Technical Report: Vercel-like CLI Tool

Project Title: Frontend Deployment Automation CLI Tool (deploy-tool)

Author: Priyesh Rai

Company: Minfy Technologies

Position: Tech Intern

Date: July 22, 2025

Contents

Executive Summary

This report details the development of the deploy-tool, a custom command-line tool built to simplify and accelerate the process of deploying frontend web applications to Amazon Web Services (AWS).

- A custom Python CLI tool built for fast-paced environments like hackathons, where quick deployment is essential.
- Supports React, Next.js, and static frontend apps with zero manual AWS console interaction.
- Automates:
    - Infrastructure provisioning via Terraform
    - Containerization using Docker
    - Image storage using AWS ECR
    - Deployment through AWS ECS Fargate and Application Load Balancer (ALB)
- Offers rapid rollback to previous versions using version history.
- Includes integrated monitoring with Prometheus and Grafana, provisioned automatically.
- Greatly reduces setup time and avoids AWS configuration overhead.
- Enables one-command deployments, letting developers focus on coding, not infrastructure.
- Ideal for hackathons, POCs, or rapid prototyping where speed and simplicity matter most.

Introduction & Problem Statement

In time-sensitive environments like hackathons, developers are under pressure to build and deploy applications quickly. However, deploying frontend apps to AWS typically involves multiple complex steps — including Docker image creation, pushing to ECR, provisioning infrastructure with Terraform, configuring ECS and ALB, and setting up monitoring with Prometheus and Grafana.

This process is:

- **Time-consuming,** with many repetitive tasks
- **Manual and error-prone,** especially under tight deadlines
- **Unfriendly to beginners,** who may not be familiar with AWS services or deployment pipelines
- **Difficult to rollback,** which slows down iteration during rapid development cycles

There's a need for a streamlined, developer-friendly solution that automates the full deployment pipeline — from app detection to live URL — allowing teams to focus on innovation rather than infrastructure.

Project Goals & Objectives

Our primary goals for this project were to:

- **Automate:** Create a single tool to handle the entire deployment process.
- **Simplify:** Design a user-friendly command-line interface (CLI) that is easy for anyone to use.
- **Ensure Safety:** Provide a reliable, one-command rollback feature to revert to a previous version if a deployment fails.
- **Provide Visibility:** Automatically include a monitoring dashboard with every deployment.
- **Maintain Quality:** Build a reliable and maintainable solution using industry-best practices.

Scope & Limitations

It is important to define what this tool does and does not do.

In Scope:

- Automated deployment of containerized frontend applications (React, Next.js, static sites).
- Provisioning of all necessary infrastructure on AWS using Terraform (ECS, ALB, ECR, etc.).
- Automated setup of a monitoring stack (Prometheus & Grafana) for the deployed application.
- Management of development and production environments.

Out of Scope:

- Deployment of backend services or databases.
- Support for cloud providers other than AWS (e.g., Google Cloud, Azure).
- CI/CD integration (the tool is designed to be run manually by a developer or from within a separate CI/CD pipeline).
- Management of DNS records or SSL certificates.

Infrastructure Architecture with Terraform

The `deploy-tool` uses Terraform to define and provision the entire cloud infrastructure as code. This ensures that every deployment is consistent, repeatable, and automated. The infrastructure is logically separated into two main components, both managed by a single `terraform apply` command orchestrated by the Python script.

Application Stack

This stack is responsible for running the user's frontend application.

- **AWS ECR (Elastic Container Registry):** Terraform creates a private ECR repository to securely store the application's Docker images. The Python tool pushes the newly built image to this repository before running Terraform.
- **AWS ECS (Elastic Container Service) on Fargate:** Terraform defines an ECS Cluster, a Task Definition, and a Service. The Task Definition is a blueprint that tells ECS how to run the application container, referencing the image in ECR. Using the Fargate launch type allows the application to run in a serverless environment without the need to manage EC2 instances.
- **Application Load Balancer (ALB):** An ALB is provisioned to expose the application to the internet. Terraform configures the ALB with listeners and target groups to route public traffic to the running ECS container.

Monitoring Stack

This stack is responsible for observing the health and performance of the application.

- **EC2 Instance:** A dedicated EC2 instance is provisioned by Terraform to host the monitoring tools. This separation ensures that monitoring resources do not impact the performance of the main application.
- **Prometheus & Grafana:** These are installed on the EC2 instance. Terraform configures the necessary security groups to allow Prometheus to scrape metrics and to allow users to access the Grafana dashboard.
- **Blackbox & Node Exporter:** The Prometheus setup is configured to use two key exporters:
  - **Node Exporter:** Collects system-level metrics from the EC2 instance itself (CPU, memory, disk usage).
  - **Blackbox Exporter:** Probes the public endpoint of the main application running on ECS. This provides crucial "black-box" monitoring to test if the application is responsive, check SSL certificate validity, and measure response times.

The `deploy-tool` CLI acts as the orchestrator, dynamically providing the application's public URL to the Terraform configuration so that the Blackbox Exporter knows which endpoint to monitor.

