Deployment Guide

Ape Framework - Deployment Guide

Introduction

This guide covers deploying the Ape Framework to production environments, including IIS on Windows Server and Azure App Service.

Pre-Deployment Checklist

Before deploying, ensure you have:

- [ ] Production database server ready
- [ ] Master encryption key generated
- [ ] Email provider configured (Azure and/or SMTP)
- [ ] SSL certificate for HTTPS
- [ ] Backup strategy planned
- [ ] Domain name configured

Building for Production

Publish Command

```
cd Ape
dotnet publish -c Release -o ./publish
```

This creates an optimized, production-ready build in the `./publish` folder.

What's Included

The publish folder contains: - Compiled application DLLs - Static files (wwwroot) - Configuration files - Runtime dependencies

What's NOT Included

- Source code
- Development dependencies
- User secrets
- Environment-specific configs

IIS Deployment (Windows Server)

Prerequisites

1. **Windows Server** with IIS installed
2. **.NET Hosting Bundle** - Download from Microsoft
3. **SQL Server** accessible from the server

Step 1: Install .NET Hosting Bundle

1. Download from: https://dotnet.microsoft.com/download/dotnet
2. Select your .NET version (10.0)
3. Download “Hosting Bundle” (not just Runtime)
4. Install on the server
5. Restart IIS: `iisreset`

Step 2: Create the Website

1. Open **IIS Manager**
2. Right-click **Sites** → **Add Website**
3. Configure:
    - Site name: `Ape` (or your choice)
    - Physical path: Path to published files
    - Binding: Choose IP, port, hostname
    - SSL: Configure HTTPS binding

Step 3: Configure Application Pool

1. Select your site’s Application Pool
2. Click **Advanced Settings**
3. Set:
    - **.NET CLR Version:** No Managed Code
    - **Start Mode:** AlwaysRunning (recommended)
    - **Idle Timeout:** 0 (for always-on)

Step 4: Set Environment Variables

Method 1: Application Pool (Recommended)

This keeps variables isolated to your application:

1. Select the Application Pool
2. Click **Advanced Settings**
3. Find **Environment Variables**
4. Add each variable:

```
DB_SERVER_ILLUSTRATE = your-db-server
DB_NAME_ILLUSTRATE = your-db-name
DB_USER_ILLUSTRATE = your-db-user
DB_PASSWORD_ILLUSTRATE = your-db-password
MASTER_CREDENTIAL_KEY_ILLUSTRATE = your-32-char-key
```

Method 2: System Environment Variables

1. Open **System Properties** → **Advanced** → **Environment Variables**
2. Add as System variables
3. Restart IIS: `iisreset`

Method 3: web.config

Add to web.config (less secure, visible in file):

```
<aspNetCore>
  <environmentVariables>
    <environmentVariable name="DB_SERVER_ILLUSTRATE" value="..." />
  </environmentVariables>
</aspNetCore>
```

Step 5: Configure Folder Permissions

Grant IIS_IUSRS write access to: - `/ProtectedFiles/` - For document uploads - `/wwwroot/Galleries/` - For image uploads

```
icacls "C:\inetpub\wwwroot\Ape\ProtectedFiles" /grant "IIS_IUSRS:(OI)(CI)M
icacls "C:\inetpub\wwwroot\Ape\wwwroot\Galleries" /grant "IIS_IUSRS:(OI)(C
```

Step 6: Configure HTTPS

1. Obtain SSL certificate (Let's Encrypt, commercial CA, etc.)
2. Import certificate to server
3. In IIS, edit site bindings
4. Add HTTPS binding with certificate

Step 7: Test Deployment

1. Browse to your site URL
2. Verify:
   - Home page loads
   - Can log in
   - Email test works
   - File upload works

Azure App Service Deployment

Step 1: Create Azure Resources

1. Log in to Azure Portal
2. Create **Resource Group** (if needed)
3. Create **App Service**:
   - Name: Choose unique name
   - Runtime: .NET 10
   - Region: Choose closest to users
   - Plan: Choose appropriate tier
4. Create **SQL Database** (if needed):
   - Server: Create new or use existing
   - Database: Create with appropriate tier
   - Configure firewall rules

Step 2: Configure Application Settings

In App Service → **Configuration** → **Application settings**:

Add these settings:

Name	Value
`DB_SERVER_ILLUSTRATE`	your-server.database.windows.net
`DB_NAME_ILLUSTRATE`	your-database-name
`DB_USER_ILLUSTRATE`	your-username
`DB_PASSWORD_ILLUSTRATE`	your-password
`MASTER_CREDENTIAL_KEY_ILLUSTRATE`	your-32-char-key

Click **Save**.

