🌾 Farm Financial RAG Application — Cloud Foundry on Azure Migration Guide

Note on platform naming: Azure doesn't offer a native service named "Azure Foundry." This guide assumes you are deploying a Cloud Foundry-compatible runtime (e.g., VMware Tanzu Application Service) hosted on Azure. Where appropriate, we also note Azure-native alternatives (Azure App Service / Azure Container Apps).

📋 Table of Contents

1. Pre-Migration Assessment

1.1 Current Application Analysis

- Application Type: Python FastAPI web application
- Database: SQLite (local file-based)
- External Dependencies: OpenAI API
- Static Files: HTML, CSS, JavaScript (migrate to Blob Storage/CDN)
- Port: 8000 (configurable)

1.2 Platform Compatibility Check

- ✅ Python Support: Cloud Foundry buildpacks support Python 3.8+ (use Python 3.11 where possible).
- ✅ FastAPI Support: Compatible; run behind Uvicorn/Gunicorn.
- ✅ Static Files: Serve from Azure Blob Storage (optionally fronted by Azure CDN).
- ⚠️ Database: Migrate from SQLite to Azure SQL Database (SQL Server).
- ✅ Secrets: Use Azure Key Vault (access via Managed Identity or service principal).

1.3 Prerequisites for Migration

- Azure subscription with appropriate permissions
- Cloud Foundry runtime on Azure (e.g., VMware Tanzu Application Service)
- Azure SQL Database instance
- Azure Key Vault for secrets management
- Azure Blob Storage for static files
- Domain name (optional, for custom URLs)

2. Cloud Foundry on Azure Setup

2.1 Create/Configure Cloud Foundry Runtime

- Access Azure Portal (portal.azure.com) and verify resource groups, VNets, and identity setup for the CF/Tanzu foundation.
- Confirm buildpacks (Python) and routing/health-check configuration (HTTP /health).
- Enable centralized logging/metrics export (e.g., to Azure Monitor / Application Insights).
- Configure autoscaling policies based on CPU/Memory/Queue depth as needed.

2.2 Create Supporting Azure Services

2.2.1 Azure SQL Database

1. Create SQL Database: Name: farm-rag-db; choose performance tier (Basic for dev, Standard for prod).
2. Create/Select SQL Server; set collation (SQL_Latin1_General_CP1_CI_AS).
3. Configure firewall: add admin IPs and enable “Allow Azure services to access server.”
4. Create a dedicated application user and grant least-privilege permissions.

2.2.2 Azure Key Vault

5. Create Key Vault: Name: farm-rag-vault-[suffix].
6. Configure access policies or RBAC: add your user and the app’s managed identity.
7. Grant “Get” and “List” for secrets.
8. Store secrets: OpenAI API key, DB connection string, other sensitive config.

2.2.3 Azure Blob Storage

9. Create Storage Account: Name: farmragstorage[suffix]; choose replication (LRS for dev, GRS for prod).
10. Create container: static-files; Access Level: Blob (anonymous read) or private + CDN as appropriate.
11. Upload static assets and set correct content types; consider enabling Azure CDN.

3. Application Preparation

3.1 Code Modifications Required

3.1.1 Database Migration (SQLite → Azure SQL)

- Replace SQLite driver with SQL Server driver (SQLAlchemy + pyodbc or pymssql).
- Update connection string to Azure SQL format; enable connection pooling and retry logic.
- Update DDL/DML for SQL Server compatibility; create schema creation scripts.

3.1.2 Environment & Static Files

- Replace local .env with Key Vault integration; add fallback to env vars for local dev.
- Move static files to Azure Blob Storage; optionally front with Azure CDN.
- Add health check endpoint (/health); implement graceful shutdown handlers.
- Add structured logging suitable for cloud aggregation.

