# Multi-Account Droppy AWS ECS Deployment Guide with Terraform

This repository contains **Terraform** configurations to deploy the **Droppy** file server application across a **multi-account AWS architecture**, leveraging **Amazon ECS on Fargate** for highly available and scalable hosting. It establishes secure cross-account networking and provides a convenient **Jump Host** for access.

**Architecture Overview**

This setup orchestrates resources across two AWS accounts:

**Account A: Droppy Application Core**

* **VPC (e.g., vpc-a)**: A dedicated VPC with a /16 CIDR block.
  + **Networking**: Configured with a single **NAT Gateway** in a public subnet for outbound internet access from private subnets.
  + **Subnets**: Includes 2 public, 2 private load-balancer, 2 private ECS, and 2 intra (isolated) subnets (each /24).
  + **VPC Peering**: Peered with **VPC B** for cross-account communication.
* **Application Load Balancer (ALB)**: An **internal** ALB deployed within the private load-balancer subnets to distribute traffic to the Droppy service.
* **ECS on Fargate**: The **Droppy service** runs on **ECS Fargate** with 2 replicas, deployed within the private ECS subnets.
* **EFS (Elastic File System)**: Used for **persistent data** storage for Droppy, with mount targets located in the isolated intra subnets for enhanced security.

**Account B: DNS & Access**

* **VPC (e.g., vpc-b)**: A dedicated VPC with a /23 CIDR block.
  + **Networking**: Contains a single public subnet (/24).
  + **VPC Peering**: Peered with **VPC A**.
* **Route 53 Private Hosted Zone**: Manages private DNS for the droppy.lan domain.
  + A **DNS record (app.droppy.lan)** is configured to point to the **internal ALB** in **Account A**.
  + The Private Hosted Zone is **associated with both VPC A and VPC B** to ensure seamless name resolution.
* **Jump Host**: A **Windows EC2 instance** provisioned in the public subnet, providing **RDP access** to securely reach the Droppy application in the browser via its internal DNS name.

**Getting Started**

To deploy this infrastructure, you will need:

* **Terraform CLI** installed.
* **AWS CLI** configured with credentials for both Account A and Account B, using named profiles (e.g., account-a and account-b).
* Your SSH public key for the Jump Host.

Refer to the account-a/ and account-b/ directories for their respective Terraform configurations.

**🏗️ Architecture Overview**

This deployment creates a secure, multi-account setup with:

**Account A (Application Account)**

* **VPC A** (10.0.0.0/16) with multiple subnet tiers
* **ECS Fargate** cluster running 2 Droppy replicas
* **Internal ALB** for load balancing
* **EFS** for persistent file storage
* **NAT Gateway** for outbound internet access

**Account B (Access Account)**

* **VPC B** (10.1.0.0/23) with public subnet
* **Windows Jump Host** for secure access
* **Route53 Private Hosted Zone** (droppy.lan)
* **VPC Peering** to Account A

**📋 Prerequisites**

1. **Two AWS Accounts** with appropriate permissions
2. **AWS CLI** installed and configured
3. **Terraform** >= 1.5 installed
4. **SSH key pair** for jump host access

**🚀 Step-by-Step Deployment**

**Step 1: Setup Directory Structure**

Run the setup script to create all necessary files:

# Save the setup script content to setup-terraform-files.sh

chmod +x setup-terraform-files.sh

./setup-terraform-files.sh

cd droppy-deployment

**Step 2: Configure AWS CLI Profiles**

Configure both AWS accounts:

# Account A configuration

aws configure --profile account-a

# Enter: Access Key ID, Secret Access Key, Region (e.g., us-east-1), Output format (json)

# Account B configuration

aws configure --profile account-b

# Enter: Access Key ID, Secret Access Key, Region (same as Account A), Output format (json)

# Test the profiles

aws sts get-caller-identity --profile account-a

aws sts get-caller-identity --profile account-b

**Step 3: Generate SSH Key Pair**

# Generate key pair for jump host access

./scripts/generate-keypair.sh

# This will create:

# ~/.ssh/jumphost-key (private key)

# ~/.ssh/jumphost-key.pub (public key)

**Step 4: Configure Account A**

cd account-a

cp terraform.tfvars.example terraform.tfvars

# Edit terraform.tfvars with your specific values:

nano terraform.tfvars

Example terraform.tfvars for Account A:

aws\_region = "us-east-1"

aws\_profile\_a = "account-a" # Match your AWS CLI profile name

vpc\_a\_cidr = "10.0.0.0/16"

vpc\_b\_cidr = "10.1.0.0/23"

