Odin’s Eye - Verification & Testing Guide

Business: Viking Restaurant Consultants LLC
Application: Odin’s Eye (P&L Converter)
Version: 1.0.0

Table of Contents

Post-Deployment Verification

Quick Verification Checklist

Complete this checklist immediately after deployment:

- [ ] Application URL is accessible
- [ ] Landing page loads without errors
- [ ] No console errors in browser DevTools
- [ ] Database connection is working
- [ ] Environment variables are correctly set
- [ ] SSL/HTTPS is enabled
- [ ] Stripe integration loads
- [ ] User authentication works
- [ ] File upload functionality works

Step 1: Basic Connectivity

```
# Get your app URL
APP_NAME="odins-valhalla"  # or "odins-almanac"
RESOURCE_GROUP="viking-restaurant-rg"

APP_URL=$(az webapp show \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP \
  --query defaultHostName -o tsv)

echo "Application URL: https://$APP_URL"

# Test if the app is responding
curl -I https://$APP_URL
```

Expected Response:

```
HTTP/2 200
content-type: text/html; charset=utf-8
```

If you get 503 Service Unavailable:
- The app is still starting up (wait 2-3 minutes)
- Check logs: `az webapp log tail --name $APP_NAME --resource-group $RESOURCE_GROUP`

Step 2: Check Application Logs

```
# View real-time logs
az webapp log tail \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP

# Download logs for detailed analysis
az webapp log download \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP \
  --log-file odins-eye-logs.zip
```

What to Look For:
- ✅ "Server started on port..." message
- ✅ "Database connected successfully"
- ✅ No error stack traces
- ❌ Connection refused errors
- ❌ Module not found errors
- ❌ Environment variable missing warnings

Step 3: Verify Environment Variables

```
# List all configured environment variables
az webapp config appsettings list \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP \
  --query "[].{Name:name, Value:value}" \
  --output table
```

Required Variables:
- `STRIPE_PUBLISHABLE_KEY` (starts with `pk_`)
- `STRIPE_SECRET_KEY` (starts with `sk_`)
- `DATABASE_URL` (starts with `postgresql://`)
- `NODE_ENV` = `production`
- `WEBSITE_NODE_DEFAULT_VERSION` = `~20`

Fix Missing Variables:

```
az webapp config appsettings set \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP \
  --settings "VARIABLE_NAME=value"

# Restart after changes
az webapp restart \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP
```

Step 4: Test Database Connection

```
# SSH into the app
az webapp ssh \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP

# Once connected, test database connection
cd /home/site/wwwroot
node -e "const { Pool } = require('pg'); const pool = new Pool({ connectionString:
process.env.DATABASE_URL }); pool.query('SELECT NOW()', (err, res) => { if (err) con-
sole.error(err); else console.log('Database connected:', res.rows[0]);
pool.end(); });"
```