Technology Stack

The CLI tool leverages a combination of languages, tools, and platforms to provide a fully automated, production-ready deployment workflow:

- **Python:** Core programming language used to build the CLI using:
  - **Click** for command-line interface structure
  - **subprocess** for shell command execution
  - **boto3** for AWS service integration
  - **json** for config and version history management
- **Docker:** Used to containerize frontend applications before deployment.
- **Terraform:** Automates infrastructure provisioning including VPCs, subnets, ECS, EFS, ALB, and IAM roles.
- **AWS Services:**
  - **ECS Fargate:** Serverless container hosting for frontend apps and monitoring stack
  - **ECR:** Secure container image storage
  - **ALB:** Application Load Balancer for routing traffic to services and Grafana dashboards
- **Prometheus & Grafana:** Integrated monitoring stack:
  - **Blackbox Exporter:** For HTTP endpoint probing (frontend)
  - **Node Exporter:** For system metrics (EC2, Fargate insights)
  - Dashboards are auto-provisioned via EFS volumes and displayed via ALB

Implementation Details & Key Features

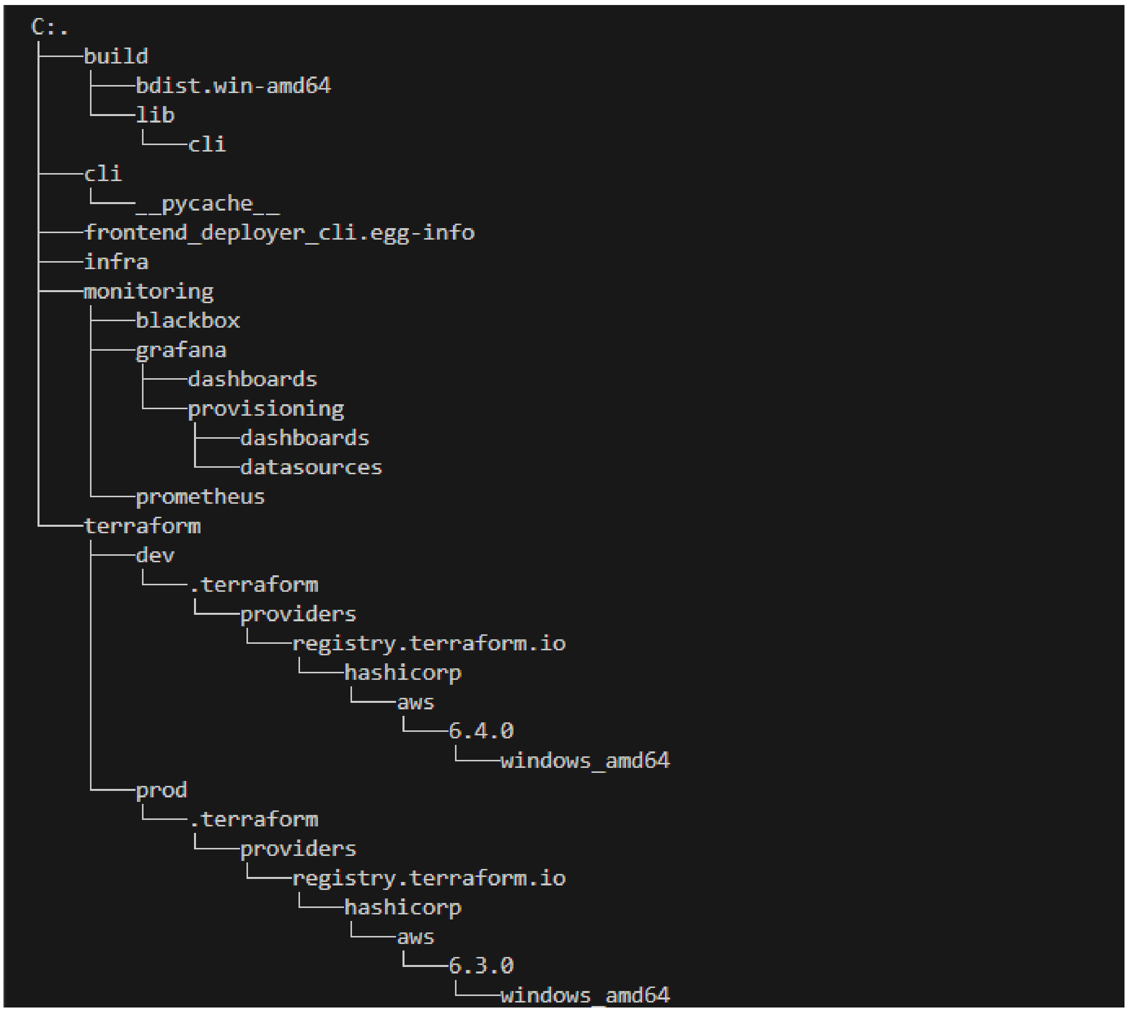
The tool's functionality is delivered through a set of simple, powerful commands.

Core Commands

Command	Description
deploy-tool init	Analyzes the project and creates initial configuration files.
deploy-tool config	Sets up the target AWS environment and region for deployment.
deploy-tool deploy	Builds, versions, and deploys the entire application and infrastructure stack.
deploy-tool monitoring	Checks the application's health and provides a link to the Grafana dashboard.
deploy-tool rollback	Reverts the live application to a previously deployed, stable version.

