**Terraform Setup Amendments**

### **Overview**

This document outlines amendments to the existing *MessageAI MVP - Development Task List* to integrate **Terraform** into the infrastructure setup and provisioning workflow. Terraform will replace several manual configuration steps for **Railway, AWS, and environment management**, improving reproducibility, scalability, and CI/CD automation.

## **1. New Tooling and Repository Structure**

**Add Terraform Directory**

gauntlet-messageai-24hr-mvp/

├── infrastructure/

│ ├── terraform/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ ├── outputs.tf

│ │ ├── aws/

│ │ ├── railway/

│ │ ├── providers.tf

│ │ └── README.md

**Tooling Changes:**

* Install Terraform CLI locally (>=1.7.0).
* Configure Terraform Cloud or S3 + DynamoDB backend for remote state management.
* Define providers: aws, railway, random, null, and local.

**Reasoning:** This structure allows modular provisioning of backend infrastructure (AWS services, Railway environments, etc.) with isolated state management per environment (dev/staging/prod).

## **2. Phase 0: Pre-Development Setup (Modified)**

### **Original Tasks Affected**

* **Task 0.2:** Railway Account Setup
* **Task 0.3:** AWS Account Setup
* **Task 0.4:** Expo Account Setup
* **Task 0.5:** Firebase Setup

### **Amendments**

**Task 0.2 – Railway Infrastructure via Terraform**

* Replace manual creation of the Railway project and PostgreSQL instance with Terraform-managed resources using the Railway provider or REST API via terraform-provider-railway or null\_resource + API calls.
* Automatically export environment variables (DATABASE\_URL, RAILWAY\_PROJECT\_ID, RAILWAY\_SERVICE\_DOMAIN) to .env via Terraform output.

**Task 0.3 – AWS Infrastructure via Terraform**

* Replace manual AWS Console setup with a Terraform module under infrastructure/terraform/aws.  
  + **Resources created:**
    - S3 bucket for messageai-media-\* with CORS config
    - SQS queue messageai-notification-queue
    - Lambda functions for notification-worker and cleanup-jobs (linked to code in /aws-lambdas)
    - IAM role/policies for S3/Lambda/SQS/EventBridge
    - CloudWatch log groups for monitoring

Environment variables exported via Terraform outputs:  
  
 AWS\_ACCESS\_KEY\_ID

AWS\_SECRET\_ACCESS\_KEY

AWS\_REGION

AWS\_S3\_BUCKET

AWS\_SQS\_QUEUE\_URL

**Task 0.4 – Expo Configuration (No Change)**

* Expo setup remains manual (account tokens cannot be provisioned via Terraform).

**Task 0.5 – Firebase Configuration (Partial Change)**

* Firebase can be partially automated with Terraform’s Google provider to create a project skeleton and retrieve server keys.
* Still requires manual download of google-services.json and GoogleService-Info.plist.

**New Task 0.6 – Terraform Init & Apply**

Initialize Terraform project:  
  
 cd infrastructure/terraform

terraform init

terraform plan -out=tfplan

terraform apply tfplan

* Store remote state (S3/DynamoDB or Terraform Cloud) for team collaboration.

## **3. Phase 1: Project Initialization (Slight Reorder)**

### **Amendment Summary**

* Move Terraform provisioning **before** npm install and backend initialization so that .env variables are available automatically.

### **New Order:**

1. Run Terraform provisioning (create infrastructure + outputs).

Generate .env files automatically using Terraform output to local file:  
  
 terraform output -json > ../.env.auto.json

1. Continue with monorepo setup, backend/.env population, and project scaffolding.

## **4. Phase 2: Database & Authentication**

**Task 2.2 – Run Initial Migration**

* Update to reference the Terraform-managed PostgreSQL database connection string.
* No manual copy/paste of DATABASE\_URL required.

## **5. Phase 6: Deployment & CI/CD Integration (New Task Addition)**

**New Task 6.x – CI/CD Terraform Integration**

* Add GitHub Actions workflow .github/workflows/terraform.yml for automated infrastructure validation.

**Example Workflow:**

name: Terraform CI

on:

push:

branches: [ main ]

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: hashicorp/setup-terraform@v3

with:

terraform\_version: 1.7.0

- run: cd infrastructure/terraform && terraform fmt -check && terraform validate

- run: terraform plan -no-color

**Optional Extension:** Configure GitHub Actions secrets for AWS and Railway credentials, ensuring non-interactive provisioning for future environments.

## **6. Benefits of Terraform Integration**

| **Benefit** | **Description** |
| --- | --- |
| **Reproducibility** | All cloud infrastructure defined in code and version-controlled. |
| **Scalability** | Enables repeatable deployment of staging and production environments. |
| **Automation** | Eliminates manual steps in AWS and Railway setup, reducing human error. |
| **State Tracking** | Terraform state provides a single source of truth for infrastructure configuration. |
| **CI/CD Alignment** | Allows infrastructure validation and change automation as part of GitHub Actions. |

## **7. Summary of Task Order Changes**

| **Original Order** | **Revised Order** | **Key Change** |
| --- | --- | --- |
| Task 0.2 → Task 0.3 (Manual Railway/AWS Setup) | Task 0.2 → Task 0.6 (Terraform Setup & Apply) | Replace manual setup with Terraform provisioning. |
| Task 1.1 (Project Initialization) | After Terraform Apply | Terraform must complete first to populate environment variables. |
| Task 2.2 (Prisma Migration) | Unchanged but depends on Terraform outputs | DB credentials pulled from Terraform outputs. |

### **✅ Deliverables After Amendments**

* /infrastructure/terraform/ directory with provider configs and modules
* GitHub Actions workflow for Terraform validation
* Automated .env generation from Terraform outputs
* Reduced manual cloud configuration

**End of Document**