року навчання,
освітньо-наукової програми
«Інженерія програмного
забезпечення», 121
Земляний Данило Сергійович
Керівник Черкасов Д.І.,
старший викладач, кандидат
технічних наук
Рецензент_____
(прізвище та ініціали)

Магістерська робота захищена
з оцінкою _____
Секретар ЕК _____
«____» _____ 20____ р.

Київ – 2025

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

БД — база даних;
ІТ — інформаційні технології;
ПЗ — програмне забезпечення;
ACM — AWS Certificate Manager;
ALB — Application Load Balancer;
API — Application Programming Interface;
AWS — Amazon Web Services;
CDN — Content Delivery Network;
CIDR — Classless Inter-Domain Routing;
CPU — Central Processing Unit;
CSS — Cascading Style Sheets;
DaaS — Desktop as a Service;
DNS — Domain Name System;
EC2 — Elastic Compute Cloud;
ECR — Elastic Container Registry;
ECS — Elastic Container Service;
EKS — Elastic Kubernetes Service;
HCL — HashiCorp Configuration Language;
HTML5 — HyperText Markup Language 5;
HTTP — Hypertext Transfer Protocol;
HTTPS — Hypertext Transfer Protocol Secure;
IaC — Infrastructure as Code;
IAM — Identity and Access Management;
IP — Internet Protocol;
JSON — JavaScript Object Notation;
KMS — Key Management Services;
LDAP — Lightweight Directory Access Protocol;

MIME — Multipurpose Internet Mail Extensions;

NAT — Network Address Translation;

RDP — Remote Desktop Protocol;

RDS — Relational Database Service;

SaaS — Software as a Service;

S3 — Simple Storage Service;

SPA — Single Page Application;

SSH — Secure Shell;

SSL — Secure Sockets Layer;

TLS — Transport Layer Security;

VDI — Virtual Desktop Infrastructure;

VNC — Virtual Network Computing;

VPC — Virtual Private Cloud;

ВСТУП

Зважаючи на поширення розміщення корпоративних обчислювальних ресурсів у хмарних сервісах та зростання вимог до віддаленої роботи, виникає необхідність у розробці ефективних рішень для організації віддаленого доступу до робочих місць і серверів. Дедалі частіше роботодавці розгортають Virtual Desktop Infrastructure (VDI) для віддаленої роботи. Порівняно з традиційним підходом надання працівникам корпоративних пристроїв, VDI має низку переваг, серед яких [1]:

- Зниження витрат на придбання, обслуговування та модернізацію обладнання.
- Зменшення ризиків інформаційної безпеки за рахунок централізованого управління доступом і впровадження безпекових політик.
- Підвищення мобільності й продуктивності працівників.
- Зручність та гнучкість роботи для користувачів.
- Швидке централізоване впровадження нових технологій
- Можливість масштабування IT-інфраструктури.

Наявні сьогодні рішення для організації доступу до віддалених робочих місць часто є пропрієтарними, вимагають встановлення додаткових програмних компонентів для отримання доступу або не забезпечують достатньої масштабованості, що створює складнощі в експлуатації. Таким чином, виникає потреба в створенні веб-орієнтованого рішення, що дасть змогу працівникам гнучко налаштовувати робочий простір та використовувати для централізованого доступу до робочої інфраструктури будь-який пристрій. Це рішення має легко інтегруватись в існуючу інфраструктуру, бути простим в налаштуванні та надавати можливість масштабування відповідно до розмірів організації.

Виходячи з тенденцій розвитку хмарних технологій та сучасних вимог ринку, метою даної роботи є розробка хмарного веб-орієнтованого, масштабованого

рішення для доступу до віддалених та віртуальних робочих станцій, що може бути легко впроваджене в існуючу інфраструктуру підприємств.

У цій роботі буде проведено аналіз існуючих засобів реалізації веб-орієнтованого доступу до віртуальних та віддалених робочих станцій, їх переваг та недоліків а також оптимальних умов використання. Буде запропоновано оригінальне рішення, для використання в корпоративних умовах. Базовий компонент рішення — код для створення інфраструктури для хмарних сервісів буде розроблено детально.

РОЗДІЛ 1: ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Огляд підходів організації доступу до віддалених робочих місць

У сучасній практиці існує кілька підходів для організації корпоративного доступу до віддалених робочих місць. Найпоширенішими з них є:

- Віртуальна інфраструктура робочих місць (VDI, Virtual Desktop Infrastructure).
- Робочі столи як сервіс (DaaS, Desktop as a Service).
- Програмні засоби віддаленого доступу.
- Рішення, що потребують ручного розгортання.

У цьому розділі здійснено аналіз вказаних підходів та реалізованих на їх основі рішень, визначено їх ключові переваги та обмеження; обґрунтовано доцільність створення альтернативного рішення, здатного усунути наявні недоліки.

1.1.1. Огляд та аналіз Virtual Desktop Infrastructure

VDI є моделлю, у якій користувацькі середовища розгортаються у вигляді віртуальних робочих станцій на центральному сервері або кластері серверів. Доступ до цих середовищ здійснюється дистанційно — через протокол RDP (Remote Desktop Protocol) або інші протоколи відображення. Обчислювальні процеси при цьому виконуються на стороні серверної інфраструктури. Такий підхід забезпечує централізоване управління IT-ресурсами, високий рівень контролю безпеки, а також масштабованість. До найвідоміших комерційних рішень, що реалізують VDI модель, належать VMware Horizon і Citrix Virtual Apps and Desktops.

До переваг VDI відноситься централізація обчислювальних ресурсів, що спрощує адміністрування, резервне копіювання та оновлення ПЗ. Високий рівень інформаційної безпеки досягається завдяки тому, що дані не зберігаються на

кінцевих пристроях і не виходять за межі корпоративної мережі. Окрім цього, VDI дозволяє швидко розгортати нові віртуальні робочі станції та інтегрується із системами автентифікації, такими як Active Directory. Схема роботи VMware Horizon зображена на рисунку 1.1 [2].

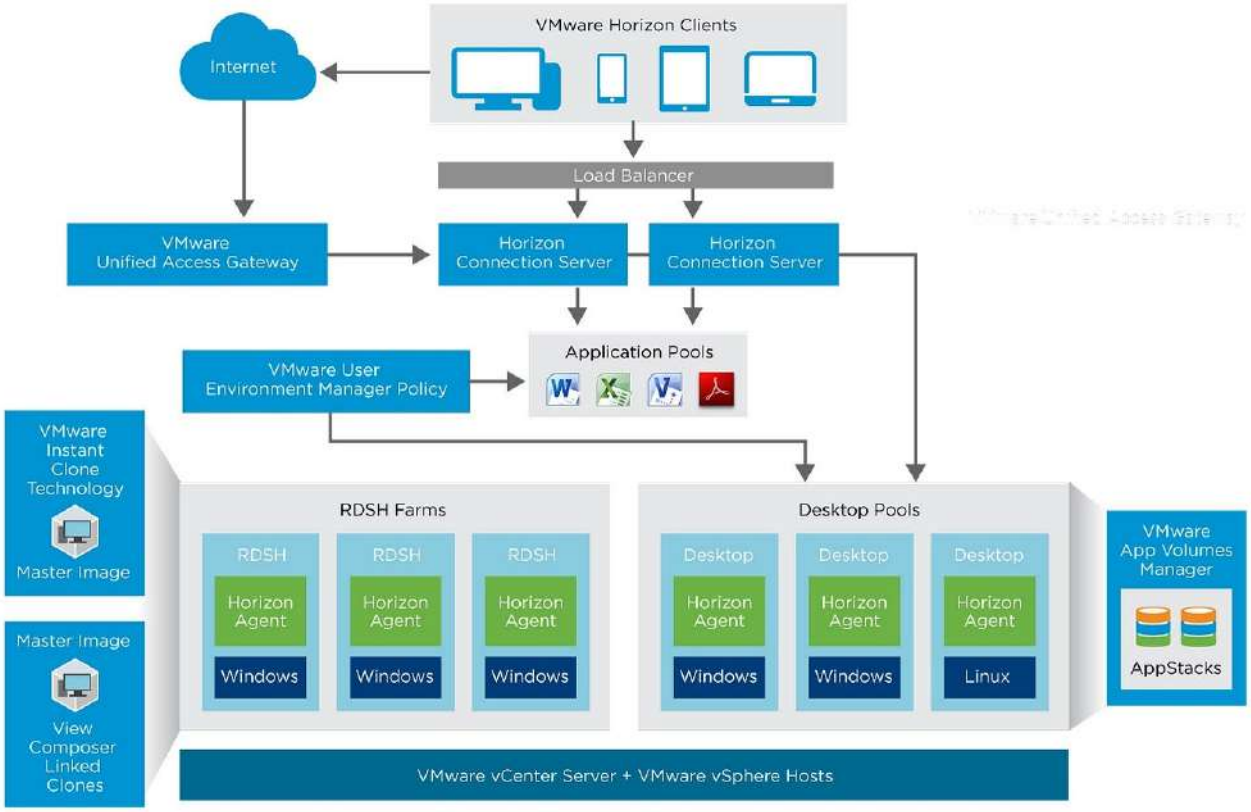

Рисунок 1.1. Схема архітектури VMware Horizon

Попри вказані переваги, рішення на основі VDI характеризуються низкою обмежень. Насамперед, їх впровадження потребує значних вкладень у серверне та мережеве обладнання, системи зберігання даних, гіпервізори, а також у разі використання інфраструктури хмарних провайдерів — відповідні сервіси та ресурси. Більшість комерційних рішень є пропрієтарними та не надають достатньої гнучкості для доступу до віртуальних робочих місць або серверів. Додатково, деякі реалізації потребують інсталяції спеціального клієнтського ПЗ, хоча окремі продукти підтримують доступ через веб-інтерфейс.

Отже, незважаючи на високу керованість і масштабованість, підхід на основі VDI не завжди є оптимальним для реалізації доступу до віртуальних робочих

станцій через складність впровадження та значні фінансові витрати. Також ця модель у більшості випадків вимагає встановлення додаткового ПЗ для доступу до віддалених робочих місць. Крім того, більшість рішень є пропрієтарними та потребують дорогого ліцензування. Це спонукає до пошуку більш економічно ефективних і технологічно гнучких альтернатив.

1.1.2. Огляд та аналіз Desktop as a Service

DaaS є моделлю, у якій віртуальні робочі станції розгортаються та обслуговуються у хмарній інфраструктурі стороннього провайдера. На відміну від VDI, де відповідальність за створення та підтримку інфраструктури покладається на саму організацію, у випадку DaaS ці функції виконує постачальник послуг. Користувачі отримують доступ до своїх віртуальних робочих місць через мережу, найчастіше — за допомогою веб-браузера або спеціального клієнтського застосунку.

До ключових переваг підходу DaaS належать зниження витрат на розгортання та обслуговування фізичної IT-інфраструктури. Оскільки розміщуючи обчислювальні ресурси у хмарній інфраструктурі, підприємство уникає витрат на придбання, модернізацію та адміністрування серверного обладнання. Більшість DaaS-сервісів функціонує за моделлю pay-as-you-go, що дозволяє гнучко керувати витратами залежно від кількості активних користувачів і споживаних ресурсів. Крім того, такі рішення забезпечують високий рівень доступності та стійкості до відмов завдяки використанню хмарних дата-центрів з глобальною інфраструктурою.

Водночас модель DaaS має і певні недоліки. Вона передбачає повну залежність від постачальника послуг, що створює ризик технологічної залежності та ускладнює міграцію до іншого провайдера або власної інфраструктури. Також ця модель не дає змоги налаштувати доступ до віртуальних робочих іншого типу, що зменшує гнучкість даного підходу. Хоча більшість DaaS платформ підтримує

веб-доступ, деякі з них для повноцінного функціонування потребують встановлення додаткового клієнтського програмного забезпечення. Представлення архітектури DaaS показано на рисунку 1.2 [3].

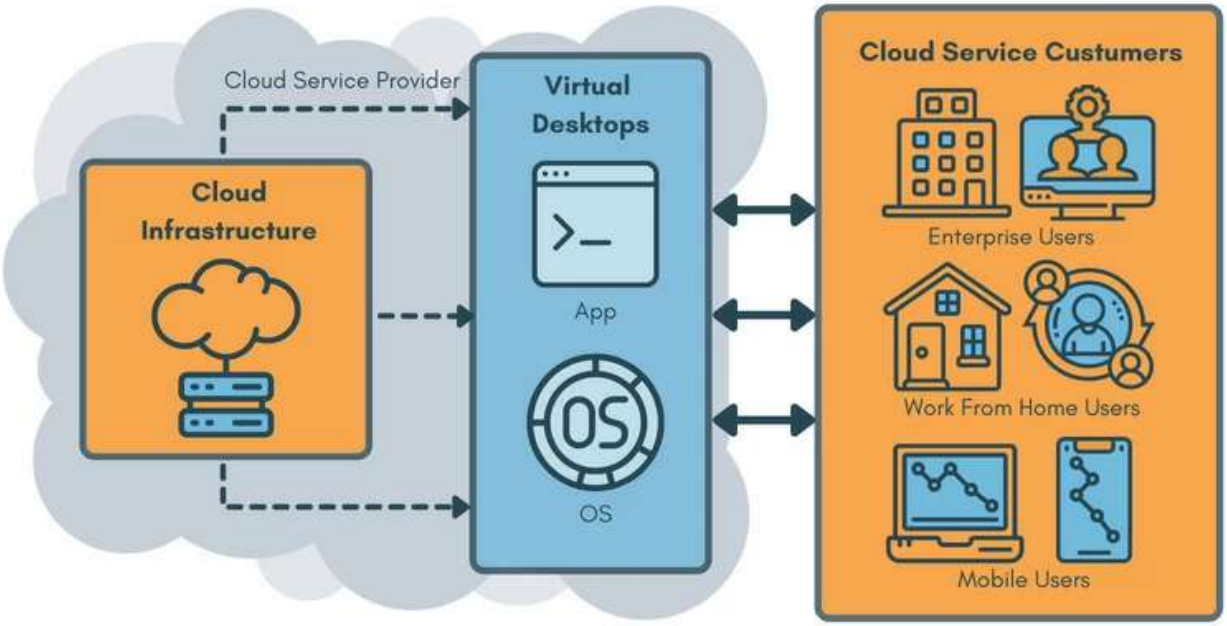

Рисунок 1.2. Візуальне представлення DaaS архітектури

Таким чином, модель DaaS дає змогу організувати доступ до віртуальних робочих місць із мінімальними витратами на інфраструктуру. Водночас її використання потребує врахування ризиків, пов’язаних із залежністю від хмарного провайдера.

1.1.3. Огляд та аналіз програмних засобів віддаленого доступу

Рішення для віддаленого доступу передбачають підключення користувача до вже існуючого фізичного або віртуального пристрою, що знаходиться в іншому місці, з метою віддаленого керування, використання ресурсів або виконання робочих задач. Використання цих рішень краще підходить для гнучкої організації доступу до віртуальних робочих місць.

До рішень у цій категорії належать TeamViewer, AnyDesk, Chrome Remote Desktop, тощо. Вони широко використовуються як у малому бізнесі, так і в

корпоративному середовищі завдяки простоті використання, швидкому налаштуванню та підтримці багатьох операційних систем.

Головною перевагою таких рішень є мінімальні вимоги до інфраструктури: користувачеві достатньо встановити клієнт або серверну частину програми для встановлення з’єднання. Більшість сучасних платформ підтримують з’єднання через браузер або веб-портал, що дозволяє реалізувати доступ навіть без встановлення додаткового ПЗ на стороні клієнта. Це забезпечує певний рівень веб-орієнтованості, особливо у випадках використання рішень із підтримкою HTML5 (наприклад, TeamViewer Web Client або AnyDesk Web Access) [4].

Проте такі інструменти мають суттєві обмеження. Ці рішення є пропрієтарними ти вимагають ліцензування. Також те, що ці послуги надає зовнішній постачальник, отримати контроль над інфраструктурою доступу до віртуальних робочих місць неможливо. Ще однією проблемою є питання безпеки та контролю. Оскільки з’єднання часто йдуть через зовнішні сервери провайдера (наприклад, TeamViewer або AnyDesk), організація втрачає повний контроль над трафіком, що може створювати ризики витоку конфіденційної інформації. Для організацій, які мають суворі вимоги до захисту даних, це є критичним обмеженням.

