Terraform Infrastructure Guide

DataMigrate AI

Alexander Garcia Angus

OKO Investments

Overview

DataMigrate AI uses **Terraform** to manage all AWS infrastructure as code. This provides version control, reproducibility, and team collaboration for infrastructure changes.

Why Terraform?

1. Infrastructure as Code (IaC)
- Infrastructure defined in version-controlled files
- Track changes over time with Git
- Review infrastructure changes like code reviews

2. Reproducible Environments
- Identical dev, staging, and production environments
- No configuration drift
- Easy disaster recovery

3. Multi-Cloud Support
- Works with AWS, Azure, GCP, and 100+ providers
- Future flexibility for OKO Investments

4. Team Collaboration
- State managed in S3 + DynamoDB
- Prevent concurrent modifications (state locking)
- Clear ownership and change history

5. Cost Transparency
- See exactly what resources are deployed
- Easy to tear down unused environments
- Estimate costs before deployment

Architecture Deployed

Terraform deploys a **production-ready, scalable architecture** with these components:

Component	Technology	Purpose
Frontend	S3 + CloudFront	Vue.js static hosting with CDN
API Gateway	ALB (HTTPS)	Load balancing and SSL termination
Backend	ECS Fargate	FastAPI containers with auto-scaling
Database	RDS PostgreSQL 15	Multi-AZ, encrypted, automated backups
Cache	ElastiCache Redis	Session storage and caching
Network	VPC (3 AZs)	Public, private, and database subnets

Network Architecture:

- **VPC** with 3 availability zones for high availability
- **Public Subnets:** ALB, NAT Gateways (internet-facing)
- **Private Subnets:** ECS tasks (FastAPI containers, no direct internet)
- **Database Subnets:** RDS (completely isolated, no internet)
- **Security Groups:** Least privilege access control

Quick Start

1. Prerequisites

```
# Install Terraform
brew install terraform # macOS
choco install terraform # Windows

# Install AWS CLI
brew install awscli

# Configure AWS credentials
aws configure
```

2. Deploy Infrastructure

```
cd terraform

# Initialize Terraform
terraform init

# Preview changes
terraform plan

# Deploy (takes 15-20 minutes)
terraform apply

# Get outputs
terraform output
```

Terraform Modules

Infrastructure is organized into reusable modules for maintainability:

Module	Purpose	Resources Created
vpc	Network infrastructure	VPC, Subnets, NAT Gateways, Route Tables
security	Security groups	ALB SG, ECS SG, RDS SG
rds	PostgreSQL database	RDS instance, Parameter group, Backups, Alarms
ecs	FastAPI backend	ECS cluster, Service, Tasks, ALB, Auto-scaling
s3_cloudfront	Vue.js frontend	S3 bucket, CloudFront CDN, SSL certificate

Cost Breakdown (Development Environment)

Service	Monthly Cost	Notes
NAT Gateways (3)	$32.40	3 AZs for high availability
RDS PostgreSQL	$14.00	db.t3.micro (free tier eligible)
ECS Fargate	$25.00	2 tasks (0.5 vCPU, 1GB RAM)
Application Load Balancer	$16.00	HTTPS with SSL certificate
S3 + CloudFront	$10.50	Static hosting + CDN
CloudWatch Logs	$5.00	30-day retention
TOTAL	**$105/month**	**Development environment**

Cost Optimization Tips:

- Use VPC Endpoints instead of NAT Gateways (saves $32/month)
- RDS Reserved Instances provide 40-60% savings
- Scale down ECS tasks during off-hours (nights/weekends)
- S3 Intelligent Tiering for automatic cost optimization

Security Features

Network Isolation
- Database in private subnets (no internet access)
- ECS tasks in private subnets
- ALB in public subnets only

Encryption
- RDS storage encrypted with AES-256
- S3 bucket encryption enabled
- HTTPS only (CloudFront, ALB with SSL)

Access Control
- IAM roles (no hardcoded credentials)
- Security groups with least privilege
- VPC Flow Logs for network monitoring

Compliance
- Automated backups (7 days retention)
- Multi-AZ deployment for high availability
- Deletion protection on production resources
- Audit logging with CloudTrail

Common Terraform Commands

```
# Initialize (first time or after adding modules)
terraform init

# Preview changes
terraform plan

# Apply changes
terraform apply

# Destroy all resources
terraform destroy

# Show current state
terraform show

# Get specific output
terraform output api_endpoint

# Format code
terraform fmt -recursive

# Validate configuration
terraform validate
```

Best Practices

1. Never Commit Secrets
- Use AWS Secrets Manager for database passwords
- Never put credentials in terraform.tfvars
- .tfvars files are gitignored

2. Use Remote State
- State stored in S3 (team collaboration)
- State locking with DynamoDB (prevent conflicts)
- Versioning enabled on S3 bucket

3. Test in Dev First
- Always apply changes to dev environment first
- Verify everything works before staging/prod
- Use terraform plan to preview changes

4. Tag Everything
- All resources tagged with Environment, Project, Owner
- Easy cost tracking and resource management
- Automated tagging via default_tags

5. Use Modules
- DRY principle (Don't Repeat Yourself)
- Reusable infrastructure components
- Easier to maintain and update

Integration with DataMigrate AI

Frontend Deployment

```
# Build Vue.js app
cd frontend
npm run build

# Deploy to S3
aws s3 sync dist/ s3://$(terraform output -raw frontend_bucket) --delete

# Invalidate CloudFront cache
aws cloudfront create-invalidation \
--distribution-id $(terraform output -raw cloudfront_distribution_id) \
--paths "/*"
```

Backend Deployment

```
# Build and push Docker image
docker build -t datamigrate-ai/fastapi:latest .
docker tag datamigrate-ai/fastapi:latest :latest
docker push :latest

# ECS will auto-deploy new image
aws ecs update-service \
--cluster $(terraform output -raw ecs_cluster_name) \
--service $(terraform output -raw ecs_service_name) \
--force-new-deployment
```

Environment Configurations

Aspect	Development	Production
Monthly Cost	$105	$500-1,000
RDS Instance	db.t3.micro	db.t3.medium+ Multi-AZ
ECS Tasks	2 tasks	4-20 tasks (auto-scaling)
Backup Retention	3 days	7 days
Features	Basic	Performance Insights, WAF

Next Steps:
1. Review the Terraform code in the /terraform directory
2. Customize terraform.tfvars for your environment
3. Deploy to AWS with terraform apply
4. Set up CI/CD pipeline for automated deployments