3.1.3 Foundry-Specific Files

- manifest.yml — app name, buildpack, env vars, memory/disk, health checks.
- requirements.txt — pin Azure dependencies.
- Procfile — start command (e.g., web: uvicorn app.main:app --host 0.0.0.0 --port $PORT).
- runtime.txt — Python version (e.g., python-3.11.x).

3.2 Dependencies Update

3.2.1 New Dependencies

```
azure-identity>=1.15.0
azure-keyvault-secrets>=4.7.0
azure-storage-blob>=12.19.0
pyodbc>=5.0.0
sqlalchemy>=2.0.0
pymssql>=2.2.0
```

3.2.2 Updated Dependencies

```
fastapi>=0.104.0
uvicorn>=0.24.0
pandas>=2.3.0
openai>=1.0.0
python-dotenv>=1.0.0
```

4. Database Migration

4.1 Schema Migration

4.1.1 Create Migration Scripts

- Export SQLite schema; convert SQLite-specific syntax to SQL Server equivalents.

- Map data types; update constraints; add indexes where needed.

4.1.2 Data Migration

- Export data from SQLite to CSV; preserve encoding and integrity.
- Import into Azure SQL using Azure Data Factory or bulk insert scripts; rebuild FKs and validate integrity.

4.2 Connection Configuration

- Use proper Azure SQL connection strings; set timeouts.
- Configure connection pooling and retries in SQLAlchemy/driver.

5. Environment Configuration

5.1 Azure Key Vault Integration

5.1.1 Configure Managed Identity

12. Enable System-Assigned Managed Identity on the app/resource.
13. Grant the identity Key Vault 'Get' and 'List' permissions (or use RBAC).

5.1.2 Store Configuration Secrets

- openai-api-key → OpenAI API key value
- database-connection-string → Azure SQL connection string
- app-config → JSON configuration object (optional)

5.2 Application Configuration Updates

5.2.1 Environment Variables

```
# Azure-specific environment variables
AZURE_KEY_VAULT_URL=https://farm-rag-vault.vault.azure.net/
AZURE_CLIENT_ID=<managed-identity-or-app-client-id>
AZURE_TENANT_ID=<azure-tenant-id>
DATABASE_CONNECTION_STRING=<azure-sql-connection-string>
OPENAI_API_KEY=<openai-api-key>
STATIC_FILES_URL=https://farmragstorage.blob.core.windows.net/static-files/
```

5.2.2 Configuration Management

- Create a configuration class that fetches secrets from Key Vault and caches them.
- Fallback to environment variables for local development.
- Validate required configuration and fail fast with clear errors.

6. Deployment Process

6.1 Prepare Deployment Package

6.1.1 Create Foundry Manifest & Procfile

- manifest.yml: name, services, buildpack/runtime, env vars, memory/disk, health checks.
- Procfile: define web process (e.g., Uvicorn command).

6.1.2 Prepare Static Files

- Upload HTML/CSS/JS to Blob Storage; set content types and CORS if needed.
- Update application code to reference Blob/CDN URLs; provide local fallback for dev.

6.2 Deploy to Cloud Foundry on Azure

6.2.1 Initial Deployment

14. Install Azure CLI and log in.
15. Install CF/Tanzu CLI and target the foundation/space.
16. Push the app with manifest.yml (e.g., `cf push`).

6.2.2 Configure Services

6.2.2.1 Bind Database Service

- Create the DB service instance and bind it to the app.
- Restage/restart to inject VCAP_SERVICES (if applicable) and update connection config.

6.2.2.2 Configure Environment Variables

- Set app-specific variables; configure logging levels and performance parameters.

6.3 Post-Deployment Configuration

6.3.1 Domain and SSL

6.3.1.1 Configure Custom Domain

- Add custom domain; configure DNS and SSL certificate (Let's Encrypt/managed cert).

6.3.1.2 Update Application URLs

- Update CORS, redirect URLs, and static file base URLs.

6.3.2 Monitoring Setup

6.3.2.1 Enable Application Insights

- Enable monitoring, dashboards, and alerts.