Таким чином, рішення для віддаленого доступу можуть бути ефективними для малого масштабу або тимчасового використання завдяки простоті налаштування та мінімальним інфраструктурним вимогам. Проте в контексті побудови масштабованого, безпечного та контрольованого хмарного рішення для доступу до робочих місць вони виявляються недостатніми.

1.1.4. Огляд та аналіз рішень що потребують ручного розгортання

Окрему категорію серед засобів для організації доступу до віртуальних робочих місць становлять рішення які потребують самостійного розгортання та налаштування на довільній інфраструктурі. Їх особливістю є те, що вони часто є

рішеннями з відкритим кодом, не вимагають придбання ліцензій для комерційного використання та можуть вільно модифікуватися під конкретні потреби організації. Вони є привабливими для організацій, які прагнуть знизити витрати та отримати повний контроль над інфраструктурою. До найпопулярніших рішень цього типу належать Apache Guacamole, XRDP, а також сервери VNC (Virtual Network Computing), зокрема TightVNC, TigerVNC та інші [5].

Одним із таких рішень є Apache Guacamole — веб-орієнтована платформа, яка дозволяє здійснювати доступ до віртуальних робочих столів через браузер без потреби у встановленні клієнтського програмного забезпечення. Guacamole підтримує основні протоколи віддаленого з'єднання, зокрема RDP, VNC та SSH, і може бути інтегрована з LDAP, базами даних користувачів або зовнішніми системами автентифікації. Завдяки використанню HTML5-клієнта вона забезпечує повноцінний доступ до віддалених робочих середовищ з будь-якого пристрою, що має браузер [6].

Архітектура Apache Guacamole представлена на рисунку 1.3.

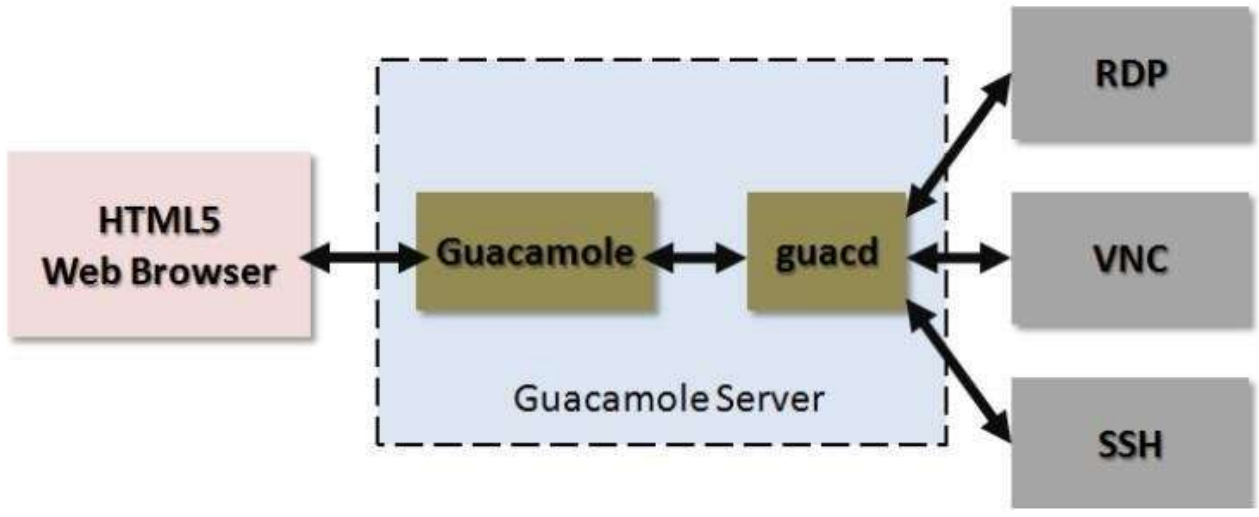

Рисунок 1.3. Архітектура Apache Guacamole

Переваги таких рішень полягають у відкритості коду, гнучкості налаштувань та високому рівні контролю над усіма компонентами системи. Вони не прив'язані до конкретного хмарного провайдера або вендора, що дозволяє будувати

незалежну інфраструктуру. Крім того, використання стандартних протоколів (RDP, VNC, SSH) забезпечує сумісність з широким спектром операційних систем, включаючи Linux і Windows.

Разом із тим, використання рішень з відкритим кодом має і певні обмеження. Рішення вимагають глибших технічних знань для розгортання, конфігурації та підтримки. Крім цього, у базових реалізаціях таких систем часто відсутні готові засоби для автоматичного масштабування або балансування навантаження, що потребує додаткових зусиль у випадку зростання кількості користувачів.
Таким чином, рішення з відкритим кодом є альтернативою для організацій, які шукають економічно ефективні, веб-орієнтовані та незалежні від постачальника підходи до реалізації віртуальних робочих місць. Їх використання доцільне у випадках, коли є необхідність побудови власного рішення доступу до віддалених робочих місць або розгортання внутрішнього сервісу доступу з високим рівнем контролю та безпеки.

1.2. Порівняння проаналізованих підходів

Таким чином, було оглянуто наявні підходи до організації доступу до віддалених робочих місць, визначено їх переваги та недоліки. З огляду на отриману інформацію, можна зробити висновок що рішення, яке одночасно не буде пропрієтарним для конкретної платформи, матиме простий процес розгортання та налаштування локально або в хмарній інфраструктурі, надаватиме можливість використовувати будь які фізичні або віртуальні машини та матиме змогу масштабуватись та балансувати навантаження наразі на ринку не представлене, що обґрунтовує необхідність створення нового рішення яке може виправити дані недоліки. Результати огляду представлені в таблиці 1.1. Описані у таблиці підходи порівнюються за такими критеріями:

- Тип рішення
- Контроль над інфраструктурою

- Залежність від постачальника
- Веб-орієнтований доступ
- Доступ до довільних робочих станцій.
- Масштабування
- Балансування навантаження

Таблиця 1.1. Порівняльна характеристика підходів для організації доступу до віддалених робочих станцій

Критерій	VDI	DaaS	Програмні засоби віддаленого доступу	Apache Guacamole
Тип рішення	Локальне або хмарне	Хмарне	Локальне або хмарне	Локальне або хмарне
Контроль над інфраструктурою	Крім ПЗ яке контролює віртуальні станції	Відсутній	Не можна застосувати	Повний
Залежність від постачальника	Відсутня для інфраструктури але висока для ПЗ	Висока	Висока	Відсутня
Веб-орієнтований доступ	Частково	Переважно	Частково	Повний
Доступ до довільних робочих станцій	Відсутній	Відсутній	Наявний	Наявний

Масштабування	Забезпечується ПЗ	Забезпечується провайдером	Не можливе	Обмежене
Балансування навантаження	Забезпечується ПЗ	Забезпечується провайдером	Не можливе	Обмежене

1.3. Висновки до Розділу 1

Проаналізувавши підходи до організації доступу до віддалених робочих місць, було встановлено, що одним із найефективніших рішень для побудови веб-орієнтованого доступу є Apache Guacamole. Це функціональна система, яка підтримує веб-орієнтований доступ до віддалених робочих місць через популярні протоколи без необхідності встановлення додаткового програмного забезпечення на стороні користувача. Guacamole здатне забезпечити централізований доступ до різних середовищ — як фізичних, так і віртуальних машин. Також підтримує інтеграцію з різними типами інфраструктури, зокрема хмарними.

Водночас, попри свої переваги, Apache Guacamole має низку обмежень, які ускладнюють його практичне використання в умовах динамічного масштабованого середовища. До основних недоліків належать складність розгортання, необхідність ручного налаштування доступу та підключень, а також відсутність вбудованих механізмів масштабування та балансування навантаження, що є обмеженням для великих корпоративних структур.

У межах цієї роботи було розроблено повноцінне хмарне рішення на основі Apache Guacamole, що вирішує вказані обмеження за рахунок автоматизації розгортання, масштабування, балансування навантаження та централізованого управління конфігурацією.

Запропоноване рішення особливо ефективне у корпоративних умовах, зокрема для підприємств, які мають велику кількість віртуальних робочих місць або потребують організації віддаленої роботи. Наприклад, у випадку IT-компанії,

що надає своїм співробітникам або клієнтам доступ до віддалених робочих станцій, дане рішення дозволяє централізовано керувати усіма підключеннями, масштабувати інфраструктуру за навантаженням та забезпечити безпечний доступ без встановлення додаткових клієнтів.

РОЗДІЛ 2: РОЗРОБКА АРХІТЕКТУРИ РІШЕННЯ

2.1. Опис вимог до рішення

У даному розділі сформульовано основні вимоги , які визначають очікувану поведінку, можливості та характеристики розробленого рішення, з поясненням як їх виконано, використовуючи відповідні сервіси та підходи. Також наведено поверхневий опис концепції рішення і детально описано архітектуру модулів, включаючи веб-інтерфейс, серверну частину та базу даних.

Основною вимогою до рішення є можливість надавати користувачам веб-орієнтований доступ до віддалених робочих місць. Цю вимогу виконано використанням Apache Guacamole — сервісу який за надає доступ до віддалених робочих місць використовуючи HTML5-інтерфейс. Також сервіс надає змогу організувати автентифікацію, авторизацію та гранулювання доступу користувачів до відповідних віддалених робочих станцій.

Рішення має бути легким в розгортанні та налаштуванні. Цю вимогу виконано через використання підходів «Інфраструктура як код» (Infrastructure as Code, IaC) [7]. IaC рішення Terraform, використане в рішенні, дозволяє керувати конфігурацією та розгортанням інфраструктури в хмарних сервісах. Інструмент дозволяє безпечно та автоматизовано розгортати необхідну інфраструктуру через декларативний опис компонентів. Для того щоб спростити налаштування, при розгортанні, на вхід Terraform отримує конфігураційний файл, у якому описано робочі місця, дані доступу до них та користувачів з дозволами на відповідні робочі місця. Рішення автоматично обробляє конфігураційний файл та створює відповідні записи в інфраструктурі, які згодом використовуються Apache Guacamole для організації доступу. Також використання Terraform дозволить в майбутньому розширити рішення надавши підтримку інших хмарних сервісів, гібридних хмар та специфічних інструментів.

Рішення має бути розгорнуто в хмарі. Ця вимога виконується використанням хмарного сервісу Amazon AWS, який надає низку ресурсів завдяки яким можна досягнути створення необхідної інфраструктури та забезпечити виконання вимог [8].

Рішення має мати можливість гнучко масштабуватись відповідно до кількості користувачів та рівня навантаження, використовуючи мінімум необхідних ресурсів для забезпечення роботи сервісу. Цю вимогу виконано контейнеризацією екземплярів Apache Guacamole в Docker контейнери, які зберігаються в репозиторії AWS Elastic Container Registry (ECR) [9] та використанням сервісу Amazon Elastic Container Service (ECS) [10]. ECS дозволяє запускати та автоматично масштабувати контейнери відповідно до навантаження. Так як вимоги архітектури контейнеризації невеликі (необхідність горизонтально масштабувати один образ контейнеру) то ECS підходить для рішення, бо є простим варіантом імплементації на відміну від Elastic Kubernetes Service (EKS) так, як EKS потребує більше сервісів та складніших налаштувань, що призводить до підвищеного використання ресурсів. При ускладненні рішення в майбутньому не виключено перехід на використання сервісів Kubernetes [11]. Як обчислювані ресурси використовується AWS Fargate — сервіс безсерверних обчислень який дає змогу гнучко виділяти процесорний час та пам’ять для контейнерів. Це забезпечить максимально ефективне використання ресурсів та зменшить витрати коштів через мінімізацію простою серверів яке майже невідворотнє при використанні віртуальних машин EC2 [12].

Рішення має рівномірно розподіляти навантаження на доступні ресурси. Ця вимога виконується використанням AWS Application Load Balancer (ALB) [13]. Це рішення дозволяє автоматизувати розподілення трафіку на доступні ресурси та має вбудовану інтеграцію з ECS, що дозволить просто налаштувати масштабування контейнерів відповідно до навантаження та рівномірно розподілити запити користувачів.

Рішення має бути стійким до відмов. Стійкість до відмов досягається декількома способами. Для користувацького веб-інтерфейсу використовується Single Page Application (SPA) [14], який статично зберігається в Amazon S3 [15] та доставляється користувачам за допомогою AWS CloudFront [16]. CloudFront забезпечує балансування навантаження, географічне покриття, кешування та стійкість до відмов. Стійкість до відмов серверної частини рішення досягається функціями ECS. Так як контейнери Apache Guacamole не зберігають жодних станів то їх можна створювати та видаляти без шкоди для доступності сервісу. ECS має функцію Auto Healing, яка при виявленні зупинки або відмови одного з контейнерів призупиняє його доступність через балансувальник навантаження, видаляє його та створює новий який після успішного запуску знову стає доступним для запитів. Стійкість до відмов бази даних в даній архітектурі не передбачена але може бути додана в майбутньому шляхом створення бекапів бази даних та увімкнення реплікації з репліками налаштованими тільки на читання.

У даній архітектурі не реалізоване географічне покриття для серверної частини та бази даних. Для досягнення георепліkації серверної частини в майбутньому необхідно додати ECS кластери в інших регіонах та зонах доступності. За допомогою Amazon Route 53 запити від користувачів можна направляти до різних кластерів в різних географічних зонах базуючись на розташуванні користувача або розрахунках затримки запитів до доступних кластерів [17]. Георепліkацію бази даних можна організувати схожим способом з організацією стійкості до відмов, тільки репліки можна розташувати в різних регіонах.

Рішення має забезпечувати захист переданих і збережених даних, обмеження доступу до інфраструктури та контроль прав доступу користувачів. Безпека досягається багатьма підходами. Рішення розгортається в хмарному середовищі, провайдер якого забезпечує захист даних та доступність. Всі підключення користувачів відбувають через HTTPS протокол. ALB забезпечує TLS-термінацію шифруючи весь трафік між клієнтом та сервером сертифікатами, які зберігаються

в AWS Certificate Manager (ACM) [18]. Дані про конфігурацію зберігаються в базі даних AWS Relational Database Service (RDS) який дозволяє увімкнути автоматичне шифрування всіх збережених даних. База даних розгортається в приватній підмережі доступ до якої мають тільки екземпляри Apache Guacamole [19]. Трафік контейнерів можливий тільки через балансувальник навантаження шляхом налаштування груп безпеки. Гранульований доступ користувачів до доступних для них робочих місць реалізується системою автентифікації та авторизації Apache Guacamole.

2.2. Опис архітектури рішення

Основою концепції є повністю автоматизоване та масштабоване рішення, яке забезпечує доступ користувачів до віртуальних або фізичних робочих місць через веб-інтерфейс без використання клієнтського ПЗ. Інфраструктура реалізована за допомогою підходу «Інфраструктура як код», що гарантує повторюваність, швидкість розгортання та легкість адміністрування.

2.2.1. Концепція рішення

Користувач взаємодіє з системою через захищений веб-інтерфейс, де проходить автентифікацію та авторизацію. Після успішного входу він отримує доступ до призначеного йому робочого місця. Конфігурація доступу визначається під час створення інфраструктури та зберігається у базі даних.

Конфігураційні дані (адреси віддалених робочих місць, протоколи доступу, облікові дані) визначаються у вигляді окремого конфігураційного файлу, що подається на вхід інструменту інфраструктури як коду. У процесі розгортання ця конфігурація використовується для створення записів у базі даних, дані в якій зберігаються у зашифрованому вигляді. Таким чином, на момент запуску сервісу інфраструктура вже містить усі необхідні дані для надання доступу користувачам,

без потреби у додатковій ручній ініціалізації або завантаженні конфігурацій після розгортання.

Основою серверної частини рішення є Apache Guacamole — шлюз, який забезпечує веб-орієнтований доступ до робочих місць через HTML5-інтерфейс. Guacamole підтримує широкий спектр протоколів віддаленого доступу, що робить його гнучким та універсальним для використання в різних сценаріях. Екземпляри Apache Guacamole розгортаються як контейнери. Це дозволяє динамічно масштабувати потужності залежно від кількості одночасних користувачів або загального навантаження системи. Запити від користувачів балансуються за допомогою балансування навантаження, яке також забезпечить шифрування з'єднань за допомогою сертифікатів SSL/TLS, отриманих через сховище сертифікатів.