**Step 5: Deploy Account A Infrastructure**

# Initialize Terraform

terraform init

# Validate configuration

terraform validate

# Review the deployment plan

terraform plan

# Deploy infrastructure

terraform apply

**Important**: Account A must be deployed first as Account B depends on its outputs.

**Step 6: Configure Account B**

cd ../account-b

cp terraform.tfvars.example terraform.tfvars

# Edit terraform.tfvars with your values:

nano terraform.tfvars

Example terraform.tfvars for Account B:

aws\_region = "us-east-1"

aws\_profile\_a = "account-a" # Account A profile name

aws\_profile\_b = "account-b" # Account B profile name

vpc\_a\_cidr = "10.0.0.0/16"

vpc\_b\_cidr = "10.1.0.0/23"

public\_key = "ssh-rsa AAAAB3NzaC1yc2E... your-actual-public-key-content"

allowed\_rdp\_cidr = "YOUR.IP.ADDRESS.0/24" # Restrict RDP access

**Step 7: Deploy Account B Infrastructure**

# Initialize Terraform

terraform init

# Validate configuration

terraform validate

# Review the deployment plan

terraform plan

# Deploy infrastructure

terraform apply

**Step 8: Access Droppy Application**

After successful deployment:

# Get the jump host public IP

terraform output jumphost\_public\_ip

1. **RDP to Jump Host**: Use the public IP with Windows RDP client
2. **Open Browser**: Launch Chrome/Firefox on the jump host
3. **Access Droppy**: Navigate to http://app.droppy.lan

**🔧 Alternative: Automated Deployment**

Use the provided deployment script for automated setup:

# Run the automated deployment script

./scripts/deploy.sh

This script will:

* Check prerequisites
* Deploy Account A infrastructure
* Deploy Account B infrastructure
* Provide connection details

**📊 Verification Commands**

**Check ECS Service Status**

aws ecs describe-services \

--cluster droppy-cluster \

--services droppy-service \

--profile account-a

**Check ALB Target Health**

# Get target group ARN from terraform output

terraform output -raw alb\_arn

# Check target health

aws elbv2 describe-target-health \

--target-group-arn <target-group-arn> \

--profile account-a

**Check Route53 DNS Resolution**

# From jump host, test DNS resolution

nslookup app.droppy.lan

**🔒 Security Considerations**

1. **Jump Host Access**:
   * Change allowed\_rdp\_cidr to your specific IP range
   * Use strong passwords for Windows administrator account
2. **Network Isolation**:
   * ALB is internal-only (no internet access)
   * ECS tasks run in private subnets
   * EFS is in isolated subnets
3. **Encryption**:
   * EFS filesystem encrypted at rest
   * All data in transit encrypted

**🧹 Cleanup**

To destroy all resources:

# Use the cleanup script

./scripts/cleanup.sh

# Or manually destroy in reverse order:

cd account-b && terraform destroy

cd ../account-a && terraform destroy

**🐛 Troubleshooting**

**Common Issues**

1. **VPC Peering Not Working**:
   * Verify both route tables have peering routes
   * Check security group rules
   * Ensure DNS resolution is enabled
2. **ECS Tasks Not Starting**:
   * Check CloudWatch logs: /ecs/droppy
   * Verify EFS mount targets are ready
   * Check security group connectivity
3. **DNS Resolution Issues**:
   * Verify Route53 zone association with both VPCs
   * Check DNS hostname/resolution settings on VPCs
4. **Jump Host Connection Issues**:
   * Verify security group allows RDP (port 3389)
   * Check that jump host has public IP
   * Ensure key pair is correctly configured

**Useful Commands**

# Check ECS task logs

aws logs tail /ecs/droppy --follow --profile account-a

# List Route53 records

aws route53 list-resource-record-sets \

--hosted-zone-id <zone-id> \

--profile account-b

# Check VPC peering status

aws ec2 describe-vpc-peering-connections \

--profile account-b

**📝 File Structure Summary**

droppy-deployment/

├── account-a/

│ ├── main.tf # Account A infrastructure

│ ├── variables.tf # Account A variables

│ ├── outputs.tf # Account A outputs

│ ├── terraform.tfvars # Account A values (create from example)

│ └── terraform.tfvars.example

├── account-b/

│ ├── main.tf # Account B infrastructure

│ ├── variables.tf # Account B variables

│ ├── outputs.tf # Account B outputs

│ ├── terraform.tfvars # Account B values (create from example)

│ └── terraform.tfvars.example

├── scripts/

│ ├── deploy.sh # Automated deployment

│ ├── cleanup.sh # Resource cleanup

│ └── generate-keypair.sh # SSH key generation

├── shared/

│ └── terraform.tf # Shared Terraform config

└── README.md # This guide

**🎯 Success Criteria**

Deployment is successful when:

* ✅ ECS service shows 2 running tasks
* ✅ ALB targets are healthy
* ✅ Jump host is accessible via RDP
* ✅ http://app.droppy.lan loads from jump host
* ✅ You can upload/download files through Droppy interface

**📞 Support**

If you encounter issues:

1. Check the troubleshooting section above
2. Review Terraform plan/apply output for errors
3. Check AWS CloudWatch logs for detailed error messages
4. Verify all prerequisites are met