6.3.2.2 Configure Logging

- Set up log aggregation, retention, and levels appropriate for prod.

7. Testing & Validation

7.1 Pre-Deployment Testing

7.1.1 Local Testing with Azure Services

- Verify Azure SQL connectivity and CRUD operations; validate data integrity.
- Verify Key Vault secret retrieval with managed identity/service principal.

7.1.2 Application Testing

- Unit tests for business logic and error handling; config loading tests.
- Integration tests for API endpoints and external services.

7.2 Post-Deployment Testing

7.2.1 Smoke Tests

- Verify /health, API endpoints, database queries, and web UI.

7.2.2 Performance Testing

- Load tests for concurrency and response times; stress tests for recovery behavior.

8. Monitoring & Maintenance

8.1 Application Monitoring

8.1.1 Azure Application Insights

- Enable telemetry, custom metrics, and dashboards.
- Configure alerts for performance, errors, and availability.

8.1.2 Log Management

- Centralize logs, set retention, and implement error/exception tracking with notifications.

8.2 Database Monitoring

8.2.1 Azure SQL Monitoring

- Monitor query performance and DTU/vCore utilization; set performance alerts.
- Configure automated backups and test restores (PITR).

8.3 Maintenance Procedures

8.3.1 Regular Maintenance

- DB index optimization, statistics updates, and old data cleanup.
- Regular security/dependency/feature updates for the application.

8.3.2 Disaster Recovery

- Backups for DB, code, and configuration; document and test recovery runbooks.

9. Troubleshooting

9.1 Common Issues

9.1.1 Database Connection Issues

- Timeouts: verify firewall rules, connection strings, and network access.
- Authentication failures: check managed identity, Key Vault permissions, and secret values.

9.1.2 Application Issues

- Startup failures: check env vars, dependencies, and app logs.
- Runtime errors: inspect logs, resource utilization, and external service connectivity.

9.2 Debugging Procedures

9.2.1 Log Analysis

- Review app logs (CF logs) and Azure service logs; analyze error patterns and metrics.

9.2.2 Performance Debugging

- Database: analyze slow queries, index usage, and resource pressure.
- Application: profile code hot paths; monitor memory and CPU.

10. Post-Migration Checklist

10.1 Functional Verification

- [ ] Application starts successfully
- [ ] All API endpoints respond correctly
- [ ] Database operations work properly
- [ ] Static files load correctly
- [ ] OpenAI integration functions
- [ ] Web interface is accessible
- [ ] All features work as expected

10.2 Performance Verification

- [ ] Response times meet requirements
- [ ] Application handles expected load
- [ ] Database performance is acceptable
- [ ] Memory usage is within limits
- [ ] CPU utilization is reasonable

- [ ] Network latency is acceptable

10.3 Security Verification

- [ ] All secrets are stored in Key Vault
- [ ] Database access is properly secured
- [ ] CORS settings are correct
- [ ] SSL certificates are valid
- [ ] Access controls are in place
- [ ] Audit logging is enabled

10.4 Monitoring Verification

- [ ] Application Insights is working
- [ ] Alerts are configured
- [ ] Logs are being collected
- [ ] Dashboards are set up
- [ ] Notifications are working
- [ ] Backup procedures are tested

10.5 Documentation Updates

- [ ] Update deployment documentation
- [ ] Create operational runbooks
- [ ] Update troubleshooting guides
- [ ] Document configuration changes
- [ ] Update user documentation
- [ ] Create maintenance procedures

Appendix A. Azure-Native Alternatives (If not using Cloud Foundry)

- Azure App Service: Deploy containerized or code-based FastAPI; use App Settings for env vars; integrate with Key Vault references.
- Azure Container Apps: Container-focused with revisions, Dapr, and scale-to-zero; integrate with Key Vault and Managed Identity.
- Adjust steps in Sections 2 and 6 accordingly (replace manifest.yml/CF push with Azure-native deployment commands).