HEAT ISLAND DETECTION AND MITIGATION SYSTEM

Deployment Report

R25-002

	Name with Initials	**Registration Number**
1.	Ayeshmantha S K S	IT21219320
2.	Kumara B D A N	IT21256264
3.	Silva G M S S	IT21802126
4.	Madhuwantha G K O	IT21802058

Table of Contents

1. Introduction

This report documents the cloud deployment architecture and automation pipeline used to host and manage the four independent microservices developed under the HeatScape – AI-Driven Detection and Mitigation of Urban Heat Islands project.

The goal of deployment was to ensure scalability, GPU-optimized inference, and automated continuous delivery while maintaining modularity between services.

All components were deployed to Amazon Web Services (AWS) using Elastic Container Service (ECS) with EC2 launch type, Elastic Container Registry (ECR) for image management, and Application Load Balancer (ALB) for service routing.

2. System Overview

Component	Function	Hosting Type	Dependencies
Image Processing Service	Runs AI models (YOLOv8, SAM, CLIP, Depth Anything) for object segmentation, material classification, and surface area estimation.	GPU (g4dn.xlarge)	CUDA 12.6, PyTorch 2.8
IoT Localization Service	Handles ESP32 data streams, performs navigation logic, and integrates SuperGlue for visual matching.	CPU (t3.medium)	Flask, OpenCV
VLM Solution Service	Logistic regression model to predict the UHI and Vision–Language reasoning using Gemini API to	CPU (t3.medium)	Flask, Gemini API

	generate mitigation suggestions		
Digital Twin Service	Simulates environment, visualizes results, and serves 3D scenarios	CPU (t3.large)	FastAPI, MATLAB Simscape integration

Each service operates in a microservice architecture, communicating via REST APIs and WebSockets, enabling independent scaling and upgrades.

3. AWS Infrastructure Setup

3.1. Elastic Container Registry (ECR)

- Created 4 private repositories
    - image-processing-service
    - iot-localization-service
    - vlm-solution-service
    - digital-twin-service
- Each GitHub Action pipeline builds, tags, and pushes a new image on main branch commits.

3.2. Elastic Container Service (ECS)

- Cluster Name : HeatScapeCluster
- Launch Type: EC2 (GPU + CPU capacity providers)
- Capacity Providers:
    - GPU Provider: Linked to g4dn.xlarge instances (for AI workloads)
    - CPU Provider: Linked to t3.* instances for lightweight APIs
- Each task definition defines its required CPU, memory, and GPU resources.

3.3. EC2 Configuration

Instance Type	Purpose	OS	Key Features
g4dn.xlarge	GPU-based AI inference (Image Processing)	Amazon Linux 2023	NVIDIA Tesla T4 GPU
t3.medium	Lightweight backend APIs	Amazon Linux 2023	Cost-optimized compute

Each EC2 instance is linked to the ECS cluster via the ECS_CLUSTER=HeatScapeCluster variable and IAM execution role.

3.4. Load Balancer (ALB)

- Type: Application Load Balancer
- Target Groups:
  - HeatScapeTargetGroupImg → port 5000
  - HeatScapeTargetGroupIoT → port 5000
  - HeatScapeTargetGroupVLM → port 5000
  - HeatScapeTargetGroupSim → port 5000
- Each service registered separately for routing through DNS.