Таким чином, ця концепція дозволяє створити надійний, масштабований та безпечний сервіс для організації веб-орієнтованого доступу до віддалених робочих місць із мінімальними вимогами до користувацького обладнання та програмного забезпечення, а також забезпечує повну прозорість управління інфраструктурою завдяки принципам IaC. Загалом, рішення складається з трьох основних компонентів, інфраструктура яких створюється та налаштовується за допомогою Terraform:

1. Веб-інтерфейс (Frontend) — забезпечує автентифікацію користувачів та взаємодію з системою через браузер.
2. Серверна частина (Backend) — представлена Apache Guacamole, який відповідає за веб-орієнтований віддалений доступ до призначених користувачам робочих місць.
3. База даних (Database) — містить заздалегідь визначені конфігурації підключень.

Діаграма архітектури високого рівня представлена на рисунку 2.1.

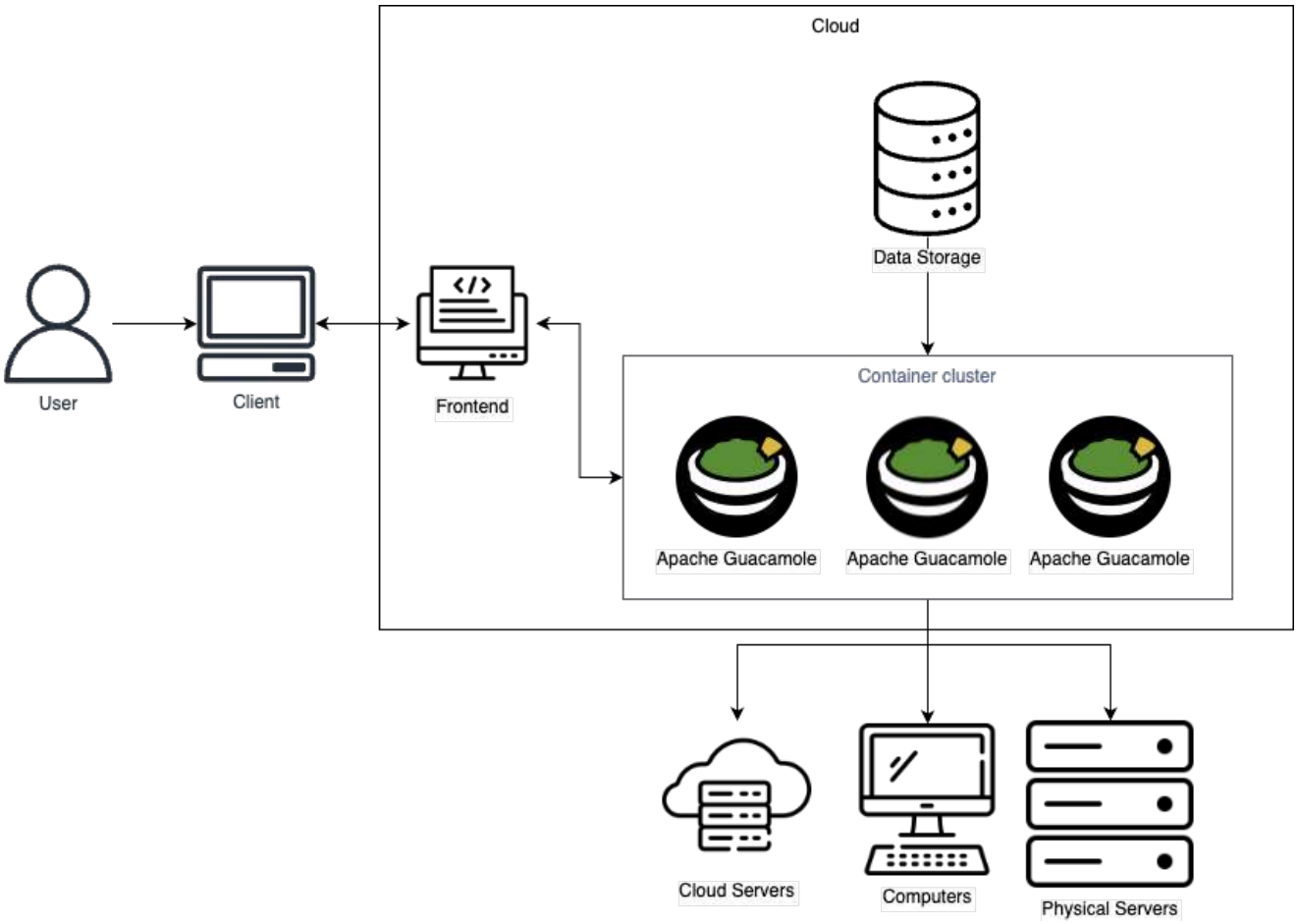

Рисунок 2.1. Діаграма архітектури високого рівня

2.2.2. Архітектура веб-інтерфейсу

Веб-інтерфейс рішення реалізується як односторінковий веб-застосунок, що розробляється з використанням JavaScript фреймворку Vue.js. Основним завданням цієї компоненти є забезпечення інтерактивного та зручного користувацького інтерфейсу для доступу до віддалених робочих місць. Через веб-інтерфейс користувачі здійснюють автентифікацію, переглядають доступні їм підключення, що були заздалегідь визначені під час створення інфраструктури, та ініціюють з’єднання з обраним середовищем через шлюз Apache Guacamole.

Зібрані артефакти застосунку (файли HTML, JavaScript, CSS, тощо) зберігаються у сховищі Amazon S3, яке виступає джерелом статичного контенту. Для забезпечення безпечної, надійної та масштабованої доставки ресурсів

використовується служба Amazon CloudFront, яка виступає в ролі CDN (Content Delivery Network), забезпечує TLS-термінацію за допомогою сертифікатів ACM та дозволяє використовувати додаткові механізми безпеки, такі як обмеження HTTP-заголовків, кешування, балансування навантаження та контроль доступу до ресурсу. Доменне ім’я веб-сайту обслуговується за допомогою Amazon Route 53, що забезпечує просту інтеграцію з CloudFront та можливість автоматизованого управління DNS-записами. Архітектурна діаграма веб-інтерфейсу представлена на рисунку 2.2.

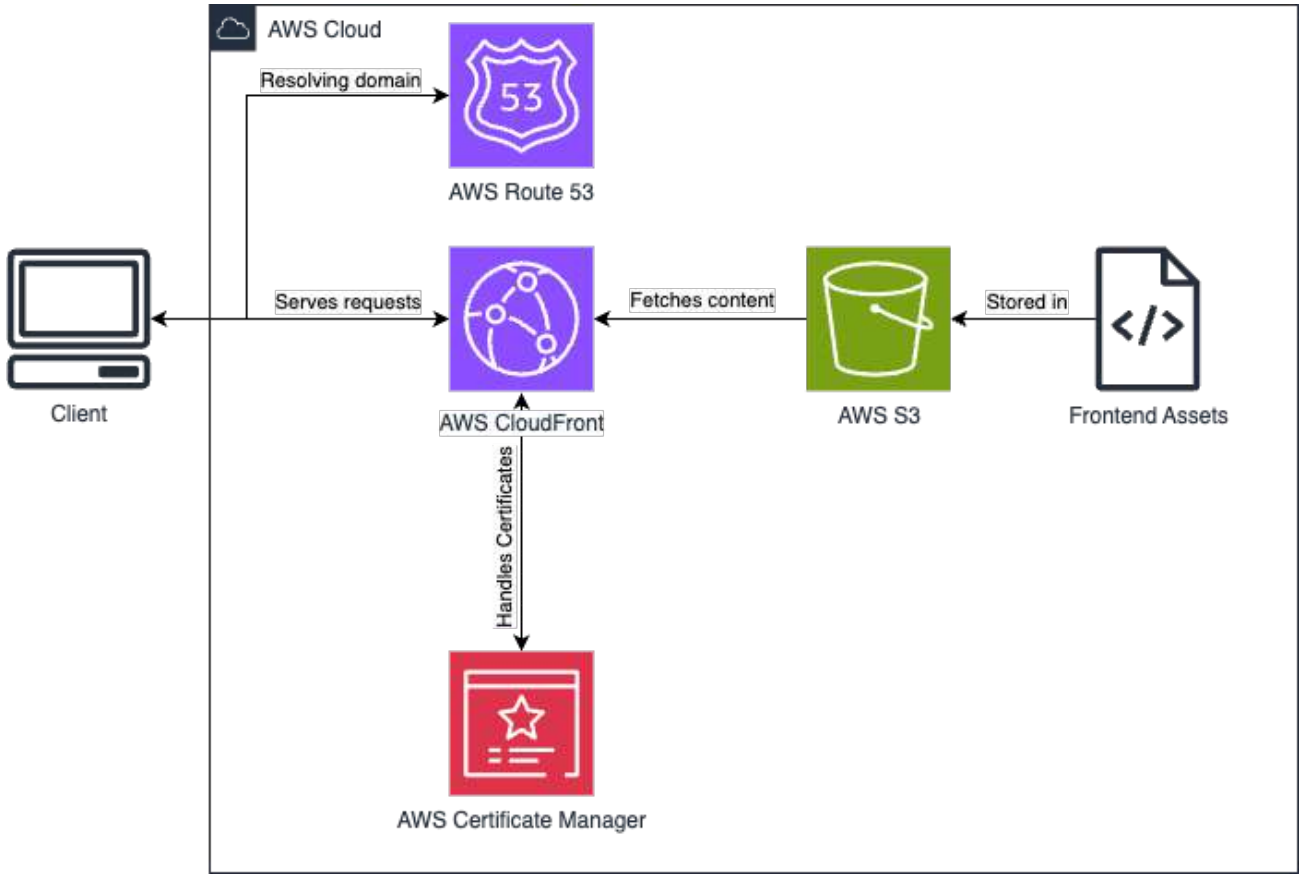

Рисунок 2.2. Архітектурна діаграма веб-інтерфейсу

Архітектура веб-інтерфейсу побудована відповідно до принципів безсерверної моделі. Вона не потребує окремого серверного програмного забезпечення або динамічного бекенду для обробки запитів — усі взаємодії здійснюються безпосередньо між клієнтом та відповідними API Apache Guacamole. Користувач вводить свої облікові дані у форму входу, після чого веб-інтерфейс

надсилає запит на сервер Guacamole, отримує токен сесії та з його допомогою виконує запити для отримання списку підключень або ініціює сесію.

Таким чином, веб-інтерфейс є масштабованим, безпечним і автономним елементом рішення, який забезпечує зрозумілу точку входу для користувачів та інтегрується з серверною частиною системи для організації повноцінного веб-орієнтованого доступу до віддалених робочих середовищ.

2.2.3. Архітектура серверної частини

Серверна частина рішення побудована на основі Apache Guacamole — веб-орієнтованого шлюзу для організації доступу до віддалених робочих місць. Основним завданням серверної частини є організація та управління віддаленими сесіями на основі заздалегідь визначених конфігурацій, а також забезпечення масштабованості та високої доступності сервісу.

Горизонтально масштабовані екземпляри Apache Guacamole розгортаються у вигляді контейнеризованих застосунків у межах AWS ECR. Це дозволяє забезпечити масштабування в залежності від навантаження — кількості одночасних користувачів або загальної кількості активних сесій. Контейнери Apache Guacamole розгортаються у середовищі AWS Fargate, що забезпечує автоматичне масштабування без необхідності адміністрування фізичних серверів. Для контейнеризації використовується Docker. Образи контейнерів зберігаються в AWS Elastic Container Registry.

Усі екземпляри Guacamole не зберігають внутрішній стан і працюють із централізованим джерелом конфігурацій у базі даних. У базі даних зберігаються всі записи необхідні для підключень, автентифікації користувачів, права доступу до віддалених робочих місць та інші службові параметри. Це дозволяє кожному екземпляру Guacamole отримувати актуальні дані в режимі реального часу та забезпечує збереження сесій незалежно від того, на який сервер було направлено користувача. Архітектура серверної частини представлена на рисунку 2.3.

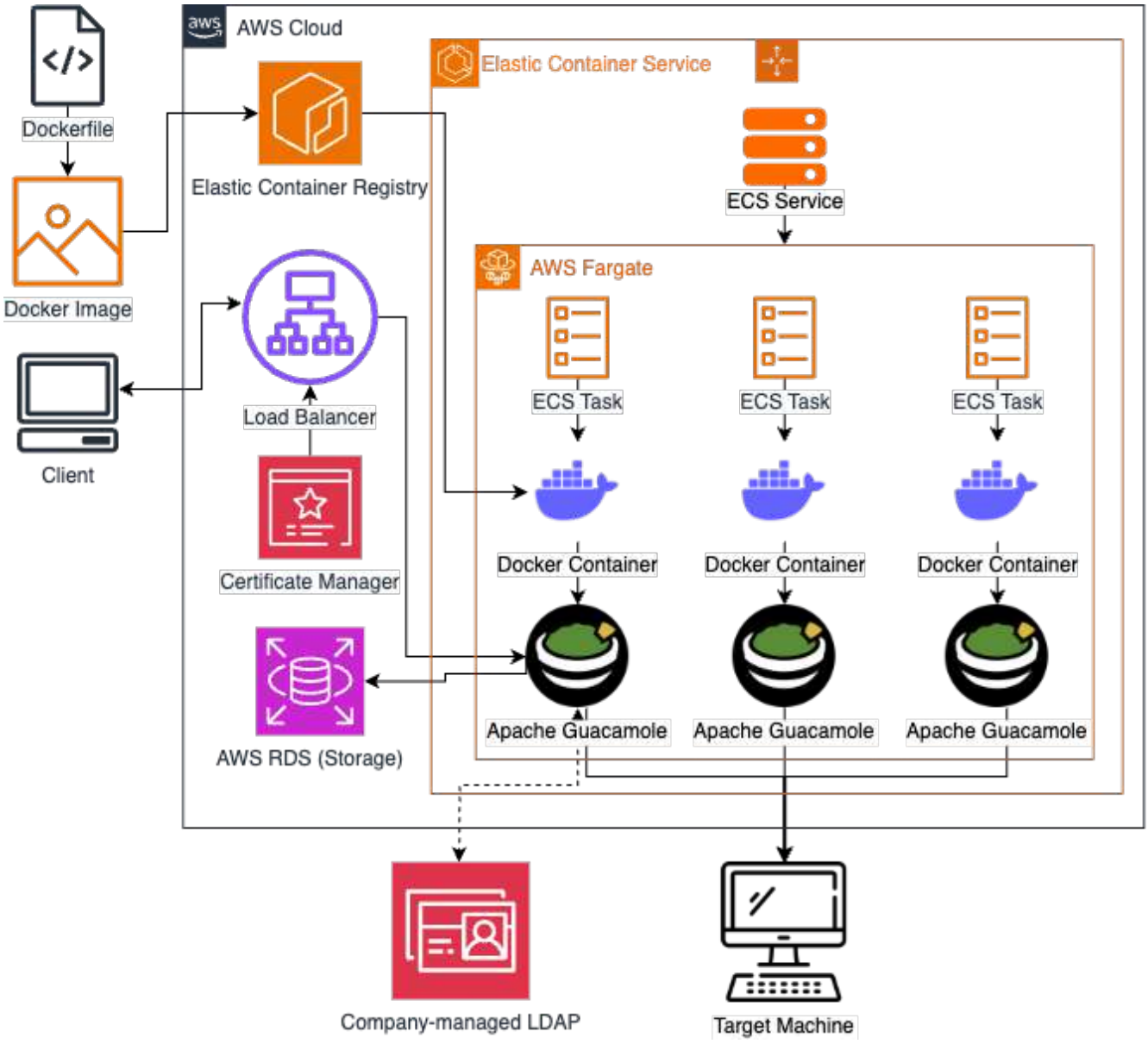

Рисунок 2.3. Архітектура серверної частини

Доступ користувачів до серверної частини здійснюється через Application Load Balancer. ALB виконує функцію балансування навантаження, рівномірно розподіляючи вхідні HTTPS-запити між доступними екземплярами Guacamole. Крім того, ALB забезпечує TLS-термінацію, використовуючи сертифікати, отримані через ACM. Це гарантує шифрування трафіку між клієнтським браузером і сервісом, підвищуючи рівень безпеки всієї системи.