Expected Output:

```
Database connected: { now: 2025-10-19T04:00:00.000Z }
```

Automated Health Checks

Create a Health Check Script

Save this as `health-check.sh`:

```bash
#!/bin/bash

APP_URL="https://odins-valhalla.azurewebsites.net"
ERRORS=0

echo "===================================="
echo "Odin's Eye Health Check"
echo "===================================="
echo ""

# Test 1: HTTP Status
echo "[1/7] Testing HTTP status..."
STATUS=$(curl -s -o /dev/null -w "%{http_code}" $APP_URL)
if [ "$STATUS" = "200" ]; then
    echo "✓ HTTP Status: $STATUS (OK)"
else
    echo "✗ HTTP Status: $STATUS (FAILED)"
    ((ERRORS++))
fi

# Test 2: Response Time
echo "[2/7] Testing response time..."
RESPONSE_TIME=$(curl -s -o /dev/null -w "%{time_total}" $APP_URL)
if (( $(echo "$RESPONSE_TIME < 3.0" | bc -l) )); then
    echo "✓ Response Time: ${RESPONSE_TIME}s (OK)"
else
    echo "⚠ Response Time: ${RESPONSE_TIME}s (SLOW)"
fi

# Test 3: SSL Certificate
echo "[3/7] Testing SSL certificate..."
SSL_EXPIRY=$(echo | openssl s_client -servername ${APP_URL#https://} -connect ${APP_UR
L#https://}:443 2>/dev/null | openssl x509 -noout -enddate 2>/dev/null | cut -d= -f2)
if [ -n "$SSL_EXPIRY" ]; then
    echo "✓ SSL Certificate: Valid until $SSL_EXPIRY"
else
    echo "✗ SSL Certificate: FAILED"
    ((ERRORS++))
fi

# Test 4: Content Check
echo "[4/7] Testing page content..."
CONTENT=$(curl -s $APP_URL)
if echo "$CONTENT" | grep -q "Odin's Eye"; then
    echo "✓ Page Content: Contains expected content"
else
    echo "✗ Page Content: Missing expected content"
    ((ERRORS++))
fi

# Test 5: JavaScript Resources
echo "[5/7] Testing JavaScript resources..."
JS_STATUS=$(curl -s -o /dev/null -w "%{http_code}" $APP_URL/assets/index.js)
if [ "$JS_STATUS" = "200" ] || [ "$JS_STATUS" = "304" ]; then
    echo "✓ JavaScript: Loading (Status: $JS_STATUS)"
else
    echo "⚠ JavaScript: Status $JS_STATUS"
fi

# Test 6: API Endpoint
echo "[6/7] Testing API endpoint..."
API_STATUS=$(curl -s -o /dev/null -w "%{http_code}" $APP_URL/api/health 2>/dev/null)
```

```
if [ "$API_STATUS" = "200" ] || [ "$API_STATUS" = "404" ]; then
    echo "✓ API: Responding (Status: $API_STATUS)"
else
    echo "✗ API: Not responding (Status: $API_STATUS)"
    ((ERRORS++))
fi

# Test 7: Database Connection
echo "[7/7] Checking application logs for database..."
if az webapp log show --name odins-valhalla --resource-group viking-restaurant-rg 2>/
dev/null | grep -q "Database"; then
    echo "✓ Database: Connection logs present"
else
    echo "⚠ Database: Cannot verify from logs"
fi

echo ""
echo "===================================="
if [ $ERRORS -eq 0 ]; then
    echo "✓ All critical checks passed!"
    echo "===================================="
    exit 0
else
    echo "✗ $ERRORS critical check(s) failed"
    echo "===================================="
    exit 1
fi
```

Run the health check:

```
chmod +x health-check.sh
./health-check.sh
```

Manual Testing Procedures

Test 1: Landing Page

1. **Navigate to:** `https://your-app-url.azurewebsites.net`
2. **Expected:**
   - Page loads within 3 seconds
   - Viking/Norse theme visible
   - “Sign In” and “Get Started” buttons present
   - No console errors in browser DevTools (F12)
3. **Open DevTools Console (F12):**
   - Check for any red errors
   - Verify no 404 or 500 errors
   - Check Network tab for failed requests

Test 2: User Registration

1. **Click “Get Started” or “Sign Up”**

2. **Fill in registration form:**
   - Email: `test@example.com`
   - Password: `TestPassword123!`
3. **Submit form**
4. **Expected:**
   - Form submits without errors
   - User is redirected to dashboard or login page
   - Success message appears

Test 3: User Login

1. **Click "Sign In"**
2. **Enter credentials:**
   - Email: `test@example.com`
   - Password: `TestPassword123!`
3. **Submit form**
4. **Expected:**
   - Login successful
   - Redirected to dashboard
   - User session established

Test 4: Dashboard Access

1. **After logging in, verify:**
   - Dashboard loads
   - Navigation menu is accessible
   - User information displays correctly
   - All UI elements render properly

Test 5: File Upload

1. **Navigate to P&L upload page**
2. **Prepare test file:**
   - Use a sample Excel or CSV file
   - Include basic P&L data (revenue, expenses)
3. **Upload file:**
   - Click upload button
   - Select file
   - Submit
4. **Expected:**
   - File uploads successfully
   - Progress indicator shows
   - File is processed
   - Results display

Test 6: Data Export

1. **After uploading data:**
2. **Click export button**
3. **Select format (PDF/Excel)**
4. **Expected:**
   - Export generates successfully

  - File downloads
  - Data is formatted correctly

Test 7: Stripe Payment Flow (If Applicable)

1. **Navigate to subscription/payment page**
2. **Click "Subscribe" or "Upgrade"**
3. **Stripe checkout should load**
4. **Use Stripe test card:**
   - Card: `4242 4242 4242 4242`
   - Expiry: Any future date
   - CVC: Any 3 digits
   - ZIP: Any 5 digits
5. **Complete payment**
6. **Expected:**
   - Payment processes successfully
   - User subscription updates
   - Confirmation message appears