4. CI/CD Automation

4.1 GitHub Actions Workflow

Each repository contains a .github/workflows/deploy.yml file executing the following steps:

1. **Checkout & Cleanup**

```
- uses: actions/checkout@v4
- name: Free Disk Space
  run: sudo rm -rf /usr/share/dotnet /opt/ghc /usr/local/lib/android
```

2. **AWS Credential Configuration**

```
- uses: aws-actions/configure-aws-credentials@v4
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: eu-north-1
```

3. **Docker Build and Push**

```
- uses: docker/setup-buildx-action@v3
- uses: aws-actions/amazon-ecr-login@v2
- uses: docker/build-push-action@v5
  with:
    context: .
    push: true
    file: ./Dockerfile
    tags: ${{ steps.ecr.outputs.registry }}/image-processing-service:latest
```

4. **ECS Deployment**

```
- uses: aws-actions/amazon-ecs-render-task-definition@v1
- uses: aws-actions/amazon-ecs-deploy-task-definition@v1
  with:
    cluster: HeatScapeCluster
    service: image-processing-service
    wait-for-service-stability: true
```

4.2. Deployment Trigger

- Triggered automatically on push to main.
- GitHub secrets store AWS credentials and ECR tokens securely.
- Pipeline logs confirm successful image push and ECS service update.

5. Service Health and Monitoring

5.1. Logging

- **AWS CloudWatch** integrated for all containers using the awslogs driver.
- Each service has its own log group:

```
/ecs/HeatScapeTask-*
```

- Real-time error tracking and performance metrics are visible in **CloudWatch Logs** and **CloudWatch Metrics** dashboards.

5.2. Health Checks

- **Grace Period:** 120 seconds (to accommodate AI model loading).
- **Check Endpoint:**

```
/health
```

- Returns **HTTP 200** after successful startup, confirming container readiness.

5.3. Scaling and Recovery

- **ECS** automatically restarts failed tasks using built-in service auto-healing.
- **Deployment Circuit Breaker** is enabled for automatic rollback on failed or unhealthy deployments.
- Supports **horizontal scaling** by increasing task count per service during high-load scenarios.

6. Security and Access Control

6.1. IAM Roles

- **ecsTaskExecutionRole** – Grants ECS permission to:
  - Pull images from **Amazon ECR**
  - Write logs to **AWS CloudWatch**
- **Service-specific roles** – Manage sensitive **environment variables** and control access to API keys or model configurations.

6.2. Networking

- **Private subnets** are used for all **EC2 instances**, ensuring internal communication security.
- **Application Load Balancer (ALB)** is placed in a **public subnet** to handle external API traffic.
- **Security Groups** are configured to allow inbound traffic only on **port 5000** (Flask/FastAPI services) and restrict all other ports by default.

7. Deployment Challenges & Solutions

Issue	Cause	Solution
GPU instance unavailability	Region AZ capacity shortage *(eu-north-1a/b/c)*	Switched to flexible AZ allocation and retried in multiple zones.
Circuit Breaker errors	Service failing due to long startup time	Added health check grace period = **180s**.
Model download hang (DepthAnything)	Hugging Face download delay	Pre-baked model weights into Docker image.
Disk space during build	GitHub runner limitation	Added cleanup script + Buildx cache push.

CORS policy issues	Missing preflight response	Configured Flask CORS(app, supports_credentials=True) with wildcard origins.

8. Final Verification

Service	Status	Endpoint	**GPU/CPU**
Image Processing	Running	http://heatscapeloadbalancer-.../api/image	GPU
IoT Localization	Running	http://heatscapeloadbalancer-.../api/iot	CPU
VLM Solution	Running	http://heatscapeloadbalancer-.../api/vlm	CPU
Digital Twin Simulation	Running	http://heatscapeloadbalancer-.../api/sim	CPU

All endpoints are accessible via the **Application Load Balancer (ALB)** and fully integrated into the **HeatScape web dashboard**.

9. Conclusion

The HeatScape deployment successfully demonstrates a production-grade, cloud-native architecture that enables real-time, scalable urban heat analysis through AI, IoT, and simulation pipelines.

The combination of GPU-backed ECS services, automated CI/CD, and microservice modularity ensures that the system can evolve with future research expansions — such as multi-city scaling, additional AI models, or integration with external APIs.

This deployment framework now serves as a blueprint for future environmental AI platforms, showcasing how cloud-native design and automation can operationalize academic research for real-world impact.