Серверна частина взаємодіє з веб-інтерфейсом через API Apache Guacamole. Клієнтська частина надсилає запити на автентифікацію користувачів, та ініціалізацію сесій. Завдяки такій моделі взаємодії забезпечується чіткий розподіл

відповідальності між клієнтом та сервером, що підвищує масштабованість та полегшує подальший розвиток системи.

Таким чином, серверна частина рішення є масштабованою, безпечною, стійкою до відмов і легкою в адмініструванні та відновленні завдяки використанню принципів інфраструктури як коду.

2.2.4. Архітектура бази даних

База даних відповідає за збереження усієї необхідної інформації для роботи системи віддаленого доступу: облікових записів користувачів, конфігурацій підключень до віддалених робочих місць, параметрів підключення та службових даних Guacamole. В основі бази даних використовується реляційна база даних MySQL яка розміщена на Amazon Relational Database Service. Вона забезпечує централізоване, масштабоване та захищене зберігання даних, необхідних для функціонування серверної частини Apache Guacamole. Вибір RDS зумовлений тим, що в майбутньому рішення може отримати більш складний веб-інтерфейс який додасть можливість реєструвати компанії, керувати розгортанням та ресурсами, оформлювати підписки і так далі. Apache Guacamole не вимагає такого складного рішення і RDS було обрано зважаючи на те, що в майбутньому в цьому може з’явитись необхідність.

База даних створюється під час розгортання інфраструктури за допомогою Terraform, де також здійснюється початкове заповнення таблиць згідно із конфігураційним файлом. При цьому облікові дані, адреси віддалених робочих станцій, протоколи доступу та права користувачів вносяться безпосередньо до відповідних таблиць Guacamole. Всі дані зберігаються у зашифрованому вигляді завдяки вбудованим можливостям Amazon RDS. База даних розміщується у приватній підмережі в межах віртуальної приватної мережі, що забезпечує захищений доступ до даних без прямого виходу в Інтернет.

Інтеграція бази даних із серверною частиною Guacamole у AWS ECS забезпечується за допомогою приватного мережевого з'єднання в межах VPC. Контейнери Guacamole під час старту використовують змінні середовища, передані у Task Definition ECS Service, які містять інформацію для підключення до RDS: ім'я хоста бази даних, порт, ім'я користувача, пароль та назву бази. Безпека підключення гарантується шляхом налаштування правил безпеки, які дозволяють лише необхідний мережевий трафік між ECS Tasks і RDS.

Guacamole зчитує дані про користувачів та підключення безпосередньо з бази даних у режимі реального часу. Завдяки такій інтеграції, при запуску або масштабуванні нових екземплярів контейнерів Guacamole автоматично отримують актуальну інформацію про доступні робочі місця для підключення користувачів.

Таким чином, архітектура бази даних забезпечує надійне, масштабоване та безпечне управління даними системи віддаленого доступу.

2.3. Опис типового сценарію роботи системи

Після успішного розгортання інфраструктури за допомогою Terraform, система готова до взаємодії з користувачами через захищений веб-інтерфейс.

Сценарій роботи користувача з системою виглядає наступним чином користувач відкриває браузер та переходить за доменним ім’ям веб-інтерфейсу, яке обслуговується службою Amazon Route 53 і перенаправляється через Amazon CloudFront до статичного веб-застосунку, збереженого у Amazon S3. Веб-інтерфейс, реалізований як односторінковий застосунок, завантажується у браузер користувача та пропонує ввести облікові дані для автентифікації.

Після введення логіну та паролю, веб-інтерфейс надсилає HTTPS-запит до ALB, який розподіляє трафік між запущеними екземплярами контейнерів Apache Guacamole. ALB забезпечує TLS-термінацію, отримуючи сертифікат з ACM та гарантуючи шифрування даних на етапі передачі.

Контейнери Apache Guacamole розгортаються у вигляді незалежних екземплярів у кластері ECS із використанням безсерверної обчислювальної моделі AWS Fargate. Завдяки контейнеризації кожен екземпляр є ізольованим, самодостатнім середовищем, що містить повний набір необхідних компонентів для обробки запитів користувачів. Контейнери працюють у незалежному від стану режимі, оскільки всі сесійні та конфігураційні дані зберігаються централізовано у базі даних Amazon RDS. Це дозволяє швидко розгортати нові екземпляри у випадку зростання навантаження або відмови існуючих.

Система масштабується горизонтально за допомогою механізмів автоматичного масштабування ECS Service: у разі збільшення кількості одночасних запитів або активних користувачів AWS Fargate автоматично запускає додаткові екземпляри контейнерів Guacamole. При зниженні навантаження надлишкові екземпляри контейнерів автоматично зупиняються, що дозволяє оптимізувати використання ресурсів і витрати.

Після отримання запиту контейнер Guacamole здійснює автентифікацію користувача, використовуючи централізовану базу даних Amazon RDS, де зберігаються облікові записи користувачів та їх права доступу. Після успішної автентифікації Guacamole звертається до тієї ж бази даних для отримання переліку доступних користувачу підключень. Веб-інтерфейс відображає цей список.

Користувач обирає бажане робоче місце зі списку, після чого ініціюється сесія віддаленого доступу через Guacamole. Встановлюється з’єднання із зазначеним віддаленим робочим місцем за допомогою відповідного протоколу доступу, через параметри підключення, які також отримуються із бази даних.

Таким чином, система забезпечує безпечний, масштабований та централізований доступ користувачів до віддалених або віртуальних робочих місць через єдиний веб-інтерфейс, гарантуючи при цьому надійність, безпеку зберігання даних і простоту адміністрування завдяки використанню хмарних сервісів AWS, контейнеризації та принципів інфраструктури як коду.

2.4. Висновки до Розділу 2

У розділі було розроблено та обґрунтовано вимоги до рішення, розроблено концепцію рішення та детальну архітектуру. Було обрано Apache Guacamole як базовий компонент рішення. Завдяки підтримці HTML5, RDP, VNC та SSH, він забезпечує повноцінний веб-орієнтований доступ до віддалених робочих місць без встановлення клієнтського ПЗ. Інфраструктура рішення реалізована за принципами Infrastructure as Code із використанням Terraform [20]. Це дозволяє автоматизувати розгортання, конфігурацію та підтримку всієї системи в середовищі Amazon Web Services. Масштабованість і балансування навантаження забезпечено за допомогою сервісів AWS ECS на базі Fargate і Application Load Balancer, що дозволяє гнучко реагувати на зміну навантаження та підвищує доступність сервісу. Безпека системи реалізується через TLS-термінацію, шифрування даних у RDS, розгортання бази даних у приватній підмережі та контроль доступу через групи безпеки. Додатково система авторизації Guacamole дозволяє налаштовувати доступ до робочих станцій на рівні користувачів. Веб-інтерфейс реалізовано як односторінковий застосунок (SPA) на основі Vue.js. Його статичні файли зберігаються в Amazon S3, а доставка до користувачів здійснюється через CloudFront. Взаємодія з серверною частиною відбувається безпосередньо через API Guacamole.

Типовий сценарій роботи демонструє зручність, безпеку та автоматизацію взаємодії користувача з системою: від автентифікації до підключення до віддаленого робочого місця.

Таким чином, запропонована архітектура забезпечує масштабоване, захищене та автоматизоване середовище для веб-орієнтованого доступу до віртуальних і віддалених робочих станцій.

РОЗДІЛ 3: ДЕТАЛЬНА РОЗРОБКА РІШЕННЯ

3.1. Організація коду проєкту

У даному розділі розглянуто практичну реалізацію інфраструктури хмарного застосування для доступу до віддалених та віртуальних робочих станцій. Рішення базується на HTML5 шлюзі для доступу до віддалених машин Apache Guacamole, який разом з іншими необхідними сервісами розгортається в Amazon AWS хмарному середовищі. Рішення побудоване за принципами інфраструктури як коду з використанням Terraform для опису всіх необхідних компонентів.

Файлову структуру проєкту зображено на рисунку 3.1.

```
1  ├── frontend/
2  ├── terraform/
3  │   ├── backend/
4  │   ├── bootstrap/
5  │   ├── database/
6  │   ├── frontend/
7  │   ├── network/
8  │   ├── resources/
9  │   ├── main.tf
10 │   └── ...
11 ├── example.config.yaml
12 ├── LICENSE
13 ├── README.md
```

Рисунок 3.1. Високорівнева структура проєкту

Код проєкту зберігається в віддаленому git репозиторії розміщеному на сервісі GitHub. Лістинг коду проєкту представлено в додатку А. Високорівнева структура проєкту включає в себе такі елементи:

1. Директорія frontend — Vue.js проєкт користувацького веб-інтерфейсу для побудови статичних файлів для SPA, які завантажуються на S3 для розгортання за допомогою AWS Cloudfront
2. Директорія terraform — розміщення файлів опису інфраструктури які використовуються Terraform для розгортання та написані мовою HCL. Опис інфраструктури розділено на модулі:
    - backend — модуль, відповідальний за розгортання серверної частини застосунку.
    - bootstrap — модуль для ініціалізації інфраструктури, який створює необхідні базові ресурси для подальшої роботи Terraform.
    - database — модуль, відповідальний за створення та ініціалізацію бази даних.
    - frontend (у складі terraform) — модуль для налаштування статичного хостингу веб-інтерфейсу в Amazon S3 та його публічного доступу через Amazon CloudFront.
    - network — модуль, що створює всі необхідні мережеві компоненти для розгортання інфраструктури.
    - resources — допоміжна директорія, що містить різні файли необхідні для інфраструктури.
    - Також в цій директорії розміщено файли необхідні для ініціації розгортання інфраструктури. Ці файли, а також модулі буде детальніше розглянуто далі в роботі.
3. example.config.yaml — файл, який містить приклад опису користувачів та конфігурацію доступу до віддалених робочих місць. Адміністратор системи має створити власний config.yaml файл який буде використано

рішенням для розгортання та налаштування доступу до віддалених робочих місць.

4. LICENSE — файл, що містить текст ліцензії для відкритого програмного забезпечення MIT, яка регулює умови використання проєкту.
5. README.md — файл, у якому описано інструкцію з використання рішення.

Інфраструктура проєкту реалізована за допомогою модульного підходу Terraform, де кожен компонент (мережа, база даних, серверна частина, веб-інтерфейс тощо) винесено в окремий модуль для забезпечення зручності супроводу та розподілу обов'язків.

У даній роботі оглянуто детальну розробку інфраструктури рішення, тому опис веб-інтерфейсу не включено. Далі в роботі буде детально розглянуто всі модулі та файли необхідні для розгортання інфраструктури які знаходяться в директорії terraform проєкту.

3.2. Верхній рівень terraform проєкту

На верхньому рівні директорії, крім модулів, також розміщено файли, які необхідні для ініціалізації та розгортання terraform проєкту:

- .terraform.lock.hcl — автоматично згенерований Terraform файл який використовується для фіксації версій провайдерів які використовуються проєктом.
- backend.tf — файл, у якому описано місце для зберігання станів інфраструктури розгорнутої Terraform.
- providers.tf — файл, у якому описано конфігурацію Terraform провайдерів використаних у проєктах.
- variables.tf — файл опису змінних які використовуються проєктом.
- outputs.tf — файл опису значень для виводу через Terraform.

- main.tf — файл який слугує точкою входу для Terraform. У ньому відбувається ініціалізація всіх модулів проєкту.

Файли variables.tf та outputs.tf наявні в кожному з зазначених вище модулів. У них описано значення які необхідно передавати між модулями. Їх опис в даній роботі не розглянуто. Усі передані значення буде розглянуто при описі файлу main.tf.

У файлі backend.tf налаштовано зберігання стану інфраструктури необхідне для функціонування Terraform в S3 бакеті. Також в цьому файлі вказано таблицю DynamoDB яка буде використовуватися для уникнення одночасних змін станів. Створення даних ресурсів буде оглянуто в описі модуля bootstrap. Файл налаштовує ім’я бакета, ім’я файлу стану, встановлює шифрування, регіон та назву таблиці.

Файл providers.tf описує конфігурацію провайдерів, що використовуються у проєкті. Для проєкту необхідно два по-різному налаштованих провайдери AWS. Основний, який встановлює регіон який передається з файлу змінних і використовується для майже всієї інфраструктури, та додатковий AWS провайдер, який налаштовано з використанням регіону us-east-1, що використовується для створення ACM сертифікату. Додатковий провайдер потрібен, оскільки TLS сертифікати для CloudFront мають бути створені виключно у регіоні us-east-1. Також для провайдерів налаштовані теги за замовчуванням які прикріпляються до всіх ресурсів створених у проєкті [21].

У main.tf відбувається виклик та налаштування залежностей між модулями інфраструктури. Першим викликається модуль frontend. Він є незалежним від інших модулів та на вхід отримує ім’я S3 бакета для розміщення файлів веб-інтерфейсу, доменне ім’я сервісу та ідентифікатор Route53 Hosted Zone, який має бути створений та налаштований для домену заздалегідь. Це обмеження виникає через те, що DNS сервери з Hosted Zone для домену мають бути зазначені в реєстратора ім’я та доступні на момент створення інфраструктури. Наступним викликається модуль network, який не має залежностей та конфігурація якого може

бути налаштована безпосередньо у файлі variables.tf відповідного модуля. Після налаштування мережі викликається модуль database. Database залежить від створення модулів network та frontend. На вхід отримується ідентифікатор приватного віртуального хмарного середовища, список ідентифікаторів приватних підмереж та їх CIDR-блоків з модулю network. Останній модуль backend отримує доменне ім’я, ідентифікатор Route53 Hosted Zone та регіон на вхід. Також отримує ідентифікатор приватного віртуального хмарного середовища, список ідентифікаторів приватних підмереж та їх CIDR-блоків з модулю network і ідентифікатор AWS Secret Manager секрету який зберігає дані автентифікації для доступу до бази даних з модуля database.

Таким чином відбувається ініціалізація всього проєкту з налаштованими залежностями, що дає змогу в правильному порядку створити всі необхідні хмарні ресурси.

3.3. Модуль bootstrap

Модуль bootstrap являє собою окремий Terraform проєкт, ціль якого — створити необхідні ресурси для управління основним проєктом такі, як таблицю DynamoDB для замикання можливості розгортання інфраструктури при можливому паралельному розгортанні задля уникнення колізій, а також S3 бакета для зберігання файлів стану Terraform. Також через те, що Terraform не має змоги створювати та управляти образами Docker, у цьому модулі створюється репозиторій Elastic Container Registry, в який треба буде завантажити образ Guacamole [22].

Модуль складається з чотирьох файлів:

- providers.tf — конфігурація провайдера. Це необхідно через те, що модуль являється окремим Terraform проєктом.
- ecr.tf — опис репозиторію Elastic Container Registry.
- s3.tf — опис S3 бакета для зберігання стану інфраструктури.

- dynamo_db.tf — опис DynamoDB таблиці.

Файл providers.tf описує використання провайдера AWS в регіоні, заданому через variables.tf. Важливо щоб цей регіон співпадав з регіоном основного проєкту. Також ресурс встановлює теги за замовчуванням для всіх ресурсів створених у проєкті.

У файлі ecr.tf описано ресурс Elastic Container Registry репозиторію. У налаштуваннях репозиторію вказано його ім’я, та налаштовано можливість примусового видалення для опції автоматичного видалення інфраструктури при наявних образах Docker.

Файл s3.tf налаштовує такі ресурси:

- aws_s3_bucket з опціями ім’я бакета та можливістю примусового видалення.
- aws_s3_bucket_versioning: увімкнення версіонування файлів створеного бакета.
- aws_s3_bucket_server_side_encryption_configuration: шифрування бакета.
- aws_s3_bucket_public_access_block: блокування публічного доступу до бакета.

Файл dynamo_db.tf описує створення ресурсу aws_dynamodb_table. Ресурс налаштовано такими параметрами: ім’я таблиці, атрибут таблиці, встановлення моделі оплати PAY_PER_REQUEST, та встановлення поля хешування.