Project Folder Structure

The project is organized to separate the CLI source code, infrastructure code, and monitoring configurations, making it clean and maintainable.



```
C:.
├───build
│   ├───bdist.win-amd64
│   └───lib
│       └───cli
├───cli
│   └───__pycache__
├───frontend_deployer_cli.egg-info
├───infra
├───monitoring
│   ├───blackbox
│   ├───grafana
│   │   ├───dashboards
│   │   └───provisioning
│   │       ├───dashboards
│   │       └───datasources
│   └───prometheus
└───terraform
    ├───dev
    │   └───.terraform
    │       └───providers
    │           └───registry.terraform.io
    │               └───hashicorp
    │                   └───aws
    │                       └───6.4.0
    │                           └───windows_amd64
    └───prod
        └───.terraform
            └───providers
                └───registry.terraform.io
                    └───hashicorp
                        └───aws
                            └───6.3.0
                                └───windows_amd64
```

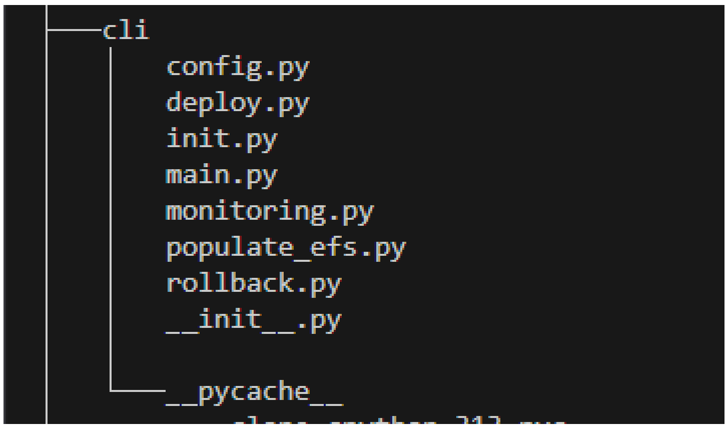
Deliverables

CLI Tool

A Python-based CLI tool (deploy-tool) built using the click library.

Automates full deployment lifecycle:

- `init`: Auto-detect frontend framework and generate `.deployconfig.json`.
- `config`: Choose AWS environment (dev/prod) and generate `.awsconfig.json`.
- `deploy`: Build Docker image, push to AWS ECR, update ECS service.
- `rollback`: Restore a previous version using ECS task definition history.
- status: Check ECS service health and open the Grafana dashboard.



```
│   ├───cli
│   │   │   config.py
│   │   │   deploy.py
│   │   │   init.py
│   │   │   main.py
│   │   │   monitoring.py
│   │   │   populate_efs.py
│   │   │   rollback.py
│   │   │   __init__.py
│   │   │
│   │   └───__pycache__
```

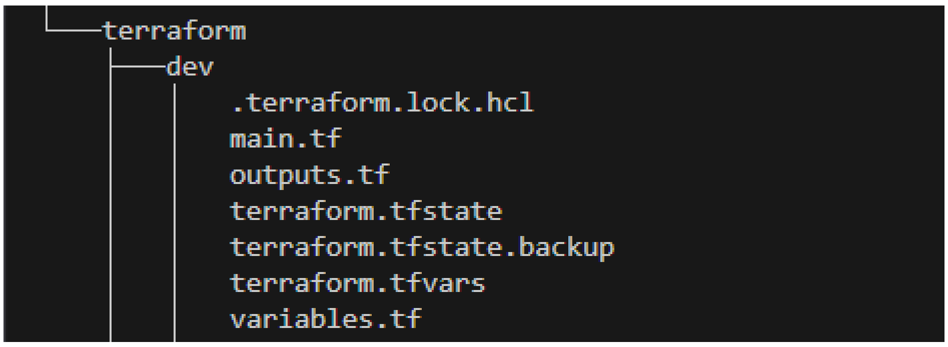
All version metadata managed via version.json.

Infrastructure as Code (IaC)

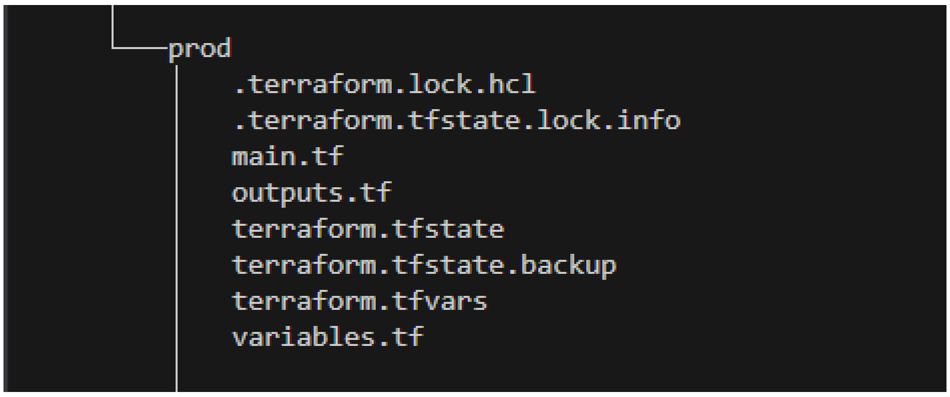
Terraform configurations to provision all required AWS infrastructure:

- ECS Cluster, Task Definitions (for frontend, Prometheus, Grafana)
- ECR Repositories for container storage
- ALB with custom listener rules for subpath routing (`/`, `/grafana`)
- EC2 instance with SSM access for provisioning config uploads
- Security Groups, IAM Roles, and autoscaling policies
- Supports isolated dev and prod environments with environment-specific state.

