Azure Migration Deployment Fixes Summary

Issue Overview

The Azure deployment was failing during the database migration step with the error:

```
"/usr/bin/python3: can't open file
'/home/site/wwwroot/scripts/migrate.py': [Errno 2] No such file or
directory"
```

Root Cause: The `scripts/migrate.py` file was not being found on the deployed Azure Web App, likely due to:

- Missing files in the deployment package
- Incorrect path references in the migration command
- Lack of validation during packaging and deployment

Applied Fixes

1. Enhanced Deployment Package Creation with Validation

Location: `.github/workflows/azure-deploy-saas.yml` - "Create deployment package" step

Changes Applied:

- Added `set -euxo pipefail` for strict error handling
- Added explicit existence checks for required directories (`app/`, `migrations/`, `scripts/`)
- Added verification that `scripts/migrate.py` file exists before packaging
- Added deployment package contents listing with `unzip -l deploy.zip`
- Added specific verification that `migrate.py` is included in the zip file

Benefits:

- CI will fail early if required files/directories are missing
- Clear visibility into what's actually being packaged and deployed
- Prevents deployment of incomplete packages

2. Post-Deployment File Verification

Location: `.github/workflows/azure-deploy-saas.yml` - New "Verify deployed files on Kudu" step

Changes Applied:

- Added step that runs after deployment but before migration
- Uses Kudu VFS API to list deployed files on Azure Web App
- Specifically checks for:

    - Root directory contents (`/site/wwwroot/`)
    - `scripts/` directory and its contents
    - Presence of `migrate.py` file
    - `app/` and `migrations/` directories

Benefits:

- Provides concrete evidence of what files actually exist on the deployed app
- Helps diagnose deployment issues before attempting migration
- Clear logging of file structure for troubleshooting

3. Improved Migration Command with Absolute Paths and Fallback

Location: `.github/workflows/azure-deploy-saas.yml` - "Run database migrations" step

Changes Applied:

- Changed migration command from relative path `scripts/migrate.py` to absolute path with fallback:

  ```
  $PYTHON_PATH /home/site/wwwroot/scripts/migrate.py || $PYTHON_PATH
  /home/site/wwwroot/app/scripts/migrate.py
  ```

- Uses absolute paths to avoid working directory ambiguity
- Includes fallback path in case the script ends up in `app/scripts/` instead of `scripts/`

Benefits:

- Eliminates path resolution issues
- Provides fallback if files are deployed to unexpected locations
- More robust migration execution

4. Enhanced Error Handling and Logging

Changes Applied:

- All critical steps now use `set -euo pipefail` for strict error handling
- Added detailed logging throughout the deployment process
- Clear success/error indicators (✅ /❌) for easy log scanning
- Comprehensive error messages with troubleshooting steps

Benefits:

- Faster identification of deployment issues
- Clear guidance for manual intervention when needed
- Better debugging information for future issues

Key Workflow Improvements

Before (Issues)

- Silent failures in package creation
- No visibility into deployed file structure
- Relative paths prone to working directory issues
- Limited error reporting

After (Improvements)

- ✅ Explicit validation of required files before packaging
- ✅ Complete deployment package contents listing
- ✅ Post-deployment file verification via Kudu API
- ✅ Absolute paths with fallback handling
- ✅ Comprehensive error reporting and troubleshooting guidance
- ✅ Early failure detection to prevent incomplete deployments

Testing the Fixes

Manual Testing Steps

1. **Verify the workflow changes are in place**:

```
git status
git add .github/workflows/azure-deploy-saas.yml
git commit -m "Fix Azure migration deployment issue with enhanced
validation and file verification"
git push origin main
```

2. **Monitor the GitHub Actions deployment**:
   - Go to your GitHub repository → Actions tab
   - Watch the "Deploy AI Event Planner SaaS to Azure" workflow
   - Pay attention to these new log sections:
     - "=== deploy.zip contents ===" in packaging step
     - "=== Verifying Deployed Files on Azure Web App ===" in verification step
     - Migration command execution with absolute paths

3. **What to expect in the logs**:

```
✅ Checking scripts/migrate.py in zip:
scripts/migrate.py

📁 Checking for scripts directory:
✅ scripts/ directory found. Contents:
✅ migrate.py found in scripts directory
```

```
🚀 Running database migration with Python: /usr/bin/python3
✅ Database migration completed successfully
```

Validation Checklist

- ☐ Package creation lists all required directories
- ☐ Deploy.zip contents show scripts/migrate.py is included
- ☐ File verification step confirms migrate.py exists on deployed app
- ☐ Migration runs successfully with absolute path
- ☐ No "file not found" errors in migration step

Troubleshooting Guide

If Package Creation Fails

- Check if `scripts/migrate.py` exists in your repository
- Verify no `.gitignore` rules are excluding the scripts directory
- Ensure all required directories (`app/`, `migrations/`, `scripts/`) exist

If File Verification Shows Missing Files

- Review the deployment zip contents in the logs
- Check if Azure Web App deployment completed successfully
- Verify the Azure Web Apps deployment action didn't encounter errors

If Migration Still Fails

- Check the file verification logs to confirm `migrate.py` is present
- Review the Python path detection logs
- Look for database connection or permission errors in migration output

Files Modified

1. **`.github/workflows/azure-deploy-saas.yml`** - Enhanced with all validation and verification steps

Next Steps

1. **Commit and deploy** the workflow changes
2. **Monitor the next deployment** for successful file verification and migration
3. **Review logs** to ensure all new validation steps pass
4. **Test the deployed application** to confirm migrations completed successfully

Additional Recommendations

For Future Robustness

1. Consider running migrations during the build phase instead of post-deployment

2. Add database connectivity tests before attempting migrations
3. Implement migration rollback procedures for failed deployments
4. Add health checks to verify application startup after migrations

Monitoring

1. Set up alerts for deployment failures
2. Monitor Azure App Service logs for runtime issues
3. Track migration execution time and success rates