Amazon PPC Optimizer - Deployment Guide

Complete guide for deploying the Amazon PPC Optimizer to Google Cloud Functions with automatic token refresh.

📋 Table of Contents

Prerequisites

Required Software

- gcloud CLI (https://cloud.google.com/sdk/docs/install)
- Git
- Python 3.11+

Required Accounts

- Google Cloud account with billing enabled
- Amazon Advertising API access

API Credentials Setup

Amazon Advertising API

You need these credentials:

1. **Client ID**: `amzn1.application-oa2-client.xxxxx`
2. **Client Secret**: `amzn1.oa2-cs.v1.xxxxx`
3. **Refresh Token**: `Atzr|IwEBIxxxxx` (long-lived, doesn’t expire)
4. **Profile ID**: Your Amazon Ads profile ID

Important: The access token is automatically refreshed by the optimizer using the refresh_token. You only need to provide the refresh_token, not the access_token.

Obtaining Credentials

If you don’t have these credentials:

1. Visit Amazon Advertising API (https://advertising.amazon.com/API/docs/en-us/setting-up/overview)
2. Register your application
3. Complete OAuth authorization flow to get refresh_token

4. Get your Profile ID from Amazon Advertising Console

Google Cloud Setup

1. Create a Google Cloud Project

```
# Set your project ID
export PROJECT_ID="your-project-id"

# Create new project (or use existing)
gcloud projects create $PROJECT_ID

# Set as active project
gcloud config set project $PROJECT_ID
```

2. Enable Required APIs

```
# Enable Cloud Functions API
gcloud services enable cloudfunctions.googleapis.com

# Enable Cloud Build API
gcloud services enable cloudbuild.googleapis.com

# Enable Cloud Scheduler API (for scheduled runs)
gcloud services enable cloudscheduler.googleapis.com

# Enable Cloud Logging API
gcloud services enable logging.googleapis.com
```

3. Set Up Billing

Ensure billing is enabled for your project:

```
gcloud beta billing accounts list
gcloud beta billing projects link $PROJECT_ID --billing-account=BILLING_ACCOUNT_ID
```

Deployment Steps

Method 1: Deploy from Git Repository

```
# Clone the repository
git clone https://github.com/natureswaysoil/Amazom-PPC.git
cd Amazom-PPC

# Deploy to Cloud Functions
gcloud functions deploy amazon-ppc-optimizer \
  --gen2 \
  --runtime=python311 \
  --region=us-central1 \
  --source=. \
  --entry-point=run_optimizer \
  --trigger-http \
  --allow-unauthenticated \
  --timeout=540s \
  --memory=512MB \
  --min-instances=0 \
  --max-instances=1 \
  --set-env-vars AMAZON_CLIENT_ID="amzn1.application-oa2-client.
5f71a2504cb34903be357c736c290a30",AMAZON_CLIENT_SECRET="amzn1.oa2-
cs.v1.a1a0e3a3cf314be2eb5269334bd4401a18762fd702e2b100a4f61697a674f3af",AMAZON_REFRESH
_TOKEN="Atzr|IwEBIBGvUBJYDy4z4OZJEU68Oqr2eNOrkyOm-
WyHjFcEW4C_lmmoKmqvy9wafePmmmDZJuMAvsQHDwt41G1vV3_C_0-9QtLxtMHDxQz46XtcnQvIJBY3HQOu9j2
Z25NCO8gDcSJ88eAgNcno_GM97qDF6meQZWULUtSqDHVq7TgP00BHxeu3A6ibHRGFWCCe5vXq7w-CW4PI-
OB68wJJpXZwkb66P52hwfGPL4vDXuwm97mBxaNBCWGwrWBeAnoKismuP1yF9hqV3fVrwN16VKh-ddF1UpUec-
u5uGkz-
sqxLJffmG2H-71_MMr89CAAlVwouWF2AbvPPxJloXc1Nen8t_pCWZB2vyGB7gki14_unEeoKlGofeXuj6jYYPs
32RnPLLa6UwopjlNz-xk83r50sLUCrhJFkKfONmS6FnjFZ84GDa0O7vkSeOTEJRp7PeJNFnlznGI18vmo-
naH4REVqythHuwKwjbGUqc1j-ebGqslIv300PECZH3Ox54hQ4-EuQ4GYxMwpylwOV4LM77k1vRN3z54"
```

Note: Replace the environment variable values with your actual credentials.