Terraform – Different infrastructure for Development & Production Environment



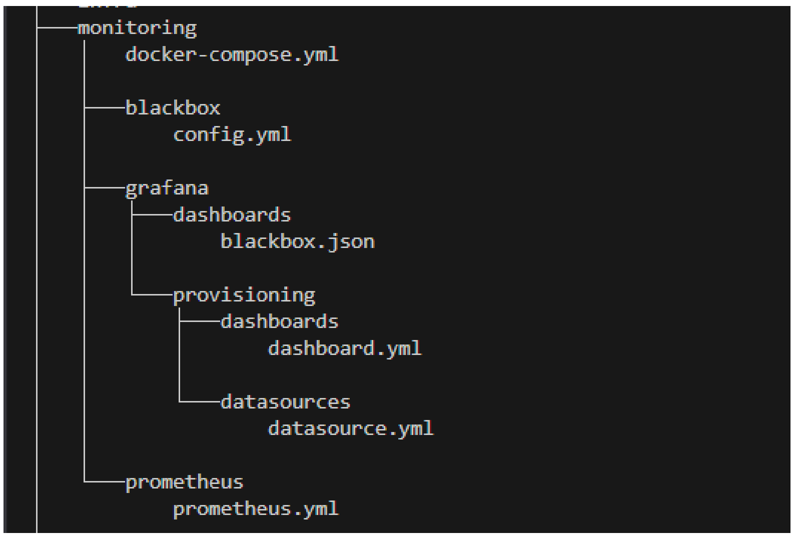
```
└───terraform
    ├───dev
    │   │   .terraform.lock.hcl
    │   │   main.tf
    │   │   outputs.tf
    │   │   terraform.tfstate
    │   │   terraform.tfstate.backup
    │   │   terraform.tfvars
    │   │   variables.tf
```



```
    └───prod
        │   .terraform.lock.hcl
        │   .terraform.tfstate.lock.info
        │   main.tf
        │   outputs.tf
        │   terraform.tfstate
        │   terraform.tfstate.backup
        │   terraform.tfvars
        │   variables.tf
        │
```

Monitoring Setup

- ECS sidecar containers for Prometheus and Grafana deployed alongside frontend app.
- Uses Blackbox Exporter (for HTTP checks) and Node Exporter (for EC2 metrics).
- Dashboard JSON and provisioning YAMLs stored locally and uploaded via CLI tool.
- Grafana auto-provisioned with CloudWatch and Prometheus as data sources.
- Public Grafana dashboard served via ALB on `/grafana`.



```
│   ├───monitoring
│   │   │   docker-compose.yml
│   │   │
│   │   ├───blackbox
│   │   │       config.yml
│   │   │
│   │   ├───grafana
│   │   │   ├───dashboards
│   │   │   │       blackbox.json
│   │   │   │
│   │   │   └───provisioning
│   │   │       ├───dashboards
│   │   │       │       dashboard.yml
│   │   │       │
│   │   │       └───datasources
│   │   │               datasource.yml
│   │   │
│   │   └───prometheus
│   │           prometheus.yml
```

Testing & Validation

To ensure the CLI tool works reliably in real-world scenarios, we tested each component across the deployment lifecycle:

- **Docker Image Testing**
  - Verified that the CLI correctly detected the frontend type and built optimized Docker images using multi-stage builds. Confirmed that only production-ready assets were included.
- **ALB Endpoint Verification**
  - After deployment, tested the public Application Load Balancer (ALB) URL to ensure:
    - The frontend app loaded successfully via `/`
    - Grafana dashboards were accessible via `/grafana`
- **Version Rollback Testing**
  - Deployed multiple versions of the app, then used the `rollback` command to restore older ECS task definitions. Confirmed that the application reverted and the ALB reflected the correct version.
- **Monitoring Validation (Without EFS)**
  - Monitoring configs (Prometheus + Grafana YAML files) were stored locally inside the CLI tool directory.
  - A dedicated EC2 instance (provisioned via Terraform) was used to upload these configs to the appropriate mount paths using AWS SSM.
  - Prometheus (with Blackbox and Node exporters) and Grafana were deployed as ECS sidecar containers.
  - Confirmed that Prometheus could scrape targets like the frontend app and EC2 system metrics.
  - Verified Grafana auto-loaded the dashboards and data sources based on the uploaded config files.

Results & Business Impact

The `deploy-tool` has successfully met its objectives and provides significant value.

- **Time Savings:** The deployment process, which previously took hours of manual work, can now be completed in under 10 minutes.
- **Reduced Errors:** By automating the process, the tool eliminates the risk of human error during manual configuration.
- **Increased Developer Velocity:** Developers can now deploy more frequently and with greater confidence, allowing them to release new features faster.
- **Improved Visibility:** The built-in monitoring gives teams immediate insight into application health, helping them identify and fix issues proactively.

Conclusion & Future Work

The `deploy-tool` is a powerful and effective solution that successfully simplifies cloud deployments on AWS. It empowers developers by automating complex tasks, improving safety, and providing essential visibility into application performance.

For future work, we recommend exploring the following enhancements:

- **A `destroy` command:** To safely and completely remove all project infrastructure from AWS, helping to manage costs.
- **CI/CD Integration:** Providing templates and guides for integrating the tool into popular CI/CD platforms like GitHub Actions.
- **Broader Framework Support:** Adding automated detection and configuration for other popular frameworks like Vue.js or Svelte.