Performance Testing

Load Time Analysis

Using curl:

```
# Measure total time
curl -s -o /dev/null -w "\nTotal Time: %{time_total}s\n" https://your-app-url.azurewebsites.net

# Detailed timing
curl -s -o /dev/null -w "\nDNS Lookup: %{time_namelookup}s\nConnect: %{time_connect}s\nSSL: %{time_appconnect}s\nFirst Byte: %{time_starttransfer}s\nTotal: %{time_total}s\n" https://your-app-url.azurewebsites.net
```

Performance Benchmarks:
- ✅ **Excellent:** < 1 second
- ✅ **Good:** 1-2 seconds
- ⚠️ **Acceptable:** 2-3 seconds
- ❌ **Needs Improvement:** > 3 seconds

Concurrent Users Test

Using Apache Bench (ab):

```
# Install ab (if not already installed)
# Ubuntu: sudo apt-get install apache2-utils
# Mac: brew install httpd

# Test with 10 concurrent users, 100 requests
ab -n 100 -c 10 https://your-app-url.azurewebsites.net/

# Test with authentication (if needed)
ab -n 100 -c 10 -H "Cookie: session_id=xxx" https://your-app-url.azurewebsites.net/
```

Target Metrics:
- **Requests per second:** > 10
- **Time per request:** < 100ms (mean)
- **Failed requests:** 0

Memory and CPU Usage

```
# View metrics
az monitor metrics list \
  --resource odins-valhalla \
  --resource-group viking-restaurant-rg \
  --resource-type "Microsoft.Web/sites" \
  --metric "MemoryPercentage" \
  --start-time 2025-10-19T00:00:00Z \
  --end-time 2025-10-19T23:59:59Z \
  --interval PT1H

# CPU usage
az monitor metrics list \
  --resource odins-valhalla \
  --resource-group viking-restaurant-rg \
  --resource-type "Microsoft.Web/sites" \
  --metric "CpuPercentage" \
  --start-time 2025-10-19T00:00:00Z \
  --end-time 2025-10-19T23:59:59Z \
  --interval PT1H
```

Healthy Ranges:
- **CPU Usage:** < 60% (under normal load)
- **Memory Usage:** < 70%
- **Response Time:** < 500ms (95th percentile)

Security Verification

Security Checklist

- [ ] HTTPS enforced (no HTTP access)
- [ ] TLS 1.2 or higher
- [ ] Security headers present
- [ ] CORS properly configured
- [ ] Authentication working
- [ ] Session management secure
- [ ] Secrets not exposed in client-side code

- [ ] Database credentials not exposed
- [ ] No sensitive data in logs

Test HTTPS Enforcement

```
# Should redirect to HTTPS
curl -I http://your-app-url.azurewebsites.net

# Should be HTTPS
curl -I https://your-app-url.azurewebsites.net
```

Check Security Headers

```
curl -I https://your-app-url.azurewebsites.net | grep -E "Strict-Transport-Security|X-Frame-Options|X-Content-Type-Options|Content-Security-Policy"
```

Expected Headers:
- `Strict-Transport-Security: max-age=31536000`
- `X-Frame-Options: DENY` or `SAMEORIGIN`
- `X-Content-Type-Options: nosniff`

Test SSL/TLS Configuration

```
# Test SSL
openssl s_client -connect your-app-url.azurewebsites.net:443 -tls1_2

# Check certificate expiry
echo | openssl s_client -servername your-app-url.azurewebsites.net -connect your-app-url.azurewebsites.net:443 2>/dev/null | openssl x509 -noout -dates
```

Verify Environment Variables are Secure

```
# Check that secrets are not exposed in browser
curl -s https://your-app-url.azurewebsites.net | grep -i "sk_test\|sk_live\|DATABASE_URL"
# Should return nothing (no secrets in HTML)
```

Stripe Integration Testing

Test Mode Verification

1. **Verify using test keys:**
   ```
   bash
     az webapp config appsettings list \
       --name $APP_NAME \
       --resource-group $RESOURCE_GROUP \
       --query "[?name=='STRIPE_PUBLISHABLE_KEY'].value" -o tsv
   ```
   - Should start with `pk_test_` for testing
   - Should start with `pk_live_` for production

Test Card Numbers

Use these Stripe test cards:

Card Number	Brand	Behavior
4242 4242 4242 4242	Visa	Succeeds
4000 0000 0000 0002	Visa	Declined (generic)
4000 0000 0000 9995	Visa	Declined (insufficient funds)
4000 0027 6000 3184	Visa	Requires authentication (3D Secure)

