# Final Practical Report: AWS Cloud Infrastructure for TechSavvy Solutions

# Introduction

TechSavvy Solutions is an e-commerce company experiencing rapid growth. To support increased user demand, the company needed to migrate to a scalable and highly available cloud architecture on AWS. This project focused on building an automated environment using AWS CloudFormation, ensuring that future deployments can be reproduced quickly and consistently.

The environment was designed as a three-tier architecture with a web tier, an application tier, and a database tier. High availability, scalability, and security were treated as the top priorities throughout the design process.

# Architecture Overview

The solution was built in AWS using the following components:

* **VPC and Networking**
  + A custom Virtual Private Cloud (VPC) was created with multiple public and private subnets spread across two Availability Zones. Public subnets host the load balancer and NAT gateway, while private subnets host the application and database layers. This layout ensures separation of concerns and better security.
* **Load Balancer (ALB)**
  + An Application Load Balancer routes incoming HTTP traffic from the internet to the web tier. The ALB improves fault tolerance by checking the health of instances and distributing requests across multiple Availability Zones.
* **Web Tier**
  + The web tier consists of an Auto Scaling Group of EC2 instances running Nginx. These servers sit behind the ALB in public subnets and serve as the entry point for the application.
* **Application Tier**
  + The application tier is another Auto Scaling Group of EC2 instances running a simple Python backend. These servers live in private subnets and are only accessible from the web tier, not directly from the internet.
* **Database Tier** 
  + The backend database is provided by Amazon RDS. A Multi-AZ deployment was used for automatic failover, and backups are configured for data protection. The database resides in private subnets and only accepts traffic from the application tier.

# Architecture Overview

The solution was built in AWS using the following components:

* **VPC and Networking** 
  + A custom Virtual Private Cloud (VPC) was created with multiple public and private subnets spread across two Availability Zones. Public subnets host the load balancer and NAT gateway, while private subnets host the application and database layers. This layout ensures separation of concerns and better security.
* **Load Balancer (ALB)**
  + An Application Load Balancer routes incoming HTTP traffic from the internet to the web tier. The ALB improves fault tolerance by checking the health of instances and distributing requests across multiple Availability Zones.
* **Web Tier**
  + The web tier consists of an Auto Scaling Group of EC2 instances running Nginx. These servers sit behind the ALB in public subnets and serve as the entry point for the application.
* **Application Tier**
  + The application tier is another Auto Scaling Group of EC2 instances running a simple Python backend. These servers live in private subnets and are only accessible from the web tier, not directly from the internet.
* **Database Tier**
  + The backend database is provided by Amazon RDS. A Multi-AZ deployment was used for automatic failover, and backups are configured for data protection. The database resides in private subnets and only accepts traffic from the application tier.

# Reliability and High Availability

Reliability was built into the design in several ways:

* The use of two Availability Zones provides redundancy for both compute and database layers.
* RDS Multi-AZ ensures the database remains available even if one AZ goes down.
* Health checks on the ALB automatically remove unhealthy web instances from rotation.
* Automated backups in RDS protect against data loss.

This means the application can continue serving users even during failures in individual components.

# Automation

The entire environment was built using a single AWS CloudFormation YAML template. This ensures:

* Infrastructure can be recreated in a new region with minimal changes.
* Updates to the environment can be version-controlled and tracked in Git.
* Human error is reduced since all components are provisioned consistently.

The template itself is stored on GitHub for easy reference and reuse.

# Access to the Template

The full CloudFormation YAML template used to build this environment is available here:  
 →[GitHub Repository: TechSavvy Final Practical](https://github.com/tehShon/Final.git?utm_source=chatgpt.com)←

This allows the YAML file to be viewed in a clean format and cloned for testing or further development.