Appendix:

Installation and Usage Guide

Step 1: Unzip the Folder Unzip the frontend-deployer-cli.zip file to a location of your choice on your local machine.

Step 2: Create a Virtual Environment (Recommended) It is best practice to create a virtual environment to avoid conflicts with other Python projects.

```
# Navigate into the unzipped folder
cd frontend-deployer-cli

# Create the virtual environment
python -m venv venv

# Activate the environment
# On Windows:
venv\Scripts\activate

# On macOS/Linux:
source venv/bin/activate
```

Step 3: Install the CLI Tool Use the included setup.py file to install the tool as a local Python package. This command registers the deploy-tool command on your system.

```
pip install .
```

Step 4: Verify the Installation Check that the CLI tool has been installed correctly by asking for help.

```
deploy-tool --help
```

You should see a list of available commands, such as init, config, deploy, rollback, and monitoring.

Step 5: Use the Tool You can now use the tool to deploy your projects.

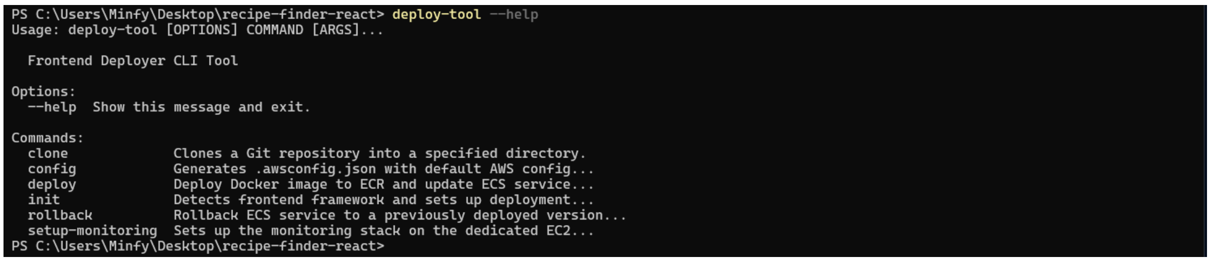
```
# Initialize a project
deploy-tool init

# Configure the AWS environment
deploy-tool config

# Deploy the application
deploy-tool deploy
```

Prerequisites Before using the tool, ensure the following are installed and configured on your system:

- Python 3.7+
- Docker (must be installed and running)
- AWS CLI (must be installed and configured with credentials or SSO)
- Terraform (must be installed and accessible in your system's PATH)

Screenshots of Functionality :

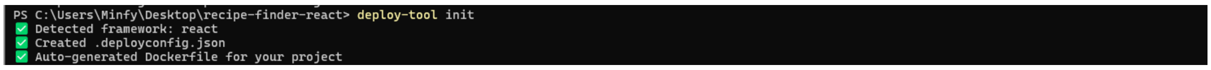
Figure 0: landing command of the CLI tool



```
PS C:\Users\Minfy\Desktop\recipe-finder-react> deploy-tool --help
Usage: deploy-tool [OPTIONS] COMMAND [ARGS]...

  Frontend Deployer CLI Tool

Options:
  --help  Show this message and exit.

Commands:
  clone             Clones a Git repository into a specified directory.
  config            Generates .awsconfig.json with default AWS config...
  deploy            Deploy Docker image to ECR and update ECS service...
  init              Detects frontend framework and sets up deployment...
  rollback          Rollback ECS service to a previously deployed version...
  setup-monitoring  Sets up the monitoring stack on the dedicated EC2...
PS C:\Users\Minfy\Desktop\recipe-finder-react>
```

Comment: Gets you aquinted by all the comments in the CLI tool

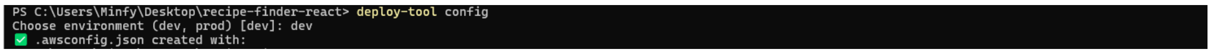
Figure 1: deploy-tool init



```
PS C:\Users\Minfy\Desktop\recipe-finder-react> deploy-tool init
✅ Detected framework: react
✅ Created .deployconfig.json
✅ Auto-generated Dockerfile for your project
```

Comment: This screenshot shows the output of the init command. The tool has successfully detected the project type (e.g., React) and created the initial .deployconfig.json file.

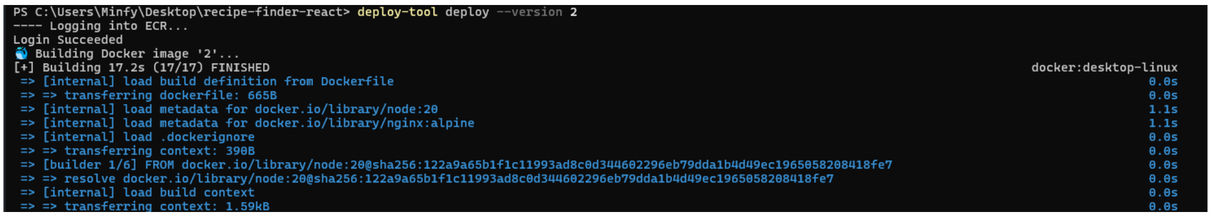
Figure 2: deploy-tool config



```
PS C:\Users\Minfy\Desktop\recipe-finder-react> deploy-tool config
Choose environment (dev, prod) [dev]: dev
✅ .awsconfig.json created with:
```