Step 3: Deploy Application

Option A: Visual Studio

1. Right-click project → **Publish**
2. Select **Azure** → **Azure App Service**
3. Select your App Service
4. Click **Publish**

Option B: GitHub Actions

Create `.github/workflows/deploy.yml`:

```yaml
name: Deploy to Azure

on:
  push:
    branches: [main]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Setup .NET
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: '10.0.x'

    - name: Build
      run: dotnet build --configuration Release

    - name: Publish
      run: dotnet publish -c Release -o ./publish

    - name: Deploy to Azure
      uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ secrets.AZURE_WEBAPP_NAME }}
        publish-profile: ${{ secrets.AZURE_PUBLISH_PROFILE }}
        package: ./publish
```

Option C: Azure CLI

```bash
# Build and publish
dotnet publish -c Release -o ./publish

# Deploy
az webapp deploy --resource-group MyResourceGroup \
                 --name MyAppName \
                 --src-path ./publish.zip
```

Step 4: Configure Storage

For persistent file storage, consider:

1. **Azure Blob Storage** for documents and images
2. **Azure Files** mounted to App Service
3. **Local storage** (lost on scale/restart)

Step 5: Configure Custom Domain

1. In App Service → **Custom domains**
2. Add custom domain
3. Verify ownership via DNS
4. Add SSL binding (free managed certificate available)

Database Migration

First Deployment

The application auto-migrates on startup. Alternatively:

```bash
# Generate migration script
dotnet ef migrations script -o migration.sql

# Apply to production database
sqlcmd -S server -d database -U user -P password -i migration.sql
```

Subsequent Updates

Option 1: Auto-migrate (default behavior) Option 2: Apply manually before deployment

```
dotnet ef database update --connection "Server=...;Database=...;..."
```

Post-Deployment Configuration

First-Time Setup

1. Navigate to your site
2. Log in with default admin (admin@admin.com / Admin123!)
3. **Immediately change the password**
4. Configure System Credentials:
   - SMTP settings
   - Azure Email settings (optional)
   - Site name
5. Configure Contact Form recipients
6. Test email functionality

Verify Functionality

- [ ] Home page loads
- [ ] User registration works
- [ ] Email verification sends
- [ ] Password reset works
- [ ] Document upload works
- [ ] Gallery upload works
- [ ] Admin functions accessible

Monitoring and Maintenance

Application Logging

Configure logging in `appsettings.Production.json`:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning",
      "Microsoft": "Warning"
    }
  }
}
```

Health Checks

Consider adding health check endpoint for monitoring.

Backup Strategy

Database: - SQL Server: Automated backups via Azure or SQL Server Agent - Frequency: Daily minimum, hourly for critical

Files: - `/ProtectedFiles/` - Document storage - `/wwwroot/Galleries/` - Image storage - Backup to separate storage/location

Encryption Key: - Store master key backup securely - Document recovery procedure

Updates

1. Test updates in staging environment first
2. Backup database before major updates

3. Apply during low-traffic periods
4. Have rollback plan ready

Troubleshooting

Common Issues

HTTP 500.30 - ASP.NET Core app failed to start

1. Check Application Event Log
2. Verify .NET Hosting Bundle installed
3. Check environment variables set correctly
4. Test database connection

HTTP 502.5 - Process failure

1. Verify .NET version compatibility
2. Check for missing dependencies
3. Review stdout log

Database connection errors

1. Verify connection string/environment variables
2. Check firewall rules
3. Test connection from server

Files not uploading

1. Check folder permissions
2. Verify path exists
3. Check file size limits in web.config

Enable Detailed Errors (Temporarily)

In web.config:

```xml
<aspNetCore stdoutLogEnabled="true" stdoutLogFile=".\logs\stdout" />
```

Create the `logs` folder and grant write permission.

Scaling Considerations

Horizontal Scaling

If scaling to multiple instances: - Use shared storage for uploads (Azure Blob, network share) - Use distributed cache for sessions - Configure database for concurrent access

Performance Optimization

- Enable response compression
- Configure output caching
- Use CDN for static files
- Optimize database queries

Security Hardening

Production Settings

1. Disable detailed errors
2. Enable HTTPS only
3. Configure HSTS headers
4. Set secure cookie policies
5. Review CORS settings

Firewall Rules

- Allow only necessary ports (80, 443)
- Restrict database access to app servers
- Use network security groups (Azure)

Regular Maintenance

- Keep .NET runtime updated
- Update NuGet packages
- Review security advisories
- Rotate credentials periodically

Version: 1.0.0 **Framework:** Ape Framework **Site:** https://Illustrate.net