Таким чином, модуль bootstrap описує всі необхідні ресурси які використовуються основним Terraform проєктом. Для того щоб виконати розгортання основної інфраструктури, адміністратору необхідно налаштувати, створити та завантажити образ Docker в репозиторій. Скрипт для автоматизації є частиною проєкту та розташований за шляхом terraform/resources/docker/push_image.sh.

3.4. Модуль network

Модуль network відповідає за створення мережевої інфраструктури, необхідної для роботи хмарного рішення. Він забезпечує створення ізольованого середовища з підтримкою публічного та приватного доступу, що дозволяє коректно розгортати ресурси з урахуванням вимог безпеки та доступності. CIDR-блоки задаються у файлі variables.tf, що дозволяє адаптувати архітектуру до існуючої інфраструктури та уникнути конфліктів IP-адрес.

Модуль складається з трьох основних файлів:

- vpc.tf — створення базової мережі (Virtual Private Cloud):
    - Створюється ресурс aws_vpc з заданим CIDR-блоком та увімкненими можливостями DNS.
    - Додається aws_internet_gateway для забезпечення виходу в Інтернет для публічних підмереж.
    - Створюється Elastic IP (aws_eip) та NAT-шлюз (aws_nat_gateway) для доступу до інтернету з приватних підмереж.
- subnets.tf — опис підмереж:
    - За допомогою ресурсу aws_availability_zones вибираються доступні зони доступності для регіону.
    - Створюються дві публічні (aws_subnet.public_a, aws_subnet.public_b) і дві приватні (aws_subnet.private_a, aws_subnet.private_b) підмережі в різних зонах доступності.
    - Для публічних підмереж вмикається параметр map_public_ip_on_launch для автоматичного призначення публічних IP-адрес при запуску.
- routes.tf — налаштування маршрутних таблиць:
    - Створюються дві таблиці маршрутизації: публічна (aws_route_table.public) та приватна (aws_route_table.private).
    - У публічній таблиці додається маршрут на 0.0.0.0/0 через інтернет-шлюз.

- У приватній таблиці створюється маршрут на 0.0.0.0/0 через NAT-шлюз.
- Підмережі асоціюються з відповідними маршрутними таблицями за допомогою ресурсів aws_route_table_association.

Таким чином, модуль network формує мережеву основу для побудови інфраструктури, забезпечуючи розділення на публічні та приватні ресурси та підвищену стійкість до відмов завдяки розгортанню в декількох зонах доступності AWS.

3.5. Модуль frontend

Модуль frontend відповідає за розгортання користувацького інтерфейсу застосунку у вигляді статичного сайту. Він реалізує архітектуру безсерверного хостингу, при якій всі файли веб-інтерфейсу (HTML, CSS, JavaScript тощо) розміщуються в Amazon S3, а публічний доступ до них надається через глобальну мережу доставки контенту Amazon CloudFront. Крім цього, модуль забезпечує налаштування TLS-сертифікату для шифрування з'єднань, DNS-записів для прив'язки доменного імені та політик безпеки для забезпечення приватності та коректного функціонування Single Page Application.

Модуль складається з таких основних файлів:

- distribution.tf — файл модуля, в якому описано CloudFront-дистрибуцію:
  - aws_cloudfront_origin_access_control — створює механізм Origin Access Control, який дозволяє CloudFront безпечно отримувати файли з приватного S3-бакету без відкриття публічного доступу.
  - aws_cloudfront_distribution — описує саму CDN-конфігурацію. Вона має два джерела:
    - <domain>-frontend — посилається на S3-бакет, де зберігається веб-інтерфейс.

        - <domain>-backend — використовується для направлення запитів до серверної частини /guacamole/*, розміщеного на окремому домені backend.<domain>.
    - У ordered_cache_behavior налаштовано перенаправлення для /guacamole/* з використанням керованих Amazon політик:
        - cache_policy_id — політика CachingDisabled, що вимикає кешування.
        - origin_request_policy_id — політика AllViewer, яка дозволяє передавати всі заголовки, куки та рядки запитів.
    - У default_cache_behavior вказано основну поведінку для обслуговування SPA:
        - Кешуються лише GET/HEAD запити.
        - Заборонено перенаправлення куків і параметрів запитів.
    - Також налаштовано обробку помилки 403: якщо запит завершується помилкою, відображається index.html, що необхідно для коректної роботи Vue Router при оновленні сторінок.
- s3.tf — опис хостингу файлів веб-інтерфейсу:
    - aws_s3_bucket — створює S3-бакет для зберігання зібраних файлів веб-інтерфейсу.
    - aws_s3_bucket_public_access_block — блокує публічний доступ до бакету.
    - aws_s3_bucket_policy — надає CloudFront-дистрибуції доступ до S3-бакету через умову AWS:SourceArn, що відповідає принципу найменших привілеїв.
    - aws_s3_object — завантажує файли із зібраного Vue-проєкту (frontend/dist) у відповідний S3-бакет. Для кожного файлу визначається MIME-тип (content_type), що важливо для правильного відображення контенту у браузері.

  - null_resource.invalidate_cloudfront_cache — автоматично виконує інвалідацію кешу CloudFront при зміні файлів, що дозволяє оновлення веб-інтерфейсу без ручного втручання.
- domain_cert.tf — описує отримання та валідацію TLS-сертифікату:
  - aws_acm_certificate — запитує сертифікат для основного домену через AWS Certificate Manager у регіоні us-east-1, який обов’язковий для CloudFront.
  - aws_route53_record — створює DNS-записи для валідації сертифікату через Route53.
  - aws_acm_certificate_validation — підтверджує валідацію сертифікату.
  - aws_route53_record — створює A-запис для доменного імені, який вказує на CloudFront-дистрибуцію, забезпечуючи зв’язок між DNS-запитом та CDN.

Завдяки модулю frontend, веб-інтерфейс рішення розгортається як масштабований, безсерверний SPA, доступний з будь-якої точки світу через швидку та захищену CDN-мережу. Усі налаштування безпеки, кешування, TLS-шифрування та маршрутизації враховані для забезпечення коректної роботи інтерфейсу та взаємодії з серверною частиною. Ілюстрації веб-інтерфейсу зображено на рисунках 3.2 та 3.3.

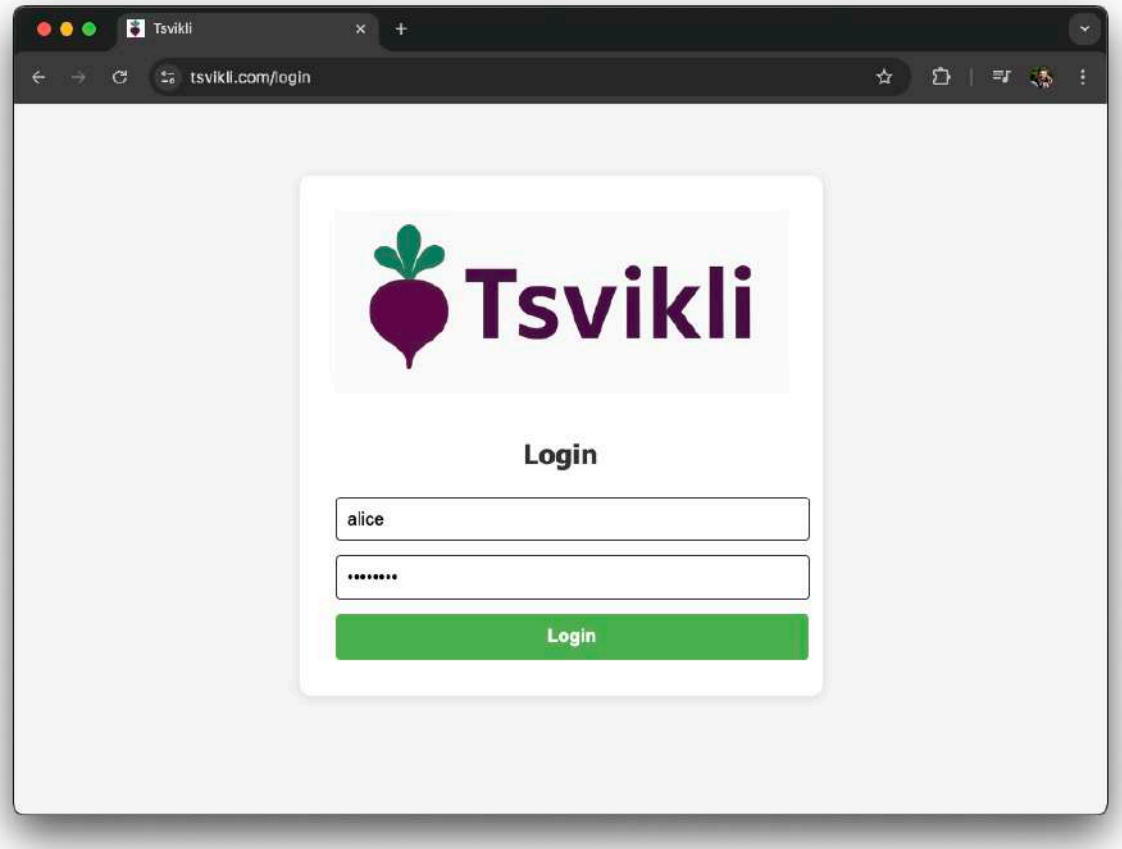

Рисунок 3.2. Сторінка автентифікації веб-інтерфейсу

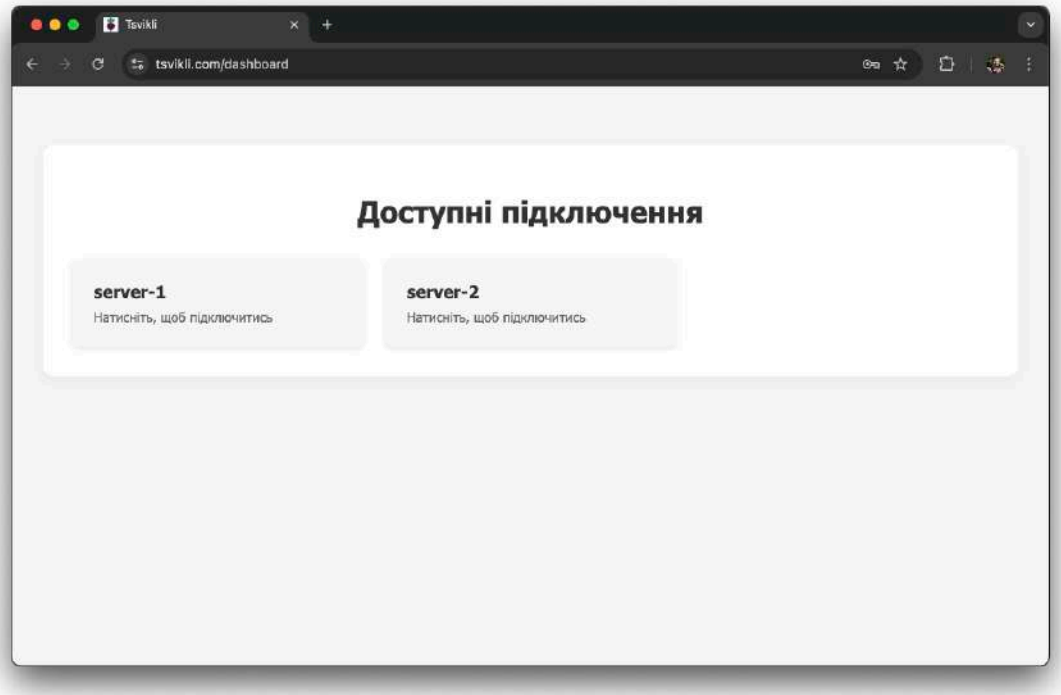

Рисунок 3.3. Сторінка вибору підключення веб-інтерфейсу

3.6. Модуль database

Модуль database відповідає за створення, налаштування та автоматичну ініціалізацію бази даних, яка зберігає конфігурації підключень, користувачів та інші службові параметри Apache Guacamole. Для підвищення безпеки всі облікові дані зберігаються у Secrets Manager, з’єднання шифрується за допомогою KMS, а виклик ініціалізації бази та оновлення конфігурації здійснюється через AWS Lambda.

Модуль складається з таких файлів:

- db.tf — опис бази даних:
    - aws_db_instance — створює екземпляр MySQL 8.0 з 20 ГБ пам'яті та шифруванням на основі KMS. Він ізольований у приватній підмережі та не має публічного доступу. Встановлено параметри резервного копіювання, дозволено автоматичне масштабування та видалення без фінального бекапу.
    - aws_kms_key — створює KMS-ключ для шифрування бази.
- secret.tf — зберігання облікових даних:
    - random_password — генерує випадковий пароль довжиною 20 символів.
    - aws_secretsmanager_secret — створює секрет у Secrets Manager з параметрами для підключення до бази: ім'я користувача, пароль, хост, порт, назва БД та тип бази даних.
    - aws_secretsmanager_secret_version — задає значення для секрету в форматі JSON.
- network.tf — налаштування мережі для доступу до бази:
    - aws_security_group — створює групу безпеки, яка дозволяє вхідний трафік на порт 3306 (MySQL) з приватних підмереж (CIDR-блоків) та дозволяє весь вихідний трафік.

    - aws_db_subnet_group — створює групу підмереж для бази даних, дозволяючи AWS розмістити екземпляр у відповідних приватних підмережах.
- config_file.tf — завантаження YAML-конфігурації користувачів до S3 та інтеграція з Lambda:
    - aws_s3_bucket — створює приватний S3 бакет для зберігання config.yaml.
    - aws_s3_bucket_server_side_encryption_configuration — увімкнено шифрування AES256.
    - aws_s3_bucket_public_access_block — забороняє будь-який публічний доступ до бакету.
    - aws_s3_object — завантажує локальний файл config.yaml у бакет та викликає подію ObjectCreated.
    - aws_lambda_permission — дозволяє виклик Lambda функції при події у бакеті.
    - aws_s3_bucket_notification — налаштовує повідомлення при зміні об’єкта config.yaml, що викликає оновлення через Lambda.
- db_init.tf — початкове створення схеми бази:
    - data.archive_file — архівує вихідні файли з папки resources/db_init.
    - aws_lambda_function — створює Lambda-функцію для ініціалізації бази (створення таблиць Guacamole).
    - aws_lambda_invocation — викликає функцію після її створення.
- db_update.tf — оновлення конфігурації бази за подією з S3:
    - data.archive_file — архівує вихідні файли з resources/db_update.
    - aws_lambda_function — створює Lambda, яка читає config.yaml з S3 та оновлює базу даних.
- iam.tf — роль для виконання Lambda-функцій:
    - aws_iam_role — дозволяє Lambda функціям виконуватись під цією IAM роллю.

- aws_iam_role_policy — дозволяє доступ до:
  - Secrets Manager для читання облікових даних.
  - S3 для читання конфігурацій.
  - EC2 (мережеві інтерфейси).
  - CloudWatch Logs для ведення журналів.

Таким чином, модуль database реалізує автоматизований цикл створення та оновлення бази даних з централізованим керуванням через YAML-конфігурації.

3.7. Модуль backend

Модуль backend відповідає за створення інфраструктури серверної частини рішення — Apache Guacamole, який надає веб-інтерфейс для веб-орієнтованого доступу до віддалених машин. Guacamole працює в середовищі Amazon ECS Fargate, забезпечуючи автоматичне масштабування, високу доступність та безпечне зберігання секретних параметрів у AWS Secrets Manager.

Модуль складається з таких компонентів:

- ecs.tf — налаштування кластеру ECS, сервісу, автоскейлінгу:
  - aws_ecs_cluster — ECS кластер, у якому запускаються контейнери.
  - aws_ecs_task_definition — описує розгортання двох контейнерів:
    - guacamole: головний застосунок, який отримує змінні середовища з Secrets Manager. Отримується з репозиторію ECR.
    - guacd: демон віддалених з'єднань, який працює окремо від веб-інтерфейсу. Отримується з публічного офіційного репозиторію.
  - В обох контейнерах налаштовано логування в CloudWatch.
  - aws_security_group — група безпеки, яка дозволяє вхідний трафік:
    - на порт 8080 з ALB,
    - на порт 4822 для внутрішньої взаємодії з guacd.