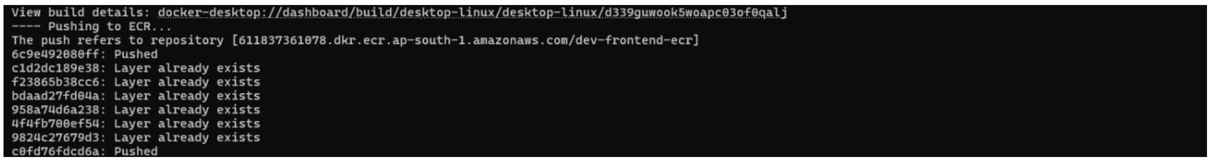
Comment: This shows the config command in action. The user is prompted to select an environment (dev or prod), and the tool then creates the .awsconfig.json file with the appropriate settings.

Figure 3: deploy-tool deploy



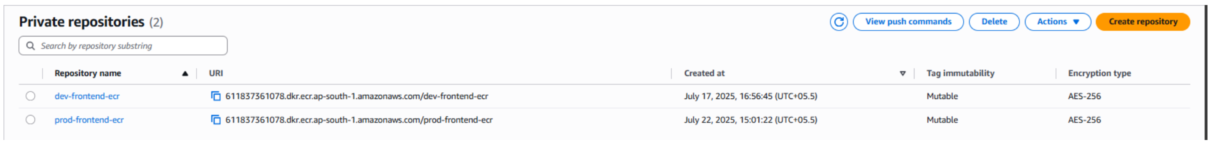
```
PS C:\Users\Minfy\Desktop\recipe-finder-react> deploy-tool deploy --version 2
---- Logging into ECR...
Login Succeeded
🐳 Building Docker image '2'...
[+] Building 17.2s (17/17) FINISHED                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                 0.0s
 => => transferring dockerfile: 665B                                                                                 0.0s
 => [internal] load metadata for docker.io/library/node:20                                                           1.1s
 => [internal] load metadata for docker.io/library/nginx:alpine                                                      1.1s
 => [internal] load .dockerignore                                                                                    0.0s
 => => transferring context: 390B                                                                                    0.0s
 => [builder 1/6] FROM docker.io/library/node:20@sha256:122a9a65b1f1c11993ad8c0d344602296eb79dda1b4d49ec1965058208418fe7   0.0s
 => => resolve docker.io/library/node:20@sha256:122a9a65b1f1c11993ad8c0d344602296eb79dda1b4d49ec1965058208418fe7          0.0s
 => [internal] load build context                                                                                    0.0s
 => => transferring context: 1.59kB                                                                                  0.0s
```

Comment: This screenshot captures the output of the core deploy command. It shows the various stages of the process: building the Docker image.



```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/d339guwook5woapc03of0qalj
---- Pushing to ECR...
The push refers to repository [611837361078.dkr.ecr.ap-south-1.amazonaws.com/dev-frontend-ecr]
6c9e492080ff: Pushed
c1d2dc189e38: Layer already exists
f23865b38cc6: Layer already exists
bdaad27fd04a: Layer already exists
958a74d6a238: Layer already exists
4f4fb700ef54: Layer already exists
9824c27679d3: Layer already exists
c0fd76fdcd6a: Pushed
```

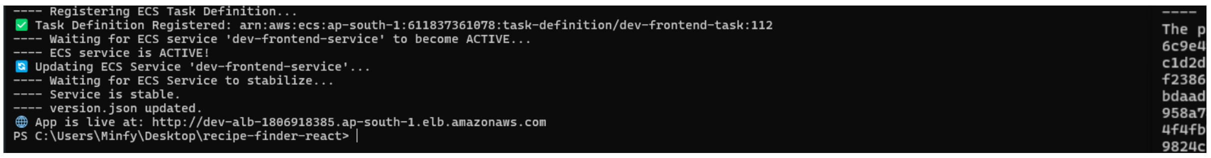
Comment: pushing it to ECR, and running terraform apply.


Private repositories (2)

Search by repository substring

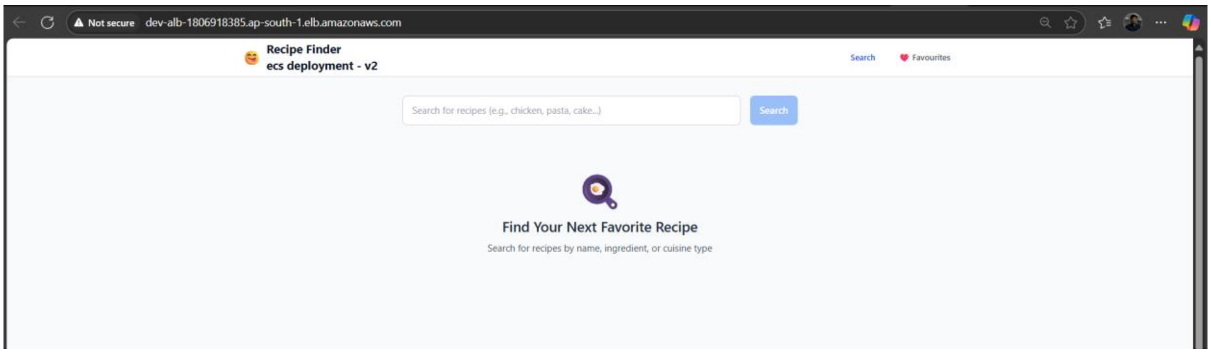
	Repository name	URI	Created at	Tag immutability	Encryption type
○	dev-frontend-ecr	611837361078.dkr.ecr.ap-south-1.amazonaws.com/dev-frontend-ecr	July 17, 2025, 16:56:45 (UTC+05.5)	Mutable	AES-256
○	prod-frontend-ecr	611837361078.dkr.ecr.ap-south-1.amazonaws.com/prod-frontend-ecr	July 22, 2025, 15:01:22 (UTC+05.5)	Mutable	AES-256