Method 2: Deploy with Configuration File

```
# Create a config file with your settings
cat > config.json <<EOF
{
  "amazon_api": {
    "client_id": "YOUR_CLIENT_ID",
    "client_secret": "YOUR_CLIENT_SECRET",
    "refresh_token": "YOUR_REFRESH_TOKEN",
    "profile_id": "YOUR_PROFILE_ID",
    "region": "NA"
  },
  "optimization_rules": { ... }
}
EOF

# Deploy with config as environment variable
export PPC_CONFIG=$(cat config.json)

gcloud functions deploy amazon-ppc-optimizer \
  --gen2 \
  --runtime=python311 \
  --region=us-central1 \
  --source=. \
  --entry-point=run_optimizer \
  --trigger-http \
  --allow-unauthenticated \
  --timeout=540s \
  --memory=512MB \
  --set-env-vars PPC_CONFIG="$PPC_CONFIG"
```

Deployment Parameters Explained

- `--gen2` : Use Cloud Functions 2nd generation
- `--runtime=python311` : Python 3.11 runtime
- `--region=us-central1` : Deployment region (change as needed)
- `--entry-point=run_optimizer` : Function name to call
- `--trigger-http` : HTTP trigger (for Cloud Scheduler)
- `--timeout=540s` : 9-minute timeout (optimizer may take several minutes)
- `--memory=512MB` : Allocated memory
- `--min-instances=0` : Scale to zero when not in use
- `--max-instances=1` : Only one concurrent execution

Environment Variables

Required Environment Variables

The optimizer requires these environment variables (set during deployment):

```
AMAZON_CLIENT_ID=amzn1.application-oa2-client.xxxxx
AMAZON_CLIENT_SECRET=amzn1.oa2-cs.v1.xxxxx
AMAZON_REFRESH_TOKEN=Atzr|IwEBIxxxxx
```

OR a single combined variable:

```
PPC_CONFIG='{"amazon_api": {...}, "optimization_rules": {...}}'
```

Updating Environment Variables

To update environment variables after deployment:

```
gcloud functions deploy amazon-ppc-optimizer \
  --update-env-vars AMAZON_REFRESH_TOKEN="new_refresh_token"
```

Scheduling

Set Up Cloud Scheduler

Run the optimizer automatically on a schedule:

```
# Create a Cloud Scheduler job (runs daily at 3 AM)
gcloud scheduler jobs create http amazon-ppc-optimizer-daily \
  --location=us-central1 \
  --schedule="0 3 * * *" \
  --uri="https://us-central1-YOUR-PROJECT.cloudfunctions.net/amazon-ppc-optimizer" \
  --http-method=GET \
  --time-zone="America/New_York"

# For dry-run mode (testing without changes)
gcloud scheduler jobs create http amazon-ppc-optimizer-dryrun \
  --location=us-central1 \
  --schedule="0 */4 * * *" \
  --uri="https://us-central1-YOUR-PROJECT.cloudfunctions.net/amazon-ppc-optimizer?
dry_run=true" \
  --http-method=GET \
  --time-zone="America/New_York"
```

Schedule Examples

- Daily at 3 AM: `"0 3 * * *"`
- Every 6 hours: `"0 */6 * * *"`
- Twice daily (9 AM, 9 PM): `"0 9,21 * * *"`
- Weekdays only at noon: `"0 12 * * 1-5"`

Manually Trigger

```
# Trigger the function manually
gcloud scheduler jobs run amazon-ppc-optimizer-daily --location=us-central1

# Or via curl
curl "https://us-central1-YOUR-PROJECT.cloudfunctions.net/amazon-ppc-optimizer"
```