  - aws_ecs_service — ECS сервіс із бажаною кількістю розгортань (desired_count = 2) з балансуванням трафіку через ALB. Автоматичне оновлення, прив'язка до приватних підмереж.
  - aws_appautoscaling_target / aws_appautoscaling_policy — політика масштабування сервісу залежно від середнього навантаження CPU (target 70%). Це дозволяє автоматично запускати додаткові екземпляри при збільшенні навантаження.
- alb.tf — конфігурація Application Load Balancer:
  - aws_security_group — група безпеки для ALB, яка дозволяє доступ по HTTPS (порт 443) з будь-якої IP-адреси.
  - aws_lb — публічний Application Load Balancer для розподілу трафіку до контейнерів ECS.
  - aws_lb_target_group — цільова група, що працює на порту 8080 з протоколом HTTP, перевірка здоров'я виконується за шляхом /guacamole. Увімкнено stickiness за допомогою lb_cookie для стабільності сесій.
  - aws_lb_listener — HTTPS listener (порт 443) з TLS-термінацією. Переадресовує запити до цільової Guacamole.
- domain_cert.tf — TLS-сертифікація та DNS-запис:
  - aws_acm_certificate — запит TLS-сертифікату для backend.<domain> з перевіркою через DNS.
  - aws_route53_record / aws_acm_certificate_validation — перевірка сертифікату через Route53.
  - aws_route53_record — DNS A-запис для backend.<domain>, що вказує на ALB.
- iam.tf — ролі доступу:
  - aws_iam_role.ecs_task_execution_role — роль для запуску ECS розгортань з доступом до CloudWatch та ECR.

  - aws_iam_role.ecs_task_role — роль, яка дозволяє контейнерам читати секрети з Secrets Manager.
  - aws_iam_policy — IAM політика, яка дає дозвіл на secretsmanager:GetSecretValue.
  - aws_iam_role_policy_attachment — прикріплення ролей до відповідних політик.
- logs.tf — моніторинг:
  - aws_cloudwatch_log_group — група логів у CloudWatch з утриманням записів протягом 30 днів.
- data_sources.tf — додаткові джерела даних:
  - data.aws_caller_identity — використовується для визначення ID акаунту при побудові образу з ECR.
  - data.aws_secretsmanager_secret_version — отримання облікових даних для бази даних з AWS Secrets Manager.

Таким чином, модуль backend реалізує повну інфраструктуру для роботи Apache Guacamole як масштабованого сервісу в середовищі AWS, з безпечним TLS-доступом, балансуванням навантаження та автоматичним масштабуванням.

3.7 Висновки до розділу 3

У даному розділі було проведено аналіз реалізації інфраструктури хмарного застосування для веб-орієнтованого доступу до віддалених робочих станцій. Представлене рішення побудоване із застосуванням хмарних сервісів Amazon Web Services, що забезпечує високу масштабованість, доступність та безпеку системи. Ілюстрації підключення до віддалених робочих місць з використанням створеного рішення представлені на рисунках 3.4 та 3.5.

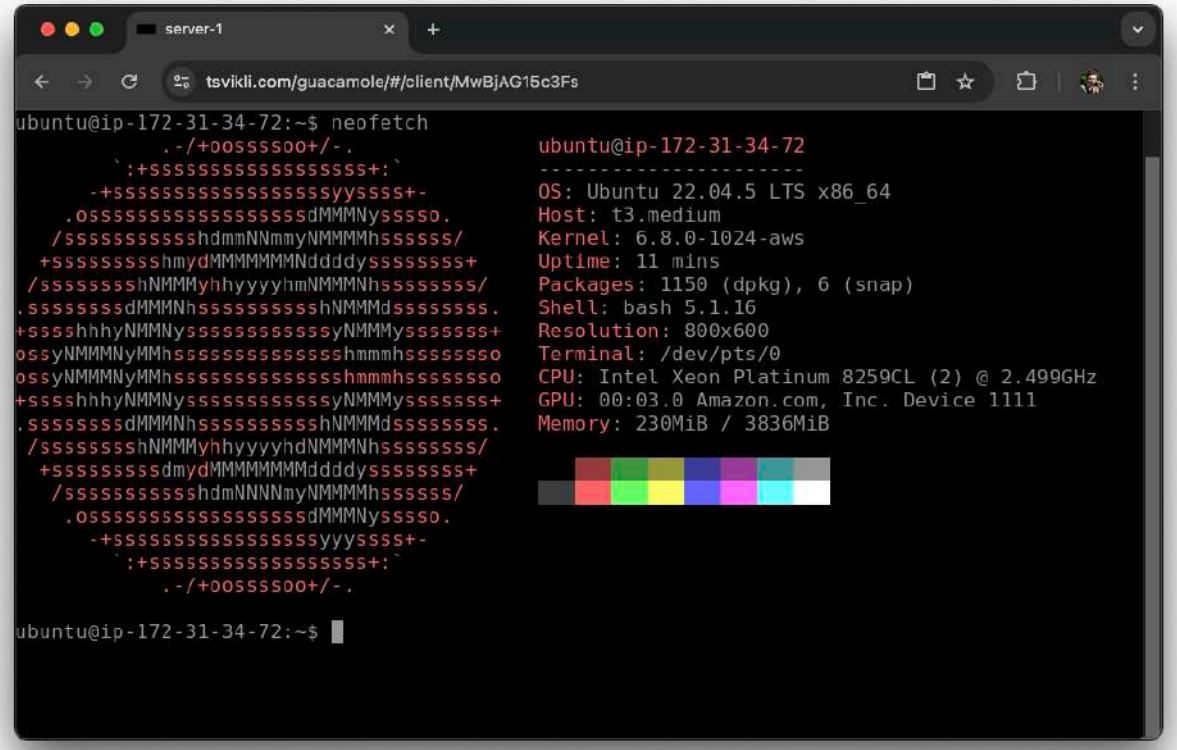

Рисунок 3.4. Демонстрація підключення через протокол SSH

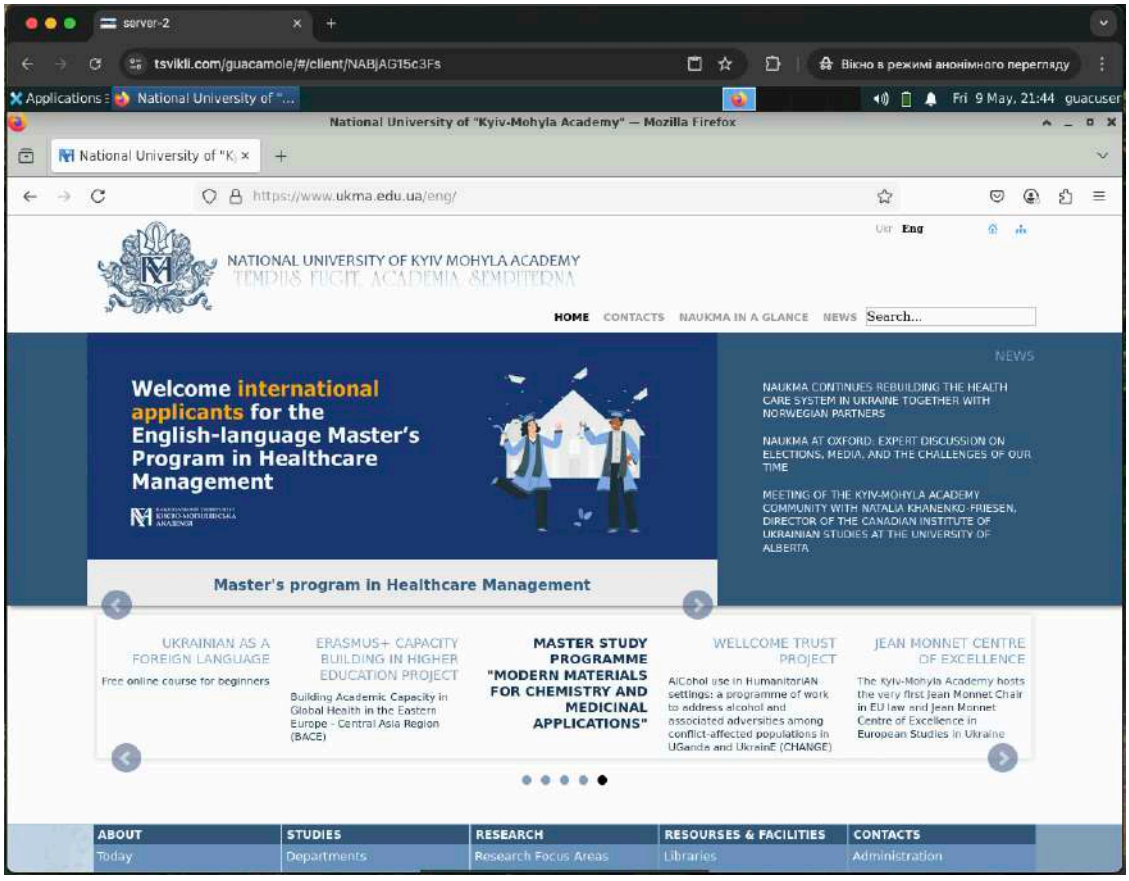

Рисунок 3.5. Демонстрація підключення через протокол RDP

Таким чином, спроєктована інфраструктура відповідає вимогам до сучасних SaaS-рішень, демонструючи приклад практичного використання хмарних технологій для створення надійного, безпечного та автоматизованого середовища для веб-орієнтованого доступу до віртуальних і віддалених робочих станцій.

ВИСНОВКИ ТА ПОДАЛЬШІ РЕКОМЕНДАЦІЇ

У роботі розглянуто задачу побудови хмарного веб-орієнтованого застосування для організації доступу до віртуальних і віддалених робочих станцій. Під час дослідження було вирішено низку теоретичних і практичних завдань. У результаті виконання поставлених завдань мету роботи було досягнуто, а саме — створено масштабоване, безпечне та автоматизоване рішення, яке легко інтегрується в існуючу корпоративну інфраструктуру та забезпечує централізований доступ до робочих середовищ через веб-браузер.

У ході виконання роботи було здійснено аналіз та порівняння наявних підходів та рішень, визначено їх переваги та недоліки, обґрунтовано необхідність розробки власного рішення, обрано базовий компонент та сформовано концепцію рішення. Проведено детальну розробку інфраструктури за принципами «інфраструктура як код» та реалізовано розроблене рішення у хмарному середовищі Amazon Web Services.

У процесі дослідження було детально проаналізовано чотири основних підходи до організації доступу до віртуальних та віддалених робочих станцій: Virtual Desktop Infrastructure, Desktop as a Service, програмні засоби віддаленого доступу та рішення які потребують ручного розгортання, зокрема Apache Guacamole. Проведений аналіз показав, що жоден з існуючих підходів не відповідає всім сучасним вимогам одночасно: веб-орієнтованість, простота інтеграції, відсутність залежності від зовнішніх провайдерів, гнучке масштабування, централізоване управління доступом та безпека. Ці обмеження стали основою для обґрунтування необхідності створення нового рішення, яке об’єднує переваги існуючих підходів і усуває їх недоліки. Під час проведення аналізу було визначено, що Apache Guacamole є найоптимальнішим базовим компонентом для побудови рішення для веб-орієнтованого доступу до віддалених та віртуальних робочих станцій. Apache Guacamole забезпечує гнучкий веб-орієнтований доступ до віддалених робочих станцій через HTML5. Водночас обмеження Apache

Guacamole, такі як відсутність вбудованих механізмів масштабування та балансування навантаження, складність розгортання та складність централізованого контролю доступу були враховані у запропонованому рішенні, що дозволило реалізувати необхідні додаткові механізми для створення повноцінного, автоматизованого хмарного рішення.

У практичній частині роботи було розроблено хмарне рішення на базі Apache Guacamole, яке відповідає всім перерахованим вище вимогам. Основними перевагами рішення, які були реалізовані під час розробки, є:

- Застосування Terraform, що дозволяє створити повністю автоматизований інструмент для швидкого та повторюваного розгортання необхідної інфраструктури на платформі AWS. Це скорочує час на розгортання, виключає ризик помилок при налаштуванні вручну та забезпечує можливість легко адаптувати інфраструктуру під потреби конкретної організації.
- Забезпечення динамічного масштабування рішення відповідно до навантаження, що реалізовано шляхом використання контейнеризації Apache Guacamole у Docker та сервісу AWS Elastic Container Service. Горизонтальне масштабування та балансування навантаження реалізовано з використанням AWS Application Load Balancer, що дозволяє ефективно розподіляти трафік між екземплярами додатку та автоматично відновлювати контейнери у разі їхньої відмови.
- Забезпечення безпеки передачі та зберігання даних. Весь трафік між користувачами та системою захищено за допомогою протоколу HTTPS із TLS сертифікатами, які видаються AWS Certificate Manager. Також забезпечено автоматичне шифрування збережених даних у базі даних Amazon RDS, що гарантує високий рівень конфіденційності інформації.
- Створення інтерфейсу веб-орієнтованого доступу без використання клієнтського ПЗ. Веб-інтерфейс реалізовано як односторінковий застосунок на базі Vue.js, що забезпечує інтерактивність, гнучкість і

зручність для користувачів. Статичні файли веб-інтерфейсу зберігаються у сховищі Amazon S3 та доставляються користувачам за допомогою Amazon CloudFront, що забезпечує додатковий рівень продуктивності, безпеки та масштабованості.

Таким чином, реалізоване рішення забезпечує створення автоматизованого, масштабованого та безпечного середовища веб-орієнтованого доступу до віртуальних та віддалених робочих станцій, що відповідає вимогам сучасних корпоративних інфраструктур.

На основі результатів виконаної роботи можна визначити кілька напрямів, які дозволять розширити функціональні можливості розробленого рішення. Серед таких рекомендацій є:

- Розгортання рішення в кількох географічних регіонах хмарного провайдера та маршрутизація трафіку, наприклад, через Route 53, дозволить реалізувати георепліkацію та високу доступність рішення з різних точок світу.
- Для спрощення та підвищення ефективності адміністрування можна розширити функціонал користувацького веб-інтерфейсу. Додаткові можливості можуть включати в себе панель адміністрування, моніторинг активності користувачів та управління правами доступу. Також поверх запропонованого рішення можна реалізувати будь-яке застосування та інтерфейс який можна монетизувати за необхідності.
- Для покращення зручності користувачів та адміністраторів рішення, для досягнення відповідності корпоративним вимогам можна інтегрувати рішення з корпоративними службами автентифікації, такими як Single Sign-On або OAuth 2.0. Це дозволить ефективно керувати доступом до ресурсів та покращить взаємодію користувачів із системою. Також це дозволить інтегрувати рішення в існуючу корпоративну інфраструктуру.
- Також, при збільшенні кількості функцій рішення, можна розглянути перехід на використання Kubernetes як оркестратора контейнерів у

рішенні замість ECS. Це дозволить гнучкіше налаштовувати інфраструктуру та впроваджувати додатковий функціонал для розвитку додатку. Також реалізація з використанням зовнішнього оркестратора контейнерів дозволить виконувати різні типи розгортання як, наприклад, гібридне розгортання, використовуючи хмарні та фізичні ресурси одночасно, або тільки фізичні ресурси.

Таким чином, реалізація запропонованих покращень дозволить підвищити ефективність розробленого рішення, забезпечить його гнучкість, надійність та здатність відповідати на виклики, пов’язані з подальшим розвитком корпоративної IT інфраструктури.

Виконання поставлених завдань забезпечило створення рішення, що відповідає сучасним вимогам корпорацій до організації віддаленої роботи з використанням хмарних технологій. Запропонована архітектура забезпечує простоту розгортання та адміністрування, високий рівень безпеки та масштабованість. Це дозволяє використовувати рішення як основу для створення веб-орієнтованого доступу до віддалених робочих станцій в умовах будь-якого підприємства, незалежно від його розмірів та сфери діяльності.