Comment: We are maintaining two different ECR to separate the Development Environment & the Production Environment


```
---- Registering ECS Task Definition...
✅ Task Definition Registered: arn:aws:ecs:ap-south-1:611837361078:task-definition/dev-frontend-task:112
---- Waiting for ECS service 'dev-frontend-service' to become ACTIVE...
---- ECS service is ACTIVE!
🔄 Updating ECS Service 'dev-frontend-service'...
---- Waiting for ECS Service to stabilize...
---- Service is stable.
---- version.json updated.
🌐 App is live at: http://dev-alb-1806918385.ap-south-1.elb.amazonaws.com
PS C:\Users\Minfy\Desktop\recipe-finder-react> |
```

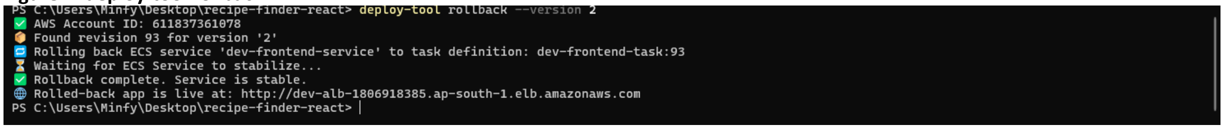
```
----
The p
6c9e4
c1d2d
f2386
bdaad
958a7
4f4fb
9824c
```

Comment: updating the task definition of the ECS service to pull that particular entry – Docker Image

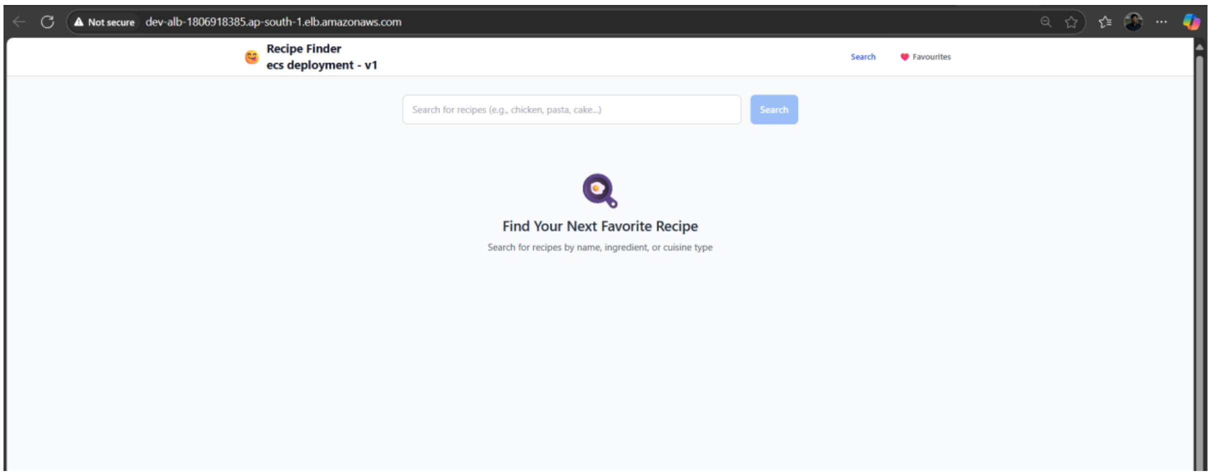

Not secure dev-alb-1806918385.ap-south-1.elb.amazonaws.com
Recipe Finder
ecs deployment - v2
Search
Favourites
Search for recipes (e.g., chicken, pasta, cake...)
Search
Find Your Next Favorite Recipe
Search for recipes by name, ingredient, or cuisine type

Comment : The final output shows the public URL for the deployed application.

Figure 4: deploy-tool rollback

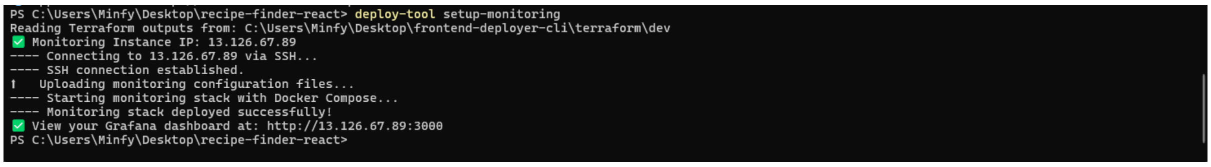

```
PS C:\Users\Minfy\Desktop\recipe-finder-react> deploy-tool rollback --version 2
✅ AWS Account ID: 611837361078
📦 Found revision 93 for version '2'
🔁 Rolling back ECS service 'dev-frontend-service' to task definition: dev-frontend-task:93
⏳ Waiting for ECS Service to stabilize...
✅ Rollback complete. Service is stable.
🌐 Rolled-back app is live at: http://dev-alb-1806918385.ap-south-1.elb.amazonaws.com
PS C:\Users\Minfy\Desktop\recipe-finder-react> |
```