Monitoring

View Logs

```
# View recent logs
gcloud functions logs read amazon-ppc-optimizer --limit=50

# Follow logs in real-time
gcloud functions logs read amazon-ppc-optimizer --follow

# View logs in Cloud Console
gcloud functions describe amazon-ppc-optimizer --gen2 --region=us-central1
```

Key Log Messages to Monitor

- ✅ "Successfully authenticated with Amazon Ads API"
- ✅ "Optimization completed successfully"
- ✅ "Dashboard updated successfully"
- ❌ "Authentication failed"
- ❌ "Optimization failed"

Dashboard Monitoring

Check the dashboard for results:
https://ppc-dashboard.abacusai.app

Troubleshooting

Authentication Errors

Error: "Authentication failed"

Solutions:
1. Verify refresh_token is correct
2. Check client_id and client_secret
3. Ensure Amazon Ads API access is still active
4. Token may have been revoked - regenerate in Amazon console

Timeout Errors

Error: "Function timeout exceeded"

Solutions:
1. Increase timeout: `--timeout=900s` (15 minutes max)
2. Optimize lookback_days in config
3. Reduce number of enabled features

Memory Errors

Error: "Exceeded memory limit"

Solutions:
1. Increase memory: `--memory=1GB`
2. Reduce lookback_days
3. Process fewer campaigns per run

Rate Limit Errors

Error: "Too many requests"

Solutions:
- The optimizer has built-in rate limiting
- Amazon API: 10 requests/second (handled automatically)
- If errors persist, reduce frequency of scheduled runs

Deployment Errors

Error: "Build failed"

Solutions:
1. Check requirements.txt syntax
2. Verify all files are included (.gcloudignore)
3. Ensure Python 3.11 compatibility

Token Not Refreshing

Issue: "Access token expired" errors

Check:
1. Verify refresh_token is set correctly
2. Check logs for token refresh attempts
3. The optimizer calls `_refresh_auth_if_needed()` before each API call

Testing

Test Locally

```
# Install dependencies
pip install -r requirements.txt

# Set environment variables
export AMAZON_CLIENT_ID="your_client_id"
export AMAZON_CLIENT_SECRET="your_client_secret"
export AMAZON_REFRESH_TOKEN="your_refresh_token"

# Run locally
python main.py
```

Test on Cloud (Dry Run)

```
# Dry run - no changes will be made
curl "https://YOUR-FUNCTION-URL?dry_run=true"
```

Verify Token Refresh

Check logs for these messages:

```
"Successfully authenticated with Amazon Ads API"
"Access token expired, refreshing..."
```

Cost Optimization

Minimize Costs

1. **Use min-instances=0**: Scale to zero when not running
2. **Optimize memory**: Start with 512MB, increase only if needed
3. **Scheduled execution**: Run only when needed (e.g., once or twice daily)
4. **Timeout optimization**: Set appropriate timeout to avoid long-running failures

Estimated Costs

- **Cloud Functions**: ~$0.01 per run (512MB, 3-5 min execution)
- **Cloud Scheduler**: $0.10/month for one job
- **Total**: ~$0.50/month for daily runs

Security Best Practices

1. **Never commit credentials** to Git
2. **Use Secret Manager** for production (optional enhancement)
3. **Rotate tokens** regularly
4. **Monitor logs** for unauthorized access attempts
5. **Restrict function access** (remove `--allow-unauthenticated` and use IAM)

Next Steps

After successful deployment:

1. ✅ Test with dry-run mode
2. ✅ Monitor logs for first few runs
3. ✅ Check dashboard for results
4. ✅ Fine-tune optimization rules based on performance
5. ✅ Set up alerting (Cloud Monitoring)

Support

For issues or questions:
- Check Cloud Function logs first
- Review error messages in this guide
- Contact: james@natureswaysoil.com

Last Updated: October 13, 2025
Version: 2.0.0