Подальший розвиток цього рішення може здійснюватися за напрямами, зазначеними вище, що дозволить розширити функціональні можливості. Реалізація запропонованих удосконалень створить умови для більш широкого застосування цього рішення в різних корпоративних середовищах та дозволить ще ефективніше вирішувати завдання організації віддаленого доступу до робочих ресурсів підприємств.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. QualityAssuranceGroup. Що таке VDI? (Virtual desktop infrastructure / Віртуальні машини). *QualityAssuranceGroup*. URL: https://qagroup.com.ua/publications/shcho-take-vdi/ (дата звернення: 22.05.2025).

2. De Novo. VMware Horizon vs стандартний VDI - відмінності та переваги. *De Novo*. URL: https://denovo.ua/blog/vmware-horizon-vs-vdi-osnovne (дата звернення: 22.05.2025).

3. Citrix. What is desktop as a service (daas)? - citrix. *Citrix.com*. URL: https://www.citrix.com/glossary/what-is-desktop-as-a-service-daas.htm (дата звернення: 22.05.2025).

4. Blechynden D., DeMuro J. P. Best remote desktop software of 2025. *TechRadar*. URL: https://www.techradar.com/news/best-remote-desktop-software (дата звернення: 22.05.2025).

5. ScreenConnect. Self-Hosted Remote Support Vs. Cloud-Based | ScreenConnect. *ScreenConnect*. URL: https://www.screenconnect.com/blog/self-hosted-remote-support (дата звернення: 22.05.2025).

6. Apache. Apache guacamole manual — apache guacamole manual v1.5.5. *Apache GuacamoleÂ®*. URL: https://guacamole.apache.org/doc/gug/ (дата звернення: 22.05.2025).

7. Amazon. What is infrastructure as code? - iac explained - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is/iac/?nc1=h_ls (дата звернення: 22.05.2025).

8. Amazon. What is AWS? - cloud computing with AWS - amazon web services. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is-aws/ (дата звернення: 22.05.2025).

9. Amazon. Container registry - amazon elastic container registry (amazon ECR) - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/ecr/ (дата звернення: 22.05.2025).

10. Amazon. Amazon ECS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/ecs/ (дата звернення: 22.05.2025).

11. Amazon. Amazon ECS vs Amazon EKS: making sense of AWS container services | Amazon Web Services. *Amazon Web Services*. URL: https://aws.amazon.com/blogs/containers/amazon-ecs-vs-amazon-eks-making-sense-of-aws-container-services/ (дата звернення: 22.05.2025).

12. Amazon. Serverless compute - AWS fargate - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/fargate/ (дата звернення: 22.05.2025).

13. Amazon. Application load balancer - ELB. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/elasticloadbalancing/application-load-balancer/ (дата звернення: 22.05.2025).

14. Lawson K. What is a single page application?. *Bloomreach*. URL: https://www.bloomreach.com/en/blog/what-is-a-single-page-application (дата звернення: 22.05.2025).

15. Amazon. Amazon S3 - cloud object storage - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/s3/?nc1=h_ls (дата звернення: 22.05.2025).

16. Amazon. CDN cloud service - amazon cloudfront - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/cloudfront/ (дата звернення: 22.05.2025).

17. Amazon. DNS service - amazon route 53 - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/route53/ (дата звернення: 22.05.2025).

18. Amazon. Certificate manager- AWS certificate manager - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/certificate-manager/ (дата звернення: 22.05.2025).

19. Amazon. Managed SQL database - amazon relational database service (RDS) - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/rds/ (дата звернення: 22.05.2025).

20. HashiCorp. Terraform | HashiCorp Developer. *Terraform | HashiCorp Developer*. URL: https://developer.hashicorp.com/terraform (дата звернення: 22.05.2025).

21. Amazon. Requirements for using SSL/TLS certificates with CloudFront - Amazon CloudFront. URL: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cnames-and-https-requirements.html (дата звернення: 22.05.2025).

22. HashiCorp. Backend Type: s3 | Terraform | HashiCorp Developer. *Backend Type: s3 | Terraform | HashiCorp Developer*. URL: https://developer.hashicorp.com/terraform/language/backend/s3 (дата звернення: 22.05.2025).

ДОДАТКИ

ДОДАТОК А. Лістинг коду проєкту

```
# terraform/main.tf

module "frontend" {
  source = "./frontend"

  bucket_name = var.bucket_name
  domain_name = var.domain_name
  dns_zone_id = var.dns_zone_id

  providers = {
    aws           = aws
    aws.us_east_1 = aws.us_east_1
  }
}

module "network" {
  source = "./network"
}

module "database" {
  source = "./database"

  vpc_id               = module.network.vpc_id
  private_subnets      = module.network.private_subnets
  private_subnet_cidrs = module.network.private_subnet_cidrs

  depends_on = [module.network, module.frontend]
}

module "backend" {
  source          = "./backend"
  vpc_id          = module.network.vpc_id
  private_subnets = module.network.private_subnets
  public_subnets  = module.network.public_subnets
  db_secret_arn   = module.database.db_secret_arn
  domain_name     = var.domain_name
  dns_zone_id     = var.dns_zone_id
  region          = var.region
}


# terraform/providers.tf

provider "aws" {
  region = var.region
```

```
  default_tags {
    tags = {
      Project     = "Tsvikli"
      ManagedBy   = "Terraform"
    }
  }
}

provider "aws" {
  alias  = "us_east_1"
  region = "us-east-1"

  default_tags {
    tags = {
      Project     = "Tsvikli"
      ManagedBy   = "Terraform"
    }
  }
}


# terraform/backend.tf

terraform {
  backend "s3" {
    bucket         = "tsvikli-terraform-state"
    key            = "terraform.tfstate"
    region         = "eu-west-1"
    dynamodb_table = "terraform-locks"
    encrypt        = true
  }
}


# terraform/variables.tf

variable "region" {
  description = "AWS region where resources for backend will be created"
  type        = string
  default     = "eu-west-1"
}

variable "bucket_name" {
  description = "Name of the S3 bucket for hosting frontend files"
  type        = string
  default     = "tsvikli-frontend"
}

variable "domain_name" {
  description = "Full domain name for frontend"
  type        = string
```

```
  default     = "tsvikli.com"
}

variable "dns_zone_id" {
  description = "DNS Zone ID created for domain"
  type        = string
}


# terraform/bootstrap/providers.tf

provider "aws" {
  region = var.region

  default_tags {
    tags = {
      Project     = "Tsvikli"
      ManagedBy   = "Manually"
    }
  }
}


# terraform/bootstrap/s3.tf

resource "aws_s3_bucket" "terraform_state" {
  bucket        = var.bucket_name
  force_destroy = true

  tags = {
    Name = "Terraform State Bucket"
  }
}

resource "aws_s3_bucket_versioning" "terraform_state_versioning" {
  bucket = aws_s3_bucket.terraform_state.id

  versioning_configuration {
    status = "Enabled"
  }
}

resource "aws_s3_bucket_server_side_encryption_configuration" "terraform_state_encryption"
{
  bucket = aws_s3_bucket.terraform_state.id

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}
```

```
# terraform/bootstrap/dynamo_db.tf

resource "aws_dynamodb_table" "terraform_locks" {
  name         = var.lock_table_name
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }

  tags = {
    Name = "Terraform Lock Table"
  }
}


# terraform/bootstrap/ecr.tf

resource "aws_ecr_repository" "guacamole" {
  name                 = "tsvikli/guacamole"
  image_tag_mutability = "MUTABLE"
  force_delete         = true

  encryption_configuration {
    encryption_type = "AES256"
  }

  tags = {
    Name = "Tsvikli Guacamole Repository"
  }
}


# terraform/network/vpc.tf

resource "aws_vpc" "main" {
  cidr_block           = var.vpc_cidr
  enable_dns_support   = true
  enable_dns_hostnames = true

  tags = {
    Name = "tsvikli-vpc"
  }
}

resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.main.id
```

```
  tags = {
    Name = "tsvikli-igw"
  }
}

resource "aws_eip" "nat" {
  domain = "vpc"
}

resource "aws_nat_gateway" "nat" {
  allocation_id = aws_eip.nat.id
  subnet_id     = aws_subnet.public_a.id

  tags = {
    Name = "tsvikli-nat"
  }
}

# terraform/network/subnets.tf

data "aws_availability_zones" "available" {}

resource "aws_subnet" "public_a" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = var.public_subnet_a_cidr
  availability_zone       = data.aws_availability_zones.available.names[0]
  map_public_ip_on_launch = true

  tags = {
    Name = "tsvikli-public-a"
  }
}

resource "aws_subnet" "public_b" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = var.public_subnet_b_cidr
  availability_zone       = data.aws_availability_zones.available.names[1]
  map_public_ip_on_launch = true

  tags = {
    Name = "tsvikli-public-b"
  }
}

resource "aws_subnet" "private_a" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.private_subnet_a_cidr
  availability_zone = data.aws_availability_zones.available.names[0]

  tags = {
    Name = "tsvikli-private-a"
  }
```

```
}

resource "aws_subnet" "private_b" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.private_subnet_b_cidr
  availability_zone = data.aws_availability_zones.available.names[1]

  tags = {
    Name = "tsvikli-private-b"
  }
}


# terraform/network/routes.tf

resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "tsvikli-public-rt"
  }
}

resource "aws_route" "public_internet_access" {
  route_table_id         = aws_route_table.public.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.gw.id
}

resource "aws_route_table_association" "public_a" {
  subnet_id      = aws_subnet.public_a.id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table_association" "public_b" {
  subnet_id      = aws_subnet.public_b.id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table" "private" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "tsvikli-private-rt"
  }
}

resource "aws_route" "private_nat_gateway" {
  route_table_id         = aws_route_table.private.id
  destination_cidr_block = "0.0.0.0/0"
  nat_gateway_id         = aws_nat_gateway.nat.id
}
```

```
resource "aws_route_table_association" "private_a" {
  subnet_id      = aws_subnet.private_a.id
  route_table_id = aws_route_table.private.id
}

resource "aws_route_table_association" "private_b" {
  subnet_id      = aws_subnet.private_b.id
  route_table_id = aws_route_table.private.id
}


# terraform/frontend/s3.tf

locals {
  dist_dir = "${path.module}/../../frontend/dist"
}

resource "aws_s3_bucket" "frontend" {
  bucket        = var.bucket_name
  force_destroy = true

  tags = {
    Project = "Tsvikli Frontend"
  }
}

resource "aws_s3_bucket_public_access_block" "frontend" {
  bucket = aws_s3_bucket.frontend.id

  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}

resource "aws_s3_bucket_policy" "frontend_policy" {
  bucket = aws_s3_bucket.frontend.id

  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Sid    = "AllowCloudFrontAccessViaOAC"
        Effect = "Allow"
        Principal = {
          Service = "cloudfront.amazonaws.com"
        }
        Action   = "s3:GetObject"
        Resource = "${aws_s3_bucket.frontend.arn}/*"
        Condition = {
          StringEquals = {
```

```
            "AWS:SourceArn" = aws_cloudfront_distribution.frontend.arn
          }
        }
      }
    ]
  })
}

resource "aws_s3_object" "frontend_files" {
  for_each = fileset(local.dist_dir, "**")

  bucket = aws_s3_bucket.frontend.id
  key    = each.key
  source = "${local.dist_dir}/${each.key}"
  etag   = filemd5("${local.dist_dir}/${each.key}")

  content_type = lookup(
    {
      html = "text/html"
      css  = "text/css"
      js   = "application/javascript"
      json = "application/json"
      png  = "image/png"
      jpg  = "image/jpeg"
      jpeg = "image/jpeg"
      svg  = "image/svg+xml"
      ico  = "image/x-icon"
      webp = "image/webp"
    },
    lower(element(split(".", each.key), length(split(".", each.key)) - 1)),
    "binary/octet-stream"
  )
}

resource "null_resource" "invalidate_cloudfront_cache" {
  triggers = {
    content_version = filemd5("${local.dist_dir}/index.html")
  }

  provisioner "local-exec" {
    command = "aws cloudfront create-invalidation --distribution-id
${aws_cloudfront_distribution.frontend.id} --paths '/*'"
  }
}


# terraform/frontend/domain_cert.tf

resource "aws_acm_certificate" "acm_cert" {
  provider                  = aws.us_east_1
  domain_name               = var.domain_name
  validation_method         = "DNS"
```

```
  tags = {
    Project = "Tsvikli Frontend"
  }

  lifecycle {
    create_before_destroy = true
  }
}

resource "aws_route53_record" "frontend_cert_validation" {
  for_each = {
    for dvo in aws_acm_certificate.acm_cert.domain_validation_options : dvo.domain_name =>
{
      name  = dvo.resource_record_name
      type  = dvo.resource_record_type
      value = dvo.resource_record_value
    }
  }

  zone_id = var.dns_zone_id
  name    = each.value.name
  type    = each.value.type
  ttl     = 60
  records = [each.value.value]
}

resource "aws_acm_certificate_validation" "frontend_cert_validation" {
  provider                = aws.us_east_1
  certificate_arn         = aws_acm_certificate.acm_cert.arn
  validation_record_fqdns = [for record in aws_route53_record.frontend_cert_validation :
record.fqdn]
}

resource "aws_route53_record" "frontend" {
  zone_id = var.dns_zone_id
  name    = var.domain_name
  type    = "A"

  alias {
    name                   = aws_cloudfront_distribution.frontend.domain_name
    zone_id                = aws_cloudfront_distribution.frontend.hosted_zone_id
    evaluate_target_health = false
  }
}


# terraform/frontend/distribution.tf

resource "aws_cloudfront_origin_access_control" "frontend_oac" {
  name                              = "frontend-oac"
  description                       = "OAC for CloudFront to S3"
```

```
  origin_access_control_origin_type = "s3"
  signing_behavior                  = "always"
  signing_protocol                  = "sigv4"
}

resource "aws_cloudfront_distribution" "frontend" {
  depends_on = [aws_acm_certificate_validation.frontend_cert_validation]

  enabled             = true
  default_root_object = "index.html"

  origin {
    domain_name              = aws_s3_bucket.frontend.bucket_regional_domain_name
    origin_id                = "tsvikli-frontend"
    origin_access_control_id = aws_cloudfront_origin_access_control.frontend_oac.id
  }

  origin {
    domain_name = "backend.tsvikli.com"
    origin_id   = "tsvikli-backend"

    custom_origin_config {
      http_port              = 80
      https_port             = 443
      origin_protocol_policy = "https-only"
      origin_ssl_protocols   = ["TLSv1.2"]
    }
  }

  ordered_cache_behavior {
    path_pattern     = "/guacamole/*"
    allowed_methods  = ["GET", "HEAD", "OPTIONS", "PUT", "POST", "PATCH", "DELETE"]
    cached_methods   = ["GET", "HEAD"]
    target_origin_id = "tsvikli-backend"

    viewer_protocol_policy = "redirect-to-https"

    cache_policy_id          = "4135ea2d-6df8-44a3-9df3-4b5a84be39ad"  # AWS Managed
CachingDisabled Policy
    origin_request_policy_id = "216adef6-5c7f-47e4-b989-5492eafa07d3"  # AWS Managed
AllViewer Policy
  }

  default_cache_behavior {
    allowed_methods  = ["GET", "HEAD"]
    cached_methods   = ["GET", "HEAD"]
    target_origin_id = "tsvikli-frontend"

    viewer_protocol_policy = "redirect-to-https"

    forwarded_values {
      query_string = false
```

```
      cookies {
        forward = "none"
      }
    }
  }

  viewer_certificate {
    acm_certificate_arn = aws_acm_certificate.acm_cert.arn
    ssl_support_method  = "sni-only"
  }

  restrictions {
    geo_restriction {
      restriction_type = "none"
    }
  }

  custom_error_response {
    error_caching_min_ttl = 0
    error_code            = 403
    response_code         = 200
    response_page_path    = "/index.html"
  }

  aliases = [var.domain_name]
}


# terraform/database/db.tf

resource "aws_kms_key" "db_key" {
  description = "KMS key for encrypting Tsvikli RDS database"
}

resource "aws_db_instance" "tsvikli_db" {
  identifier             = "tsvikli-db"
  allocated_storage      = 20
  max_allocated_storage  = 100
  storage_type           = "gp3"
  engine                 = "mysql"
  engine_version         = "8.0.35"
  instance_class         = "db.t3.micro"
  db_name                = var.db_name
  username               = var.db_username
  password               = random_password.db_password.result
  db_subnet_group_name   = aws_db_subnet_group.rds_subnet_group.name
  vpc_security_group_ids = [aws_security_group.rds_sg.id]
  storage_encrypted      = true
  kms_key_id             = aws_kms_key.db_key.arn
  multi_az               = false
  publicly_accessible    = false
```

```
  backup_retention_period   = 7
  deletion_protection       = false
  skip_final_snapshot       = true
  final_snapshot_identifier = "tsvikli-db-final-snapshot"

  tags = {
    Name = "tsvikli-db"
  }
}


# terraform/database/secret.tf

resource "random_password" "db_password" {
  length           = 20
  special          = true
  override_special = "!#%^*()_-+="
}

resource "aws_secretsmanager_secret" "db_credentials" {
  name                    = "tsvikli-db-credentials-secretk"
  recovery_window_in_days = 0

  depends_on = [aws_db_instance.tsvikli_db]
}

resource "aws_secretsmanager_secret_version" "db_credentials_version" {
  secret_id = aws_secretsmanager_secret.db_credentials.id
  secret_string = jsonencode({
    username = var.db_username
    password = random_password.db_password.result
    engine   = "mysql"
    host     = aws_db_instance.tsvikli_db.address
    port     = 3306
    dbname   = var.db_name
  })
}


# terraform/database/network.tf

resource "aws_security_group" "rds_sg" {
  name   = "tsvikli-rds-sg"
  vpc_id = var.vpc_id

  ingress {
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    cidr_blocks = var.private_subnet_cidrs
  }
```