Comment: **This demonstrates a successful rollback.**


Not secure dev-alb-1806918385.ap-south-1.elb.amazonaws.com
Recipe Finder
ecs deployment - v1
Search
Favourites
Search for recipes (e.g., chicken, pasta, cake...)
Search
Find Your Next Favorite Recipe
Search for recipes by name, ingredient, or cuisine type

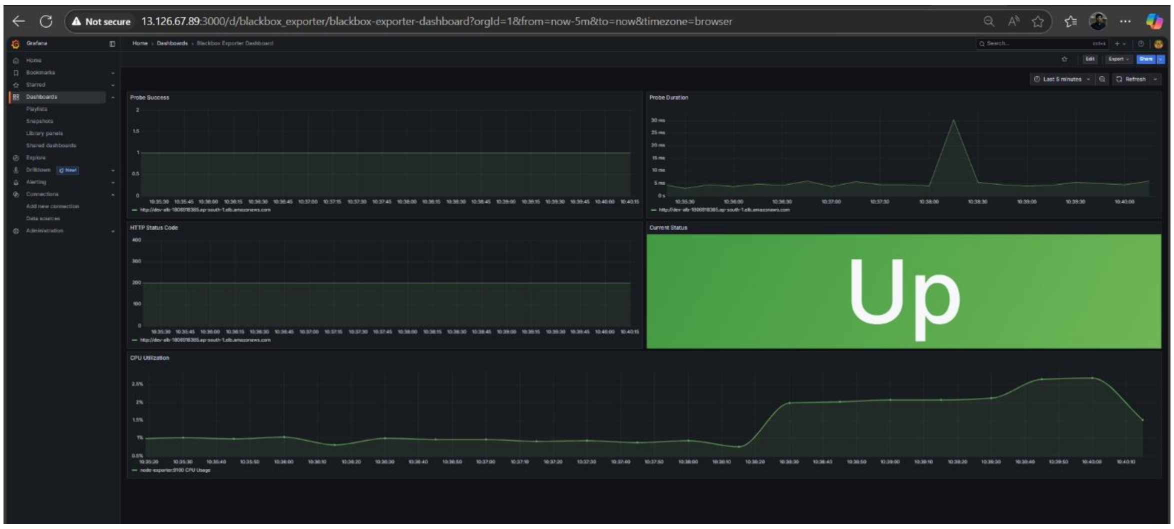
Comment: The output confirms that the tool is reverting the ECS service to a previously specified version.

Figure 5: deploy-tool monitoring



```
PS C:\Users\Minfy\Desktop\recipe-finder-react> deploy-tool setup-monitoring
Reading Terraform outputs from: C:\Users\Minfy\Desktop\frontend-deployer-cli\terraform\dev
✅ Monitoring Instance IP: 13.126.67.89
---- Connecting to 13.126.67.89 via SSH...
---- SSH connection established.
⬆   Uploading monitoring configuration files...
---- Starting monitoring stack with Docker Compose...
---- Monitoring stack deployed successfully!
✅ View your Grafana dashboard at: http://13.126.67.89:3000
PS C:\Users\Minfy\Desktop\recipe-finder-react>
```

Comment: This screenshot shows the output of the monitoring (or status) command.


Not secure
13.126.67.89:3000/d/blackbox_exporter/blackbox-exporter-dashboard?orgId=1&from=now-5m&to=now&timezone=browser
Grafana
Home › Dashboards › Blackbox Exporter Dashboard
Probe Success
Probe Duration
HTTP Status Code
Current Status
Up
CPU Utilization

Comment: **It verifies that the application is running and provides a direct, clickable link to the Grafana monitoring dashboard.**

References :

1. *https://docs.aws.amazon.com/AmazonECS/latest/developerguide/security-fargate.html*
2. *https://aws.amazon.com/blogs/compute/building-deploying-and-operating-containerized-applications-with-aws-fargate/*
3. *https://aws.amazon.com/sdk-for-python/*

4. *https://boto3.amazonaws.com/v1/documentation/api/latest/guide/examples.html*
5. *https://realpython.com/comparing-python-command-line-parsing-libraries-argparse-docopt-click/*

6. *https://github.com/prometheus/cloudwatch_exporter*
7. *https://github.com/santosh-at-github/ecs-monitoring-with-prometheus-grafana*
8. *https://blog.devops.dev/monitoring-using-prometheus-and-grafana-on-aws-ec2-which-is-built-with-terraform-f05cdd67a0ef*
9. *https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ContainerInsights-Prometheus-Setup-configure-ECS.html*

10. *https://3h4x.github.io/tech/2020/02/07/prometheus-on-ecs-poc*
11. *https://terrateam.io/blog/deploying-grafana-with-terraform*
12. *https://github.com/56kcloud/terraform-grafana?tab=readme-ov-file*
13. *https://grapeup.com/blog/monitoring-your-microservices-on-aws-with-terraform-and-grafana-monitoring/#*
14. *https://stackoverflow.com/questions/62018089/how-to-upload-local-files-to-aws-elastic-file-system-efs*