Payment Flow Test

1. **Navigate to payment page**
2. **Initiate payment with test card:**
   - Card: `4242 4242 4242 4242`
   - Expiry: `12/34`
   - CVC: `123`
   - ZIP: `12345`
3. **Verify in Stripe Dashboard:**
   - Go to dashboard.stripe.com (https://dashboard.stripe.com)
   - Check **Payments** section
   - Verify test payment appears

Webhook Testing (If Configured)

```
# Install Stripe CLI
# Mac: brew install stripe/stripe-cli/stripe
# Linux: Download from https://github.com/stripe/stripe-cli/releases/latest

# Forward webhooks to local testing
stripe listen --forward-to https://your-app-url.azurewebsites.net/api/webhooks/stripe

# Trigger test events
stripe trigger payment_intent.succeeded
stripe trigger customer.subscription.created
```

Database Verification

Connection Test

```
# Direct PostgreSQL connection test
psql "postgresql://username:password@host:5432/database?sslmode=require" -c "SELECT
version();"
```

Verify Schema

```
# SSH into app
az webapp ssh --name $APP_NAME --resource-group $RESOURCE_GROUP

# Run schema check
cd /home/site/wwwroot
npm run db:push
```

Expected Output:
- No errors
- "Schema synced" or similar success message

Check Database Tables

```
# List tables
psql "$DATABASE_URL" -c "\dt"

# Check specific table
psql "$DATABASE_URL" -c "SELECT COUNT(*) FROM users;"
```

Expected Tables:
- `users`
- `sessions`
- `pls` (P&L statements)
- Other application-specific tables

Test Database Queries

```
# Test insert
psql "$DATABASE_URL" -c "INSERT INTO test_table (name) VALUES ('test');"

# Test select
psql "$DATABASE_URL" -c "SELECT * FROM test_table LIMIT 5;"

# Test update
psql "$DATABASE_URL" -c "UPDATE test_table SET name='updated' WHERE name='test';"

# Test delete
psql "$DATABASE_URL" -c "DELETE FROM test_table WHERE name='updated';"
```

Monitoring Setup

Enable Application Insights

```
# Create Application Insights
az monitor app-insights component create \
  --app odins-eye-insights \
  --location eastus \
  --resource-group viking-restaurant-rg \
  --application-type web

# Get instrumentation key
INSIGHTS_KEY=$(az monitor app-insights component show \
  --app odins-eye-insights \
  --resource-group viking-restaurant-rg \
  --query instrumentationKey -o tsv)

# Configure app
az webapp config appsettings set \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP \
  --settings "APPINSIGHTS_INSTRUMENTATIONKEY=$INSIGHTS_KEY"

# Restart app
az webapp restart --name $APP_NAME --resource-group $RESOURCE_GROUP
```

Set Up Alerts

```
# Alert for high CPU
az monitor metrics alert create \
  --name "High CPU Alert" \
  --resource-group viking-restaurant-rg \
  --scopes "/subscriptions/$SUBSCRIPTION_ID/resourceGroups/viking-restaurant-rg/pro-
viders/Microsoft.Web/sites/$APP_NAME" \
  --condition "avg Percentage CPU > 80" \
  --window-size 5m \
  --evaluation-frequency 1m

# Alert for high memory
az monitor metrics alert create \
  --name "High Memory Alert" \
  --resource-group viking-restaurant-rg \
  --scopes "/subscriptions/$SUBSCRIPTION_ID/resourceGroups/viking-restaurant-rg/pro-
viders/Microsoft.Web/sites/$APP_NAME" \
  --condition "avg MemoryPercentage > 85" \
  --window-size 5m \
  --evaluation-frequency 1m

# Alert for failed requests
az monitor metrics alert create \
  --name "Failed Requests Alert" \
  --resource-group viking-restaurant-rg \
  --scopes "/subscriptions/$SUBSCRIPTION_ID/resourceGroups/viking-restaurant-rg/pro-
viders/Microsoft.Web/sites/$APP_NAME" \
  --condition "total Http5xx > 10" \
  --window-size 5m \
  --evaluation-frequency 1m
```

