Demo Project Report

Demo Project AWS Deployment - Challenges & Resolutions Summary

This document summarizes all the main technical challenges, issues, and fixes encountered during the process of setting up, deploying, and automating the demo_project backend API infrastructure on AWS. It covers Terraform configurations, ECS setup, IAM permissions, networking, and CI/CD automation via GitHub Actions.

1. Initial Setup Challenges

The first phase involved setting up Terraform modules for VPC, EC2, and IAM. We encountered variable and output mismatches during initialization and resource linking. Specifically, the EC2 module did not expose 'vpc_id' or 'security_group_id' outputs, leading to AWS CLI command errors.

We fixed this by defining clear output variables in each module and aligning variable names between environments.

2. Docker Image and ECR Repository Issues

We faced multiple issues with Docker image builds and AWS ECR integration. The main challenges were:

- 403 Forbidden when pushing Docker images to ECR (missing repository or wrong permissions).
- ECR repository not found after destroy/redeploy cycles.
- Architecture mismatch (ARM vs AMD64) when building from macOS for EC2 Linux.
- Gunicorn not found in the container (missing package in requirements.txt).

Fixes implemented:

Updated the Dockerfile to explicitly install Gunicorn and expose the correct port.

Added '--platform linux/amd64' to ensure compatibility with EC2 instances.

Added logic in deploy.yml to create the ECR repository automatically if missing.

Improved tagging and push logic to guarantee the image always overwrites previous versions.

3. EC2 and Django Startup Problems

Initially, the EC2 instance launched but only the PostgreSQL container ran. Django was failing to start due to environment variable parsing errors and architecture mismatches.

Logs from '/var/log/user-data-debug.log' revealed that the Docker image could not be pulled (manifest errors).

To fix this, we rebuilt the Docker image for 'linux/amd64', adjusted the startup script to correctly expand environment variables (without escape characters), and verified the image via SSM session with 'docker ps' and 'docker logs'. Once fixed, both containers started successfully on every deploy.

4. Networking and Access Issues

After the containers were running, the Django app was not reachable externally. The root cause was that the EC2 security group did not expose port 8000, and the instance route table was not associated with an internet gateway.

Added security group rule for inbound TCP on port 8000 (CidrBlock 0.0.0.0/0).

Confirmed IGW route in Terraform and verified with 'aws ec2 describe-route-tables'.

Added 'ALLOWED_HOSTS = ["*"]' to Django settings to accept external requests.

5. Terraform Destroy Hangs (DependencyViolation)

Destroy operations occasionally hung due to AWS dependency violations (security group still attached to EC2).

Added 'depends_on = [aws_security_group.app_sg]' inside the EC2 resource block to enforce proper destroy order.

Used AWS CLI to identify and manually detach stuck ENIs (via 'describe-network-interfaces').

Validated destroy behavior by running multiple destroy/apply cycles without issues.

6. ECS Deployment Issues

Initial attempts to create the ECS service failed due to invalid subnet IDs and missing VPC resources.

This was caused by destroyed or mismatched infrastructure components from previous runs.

Fix: Created a new VPC, two subnets, and a security group from scratch, then re-linked them to the ECS cluster configuration. Added proper references to subnet and security group IDs in the ECS service creation command.

7. Load Balancer Configuration Errors

Encountered errors with invalid target group ARNs and missing internet gateways during ALB creation.

Fix: Recreated the ALB using subnet IDs tied to the new VPC, then attached an internet gateway and created the correct route tables. Verified the target group ARN format and successfully created a listener.

8. IAM Role & Permissions

ECS was unable to assume the correct IAM role during service creation. This was due to missing trust relationships and the absence of an ECS task execution role.

Fix: Created the 'ecsTaskExecutionRole' with correct policies including AmazonECSTaskExecutionRolePolicy and updated task definitions to reference it.

9. Terraform Module Refactoring

The initial Terraform configuration had hardcoded resources mixed with unstructured files. Modules were created for better isolation: vpc, ec2, iam, alb, and s3.

Fix: Split the infrastructure into reusable modules with proper input/output variable definitions. Adjusted outputs to reference module outputs instead of raw resource names.

10. EC2 & IAM Integration

Terraform plan failed due to missing variables such as ec2_role_name and mismatched outputs between IAM and EC2 modules.

Fix: Declared missing variables and added outputs for IAM roles to connect correctly with EC2 instance profiles.

11. GitHub Actions Automation

Created CI/CD workflows for deploy and destroy infrastructure via manual dispatch triggers.

Encountered workflow initialization and branch protection conflicts.

Fix: Updated YAML workflow definitions to include environment permissions and ensured Terraform plan/apply/destroy steps used the correct AWS credentials and branch context.

12. Performance & Context Issues

Long chat history caused severe slowdowns due to context overload in the development process.

Fix: Recommended clean chat reset and structured project continuation using the ‘infra-cleanup' branch with all prior work consolidated.

Conclusion

The demo project successfully evolved into a clean, modular, and automated infrastructure deployment system for a Django-based API hosted on AWS. The final setup is stable, low-cost, and ready for demonstration as part of DevOps/SRE technical showcases.