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

  tags = {
    Name = "tsvikli-rds-sg"
  }
}

resource "aws_db_subnet_group" "rds_subnet_group" {
  name       = "tsvikli-db-subnet-group"
  subnet_ids = var.private_subnets

  tags = {
    Name = "tsvikli-db-subnet-group"
  }
}


# terraform/database/iam.tf

resource "aws_iam_role" "lambda_role" {
  name = "tsvikli-db-init-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [{
      Effect = "Allow",
      Principal = {
        Service = "lambda.amazonaws.com"
      },
      Action = "sts:AssumeRole"
    }]
  })
}

resource "aws_iam_role_policy" "lambda_policy" {
  name = "tsvikli-db-init-policy"
  role = aws_iam_role.lambda_role.id

  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect = "Allow",
        Action = [
          "secretsmanager:GetSecretValue",
          "secretsmanager:DescribeSecret"
        ],
```

```
        Resource = "*"
      },
      {
        Effect = "Allow",
        Action = [
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents"
        ],
        Resource = "*"
      },
      {
        Effect = "Allow",
        Action = [
          "ec2:CreateNetworkInterface",
          "ec2:DescribeNetworkInterfaces",
          "ec2:DeleteNetworkInterface"
        ],
        Resource = "*"
      },
      {
        Effect = "Allow",
        Action = [
          "s3:GetObject",
          "s3:PutObject",
          "s3:DeleteObject",
          "s3:ListBucket"
        ],
        Resource = [
          aws_s3_bucket.config_bucket.arn,
          "${aws_s3_bucket.config_bucket.arn}/*"
        ]
      }
    ]
  })
}


# terraform/database/config_file.tf

resource "random_id" "suffix" {
  byte_length = 4
}

resource "aws_s3_bucket" "config_bucket" {
  bucket        = "tsvikli-config-${random_id.suffix.hex}"
  force_destroy = true

  tags = {
    Project = "Tsvikli Config"
  }
}
```

```
resource "aws_s3_bucket_server_side_encryption_configuration" "config_bucket_encryption" {
  bucket = aws_s3_bucket.config_bucket.id

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

resource "aws_s3_bucket_public_access_block" "config_bucket_public_access" {
  bucket = aws_s3_bucket.config_bucket.id

  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}

resource "aws_s3_object" "config_file" {
  bucket = aws_s3_bucket.config_bucket.id
  key    = "config.yaml"
  source = "${path.module}/../../config.yaml"
  etag   = filemd5("${path.module}/../../config.yaml")

  depends_on = [
    aws_s3_bucket.config_bucket,
    aws_s3_bucket_notification.config_bucket_notification
  ]
}

resource "aws_lambda_permission" "allow_s3_invoke" {
  statement_id  = "AllowS3Invoke"
  action        = "lambda:InvokeFunction"
  function_name = aws_lambda_function.config_updater.function_name
  principal     = "s3.amazonaws.com"
  source_arn    = aws_s3_bucket.config_bucket.arn
}

resource "aws_s3_bucket_notification" "config_bucket_notification" {
  bucket = aws_s3_bucket.config_bucket.id

  lambda_function {
    lambda_function_arn = aws_lambda_function.config_updater.arn
    events              = ["s3:ObjectCreated:*", "s3:ObjectRemoved:*"]
    filter_prefix       = "config.yaml"
  }

  depends_on = [
    aws_lambda_permission.allow_s3_invoke
  ]
```

```
}


# terraform/database/db_init.tf

data "archive_file" "init_lambda_zip" {
  type        = "zip"
  source_dir  = "${path.module}/../resources/db_init/"
  output_path = "${path.module}/lambda_package.zip"
  excludes    = []
}

resource "aws_lambda_function" "db_initializer" {
  function_name = "tsvikli-db-init"
  handler       = "lambda.lambda_handler"
  runtime       = "python3.11"

  filename                       = data.archive_file.init_lambda_zip.output_path
  source_code_hash               =
data.archive_file.init_lambda_zip.output_base64sha256
  replace_security_groups_on_destroy = true

  role = aws_iam_role.lambda_role.arn

  timeout = 120

  vpc_config {
    subnet_ids         = var.private_subnets
    security_group_ids = [aws_security_group.rds_sg.id]
  }

  environment {
    variables = {
      DB_SECRET_NAME = aws_secretsmanager_secret.db_credentials.name
    }
  }

  depends_on = [aws_secretsmanager_secret_version.db_credentials_version,
aws_db_instance.tsvikli_db]
}

resource "aws_lambda_invocation" "db_init" {
  function_name = aws_lambda_function.db_initializer.function_name
  input         = "{}"

  depends_on = [
    aws_db_instance.tsvikli_db,
    aws_lambda_function.db_initializer
  ]
}
```

```
# terraform/database/db_update.tf

data "archive_file" "db_update_lambda_zip" {
  type        = "zip"
  source_dir  = "${path.module}/../resources/db_update/"
  output_path = "${path.module}/db_update_lambda_package.zip"
}

resource "aws_lambda_function" "config_updater" {
  function_name                  = "tsvikli-config-updater"
  handler                        = "lambda.lambda_handler"
  runtime                        = "python3.11"
  filename                       = data.archive_file.db_update_lambda_zip.output_path
  source_code_hash               =
data.archive_file.db_update_lambda_zip.output_base64sha256
  role                           = aws_iam_role.lambda_role.arn
  timeout                        = 120
  replace_security_groups_on_destroy = true

  vpc_config {
    subnet_ids         = var.private_subnets
    security_group_ids = [aws_security_group.rds_sg.id]
  }

  environment {
    variables = {
      DB_SECRET_NAME = aws_secretsmanager_secret.db_credentials.name
      CONFIG_BUCKET  = aws_s3_bucket.config_bucket.bucket
      CONFIG_KEY     = "config.yaml"
    }
  }
  depends_on = [
    aws_secretsmanager_secret_version.db_credentials_version,
    aws_db_instance.tsvikli_db,
  ]
}


# terraform/backend/data_sources.tf

data "aws_caller_identity" "current" {}

data "aws_secretsmanager_secret_version" "db_creds" {
  secret_id = var.db_secret_arn
}


# terraform/backend/logs.tf

resource "aws_cloudwatch_log_group" "guacamole_logs" {
  name              = "/ecs/tsvikli/guacamole"
  retention_in_days = 30
```

```
}


# terraform/backend/iam.tf

resource "aws_iam_role" "ecs_task_execution_role" {
  name = "tsvikli-ecs-task-execution-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [{
      Effect = "Allow",
      Principal = {
        Service = "ecs-tasks.amazonaws.com"
      },
      Action = "sts:AssumeRole"
    }]
  })
}

resource "aws_iam_role" "ecs_task_role" {
  name = "tsvikli-ecs-task-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [{
      Effect = "Allow",
      Principal = {
        Service = "ecs-tasks.amazonaws.com"
      },
      Action = "sts:AssumeRole"
    }]
  })
}

resource "aws_iam_policy" "secret_manager_access" {
  name = "SecretManagerAccessPolicy"
  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [{
      Effect   = "Allow",
      Action   = ["secretsmanager:GetSecretValue"],
      Resource = var.db_secret_arn
    }]
  })
}

resource "aws_iam_role_policy_attachment" "ecs_task_execution_policy" {
  role       = aws_iam_role.ecs_task_execution_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
}

resource "aws_iam_role_policy_attachment" "attach_secret_policy" {
  role       = aws_iam_role.ecs_task_role.name
```

```
  policy_arn = aws_iam_policy.secret_manager_access.arn
}


# terraform/backend/ecs.tf

resource "aws_ecs_cluster" "tsvikli_cluster" {
  name = "tsvikli-cluster"
}

resource "aws_ecs_task_definition" "guacamole_task" {
  family                   = "tsvikli-guacamole"
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = "512"
  memory                   = "1024"
  execution_role_arn       = aws_iam_role.ecs_task_execution_role.arn
  task_role_arn            = aws_iam_role.ecs_task_role.arn

  container_definitions = jsonencode([
    {
      name      = "guacamole"
      image     =
"${data.aws_caller_identity.current.account_id}.dkr.ecr.${var.region}.amazonaws.com/tsvikli
/guacamole:latest"
      essential = true
      portMappings = [{
        containerPort = 8080
        hostPort      = 8080
      }]
      environment = [
        { name = "GUACD_HOSTNAME", value = "localhost" },
        { name = "MYSQL_HOSTNAME", value =
jsondecode(data.aws_secretsmanager_secret_version.db_creds.secret_string)["host"] },
        { name = "MYSQL_DATABASE", value =
jsondecode(data.aws_secretsmanager_secret_version.db_creds.secret_string)["dbname"] },
        { name = "MYSQL_USER", value = "guacamole_user" },
        { name = "MYSQL_PASSWORD", value =
jsondecode(data.aws_secretsmanager_secret_version.db_creds.secret_string)["password"] }
      ]
      logConfiguration = {
        logDriver = "awslogs"
        options = {
          "awslogs-group"         = aws_cloudwatch_log_group.guacamole_logs.name
          "awslogs-region"        = var.region
          "awslogs-stream-prefix" = "guacamole"
        }
      }
    },
    {
      name      = "guacd"
```

```
      image     = "guacamole/guacd:1.5.5"  # Configuration is not necessary, thus original
image is used
      essential = true
      portMappings = [{
        containerPort = 4822
        hostPort      = 4822
      }]
      logConfiguration = {
        logDriver = "awslogs"
        options = {
          "awslogs-group"         = aws_cloudwatch_log_group.guacamole_logs.name
          "awslogs-region"        = var.region
          "awslogs-stream-prefix" = "guacd"
        }
      }
    }
  ])
}

resource "aws_security_group" "ecs_sg" {
  name        = "tsvikli-ecs-sg"
  description = "Security group for ECS tasks"
  vpc_id      = var.vpc_id

  ingress {
    from_port       = 8080
    to_port         = 8080
    protocol        = "tcp"
    security_groups = [aws_security_group.alb_sg.id]
  }

  ingress {
    from_port = 4822
    to_port   = 4822
    protocol  = "tcp"
    self      = true
  }

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

  tags = {
    Name = "tsvikli-ecs-sg"
  }
}

resource "aws_ecs_service" "guacamole_service" {
  name            = "tsvikli-guacamole-service"
```

```
  cluster         = aws_ecs_cluster.tsvikli_cluster.id
  task_definition = aws_ecs_task_definition.guacamole_task.arn
  desired_count   = 2
  launch_type     = "FARGATE"

  network_configuration {
    security_groups  = [aws_security_group.ecs_sg.id]
    subnets          = var.private_subnets
    assign_public_ip = false
  }

  load_balancer {
    target_group_arn = aws_lb_target_group.guacamole_tg.arn
    container_name   = "guacamole"
    container_port   = 8080
  }

  depends_on = [aws_lb_listener.guacamole_listener]
}

resource "aws_appautoscaling_target" "guacamole_scaling_target" {
  max_capacity       = 6
  min_capacity       = 2
  resource_id        =
"service/${aws_ecs_cluster.tsvikli_cluster.name}/${aws_ecs_service.guacamole_service.name}"
  scalable_dimension = "ecs:service:DesiredCount"
  service_namespace  = "ecs"
}

resource "aws_appautoscaling_policy" "guacamole_cpu_scaling_policy" {
  name               = "tsvikli-guacamole-cpu-scaling-policy"
  policy_type        = "TargetTrackingScaling"
  resource_id        = aws_appautoscaling_target.guacamole_scaling_target.resource_id
  scalable_dimension =
aws_appautoscaling_target.guacamole_scaling_target.scalable_dimension
  service_namespace  = aws_appautoscaling_target.guacamole_scaling_target.service_namespace

  target_tracking_scaling_policy_configuration {
    predefined_metric_specification {
      predefined_metric_type = "ECSServiceAverageCPUUtilization"
    }
    target_value       = 70.0
    scale_in_cooldown  = 300
    scale_out_cooldown = 20
  }
}


# terraform/backend/alb.tf

resource "aws_security_group" "alb_sg" {
  name        = "tsvikli-alb-sg"
```

```
  description = "Security group for ALB"
  vpc_id      = var.vpc_id

  ingress {
    from_port   = 443
    to_port     = 443
    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"]
  }

  tags = {
    Name = "tsvikli-alb-sg"
  }
}

resource "aws_lb" "guacamole_alb" {
  name               = "tsvikli-guacamole-alb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb_sg.id]
  subnets            = var.public_subnets

  enable_deletion_protection = false
}

resource "aws_lb_target_group" "guacamole_tg" {
  name        = "tsvikli-guacamole-tg"
  port        = 8080
  protocol    = "HTTP"
  vpc_id      = var.vpc_id
  target_type = "ip"

  health_check {
    path                = "/guacamole"
    interval            = 30
    timeout             = 5
    unhealthy_threshold = 3
    healthy_threshold   = 2
    matcher             = "200-399"
  }

  stickiness {
    enabled         = true
    type            = "lb_cookie"
    cookie_duration = 3600
```

```
  }
}

resource "aws_lb_listener" "guacamole_listener" {
  load_balancer_arn = aws_lb.guacamole_alb.arn
  port              = 443
  protocol          = "HTTPS"
  ssl_policy        = "ELBSecurityPolicy-2016-08"
  certificate_arn   = aws_acm_certificate.guacamole_backend_cert.arn

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.guacamole_tg.arn
  }
}


# terraform/backend/domain_cert.tf

resource "aws_acm_certificate" "guacamole_backend_cert" {
  domain_name       = "backend.${var.domain_name}"
  validation_method = "DNS"

  lifecycle {
    create_before_destroy = true
  }

  tags = {
    Project = "Tsvikli Backend"
  }
}

resource "aws_route53_record" "guacamole_backend_cert_validation" {
  for_each = {
    for dvo in aws_acm_certificate.guacamole_backend_cert.domain_validation_options :
dvo.domain_name => {
      name  = dvo.resource_record_name
      type  = dvo.resource_record_type
      value = dvo.resource_record_value
    }
  }

  zone_id = var.dns_zone_id
  name    = each.value.name
  type    = each.value.type
  ttl     = 60
  records = [each.value.value]
}

resource "aws_acm_certificate_validation" "guacamole_backend_cert_validation" {
  certificate_arn         = aws_acm_certificate.guacamole_backend_cert.arn
```

```
  validation_record_fqdns = [for record in
aws_route53_record.guacamole_backend_cert_validation : record.fqdn]
}

resource "aws_route53_record" "backend_record" {
  zone_id = var.dns_zone_id
  name    = "backend.${var.domain_name}"
  type    = "A"

  alias {
    name                   = aws_lb.guacamole_alb.dns_name
    zone_id                = aws_lb.guacamole_alb.zone_id
    evaluate_target_health = true
  }
}
```