Custom Monitoring Script

Save as `monitor-app.sh`:

```
#!/bin/bash

APP_URL="https://your-app-url.azurewebsites.net"
LOG_FILE="monitoring.log"

while true; do
    TIMESTAMP=$(date "+%Y-%m-%d %H:%M:%S")
    STATUS=$(curl -s -o /dev/null -w "%{http_code}" $APP_URL)
    RESPONSE_TIME=$(curl -s -o /dev/null -w "%{time_total}" $APP_URL)

    if [ "$STATUS" = "200" ]; then
        echo "[$TIMESTAMP] ✓ Status: $STATUS, Response: ${RESPONSE_TIME}s" | tee -a $L
OG_FILE
    else
        echo "[$TIMESTAMP] ✗ Status: $STATUS, Response: ${RESPONSE_TIME}s" | tee -a $L
OG_FILE
        # Send alert (email, Slack, etc.)
    fi

    sleep 60  # Check every minute
done
```

Common Issues Checklist

Application Won't Start

- [ ] Check environment variables are set correctly
- [ ] Verify DATABASE_URL format
- [ ] Check application logs for errors
- [ ] Ensure Node.js version is correct (20.x)
- [ ] Verify build completed successfully
- [ ] Check disk space (shouldn't be full)

Slow Performance

- [ ] Check App Service Plan tier (upgrade if on Free/Shared)
- [ ] Enable "Always On" setting
- [ ] Check database query performance
- [ ] Review application logs for bottlenecks
- [ ] Consider enabling CDN for static assets
- [ ] Check memory usage (upgrade if consistently high)

Database Connection Issues

- [ ] Verify DATABASE_URL is correct
- [ ] Check database server is running
- [ ] Verify firewall rules allow Azure connections
- [ ] Test connection from local machine
- [ ] Check SSL/TLS requirements (`sslmode=require`)

- [ ] Verify database credentials haven’t expired

Stripe Not Working

- [ ] Verify Stripe keys are set correctly
- [ ] Check keys match environment (test vs live)
- [ ] Verify keys are not expired
- [ ] Check browser console for Stripe.js errors
- [ ] Test with known-good test card
- [ ] Verify webhook endpoint (if configured)

SSL/HTTPS Issues

- [ ] Verify HTTPS-only is enabled
- [ ] Check certificate is valid and not expired
- [ ] Verify custom domain SSL binding (if using custom domain)
- [ ] Check for mixed content warnings in browser
- [ ] Verify all resources load over HTTPS

Verification Report Template

Use this template to document your verification:

```
Odin's Eye Deployment Verification Report
==========================================

Date: [DATE]
Deployed By: [NAME]
Environment: [Production/Staging/Development]
App URL: [URL]

Deployment Information:
- Azure Subscription ID: [ID]
- Resource Group: [NAME]
- App Service: [NAME]
- Region: [LOCATION]
- App Service Plan: [SKU]

Verification Results:
---------------------
[ ] Basic Connectivity: PASS/FAIL
[ ] Application Logs: PASS/FAIL
[ ] Environment Variables: PASS/FAIL
[ ] Database Connection: PASS/FAIL
[ ] User Authentication: PASS/FAIL
[ ] File Upload: PASS/FAIL
[ ] Stripe Integration: PASS/FAIL
[ ] Performance: PASS/FAIL
[ ] Security: PASS/FAIL

Issues Found:
-------------
[List any issues encountered]

Resolution:
-----------
[How issues were resolved]

Next Steps:
-----------
[ ] Enable Application Insights
[ ] Set up alerts
[ ] Configure custom domain
[ ] Enable CDN
[ ] Schedule regular backups
[ ] Document any custom configurations

Sign-off:
---------
Verified by: [NAME]
Date: [DATE]
Signature: [SIGNATURE]
```

Continuous Verification

Set up automated daily checks:

Create cron job (Linux/Mac):

```
# Edit crontab
crontab -e

# Add daily health check at 9 AM
0 9 * * * /path/to/health-check.sh
```

Create scheduled task (Windows PowerShell):

```
$action = New-ScheduledTaskAction -Execute 'PowerShell.exe' -Argument '-File C:
\path\to\health-check.ps1'
$trigger = New-ScheduledTaskTrigger -Daily -At 9am
Register-ScheduledTask -Action $action -Trigger $trigger -TaskName "Odins Eye Health
Check" -Description "Daily health check for Odin's Eye"
```

✅ **Congratulations!** You've completed the verification process. Your Odin's Eye application is now live and verified on Azure!

For ongoing monitoring and maintenance, refer to the DEPLOYMENT-GUIDE.md and set up regular automated health checks.