AfricanMarket External Services Configuration Guide

Overview

This document provides comprehensive setup instructions for all external services required by the AfricanMarket application. Each service includes step-by-step configuration, security considerations, and troubleshooting information.

Required Services

1. Database - PostgreSQL (Required)

Option A: AWS RDS PostgreSQL (Recommended)

Setup Instructions:

1. **Create RDS Instance**

```
bash
   # Using AWS CLI
   aws rds create-db-instance \
       --db-instance-identifier africanmarket-prod \
       --db-instance-class db.t3.medium \
       --engine postgres \
       --engine-version 15.4 \
       --allocated-storage 100 \
       --storage-type gp2 \
       --storage-encrypted \
       --master-username africanmarket_admin \
       --master-user-password <secure-password> \
       --vpc-security-group-ids sg-xxxxxxxxx \
       --db-subnet-group-name africanmarket-subnet-group \
       --backup-retention-period 7 \
       --multi-az \
       --auto-minor-version-upgrade
```

1. **Configure Security Groups**

```
bash
   # Allow PostgreSQL access from application servers
   aws ec2 authorize-security-group-ingress \
       --group-id sg-xxxxxxxxx \
       --protocol tcp \
       --port 5432 \
       --source-group sg-yyyyyyyyy
```

2. **Environment Configuration**

```
bash
   DATABASE_URL=postgresql://africanmarket_admin:password@africanmarket-prod.xxxxxxxxx.us-east-1.rds.amazonaws.com:5432/africanmarket
```

Option B: Google Cloud SQL PostgreSQL

Setup Instructions:

1. **Create Cloud SQL Instance**

```
bash
  gcloud sql instances create africanmarket-prod \
      --database-version=POSTGRES_15 \
      --tier=db-standard-2 \
      --region=us-central1 \
      --storage-type=SSD \
      --storage-size=100GB \
      --storage-auto-increase \
      --backup-start-time=02:00 \
      --enable-bin-log \
      --maintenance-window-day=SUN \
      --maintenance-window-hour=3
```

1. **Create Database and User**

```
```bash
# Create database
gcloud sql databases create africanmarket –instance=africanmarket-prod

# Create user
gcloud sql users create africanmarket_user \
–instance=africanmarket-prod \
–password=
```
```

1. **Configure Connection**

```
bash
  DATABASE_URL=postgresql://africanmarket_user:password@<cloud-sql-ip>:5432/africanmarket
```

Security Configuration

```
-- Create read-only user for monitoring
CREATE USER monitor_user WITH PASSWORD 'monitoring_password';
GRANT CONNECT ON DATABASE africanmarket TO monitor_user;
GRANT USAGE ON SCHEMA public TO monitor_user;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO monitor_user;

-- Enable row-level security for sensitive tables
ALTER TABLE users ENABLE ROW LEVEL SECURITY;
ALTER TABLE payment_methods ENABLE ROW LEVEL SECURITY;
```

2. Cache - Redis (Required)

Option A: AWS ElastiCache Redis (Recommended)

Setup Instructions:

1. **Create Redis Cluster**

```
bash
  aws elasticache create-cache-cluster \
      --cache-cluster-id africanmarket-redis \
      --cache-node-type cache.t3.micro \
      --engine redis \
```

```
      --num-cache-nodes 1 \
      --cache-parameter-group-name default.redis7 \
      --cache-subnet-group-name africanmarket-cache-subnet \
      --security-group-ids sg-xxxxxxxxx \
      --at-rest-encryption-enabled \
      --transit-encryption-enabled
```

1. **Environment Configuration**
   ```
   bash
     REDIS_URL=rediss://africanmarket-redis.xxxxxx.cache.amazonaws.com:6379
   ```

Option B: DigitalOcean Managed Redis

Setup Instructions:

1. **Create via Dashboard**

- Go to DigitalOcean Control Panel
- Navigate to Databases → Create Database
- Select Redis, choose configuration
- Configure trusted sources

1. **Environment Configuration**
   ```
   bash
     REDIS_URL=rediss://default:password@private-db-redis-fra1-xxxxx.b.db.ondigitalocean.com:25061
   ```

Redis Security Configuration

```
# Enable authentication
CONFIG SET requirepass "your-redis-password"

# Disable dangerous commands
CONFIG SET rename-command FLUSHDB ""
CONFIG SET rename-command FLUSHALL ""
CONFIG SET rename-command DEBUG ""
```

3. File Storage - Cloudinary (Required)

Setup Instructions

1. **Create Cloudinary Account**
   - Visit Cloudinary.com (https://cloudinary.com)
   - Sign up for account
   - Verify email address

2. **Get API Credentials**
   - Navigate to Dashboard
   - Copy Cloud Name, API Key, and API Secret
   - Note these for environment configuration

3. **Create Upload Presets**
   ```
   javascript
     // Upload preset configuration for production
     {
       "name": "africanmarket_production",
       "unsigned": false,
   ```

```
    "folder": "africanmarket",
    "resource_type": "auto",
    "allowed_formats": ["jpg", "jpeg", "png", "gif", "webp"],
    "transformation": [
      {"quality": "auto:good"},
      {"fetch_format": "auto"}
    ],
    "eager": [
      {"width": 300, "height": 300, "crop": "fill"},
      {"width": 600, "height": 400, "crop": "fill"}
    ]
  }
```

4. **Environment Configuration**
   ```
   bash
   CLOUDINARY_CLOUD_NAME=your_cloud_name
   CLOUDINARY_API_KEY=123456789012345
   CLOUDINARY_API_SECRET=your_api_secret
   CLOUDINARY_UPLOAD_PRESET=africanmarket_production
   ```

Security Configuration

```
// Configure Cloudinary with security options
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
  secure: true,
  // Add security headers
  sign_url: true,
  auth_token: {
    key: "your_auth_token_key",
    duration: 3600 // 1 hour
  }
});
```

4. Payment Processing - Stripe (Required)

Setup Instructions

1. **Create Stripe Account**
   - Visit Stripe.com (https://stripe.com)
   - Complete business verification
   - Activate live payments

2. **Configure Webhooks**
   ```bash
   # Webhook endpoint URL
   https://your-domain.com/api/webhooks/stripe

# Required webhook events
- payment_intent.succeeded
- payment_intent.payment_failed
- invoice.payment_succeeded
- customer.subscription.updated
```

- account.updated
- transfer.created
```

1. **Get API Keys**
   - Navigate to Developers → API Keys
   - Copy Publishable Key and Secret Key
   - Create webhook signing secret

2. **Environment Configuration**
   bash
   ```
   STRIPE_SECRET_KEY=your_stripe_secret_key_here
   STRIPE_PUBLISHABLE_KEY=your_stripe_publishable_key_here
   STRIPE_WEBHOOK_SECRET=your_stripe_webhook_secret_here
   STRIPE_CONNECT_CLIENT_ID=your_stripe_connect_client_id_here
   ```

### Stripe Connect Setup (for vendor payouts)

```
// Express account creation for vendors
const account = await stripe.accounts.create({
  type: 'express',
  country: 'US', // or vendor's country
  email: vendor.email,
  capabilities: {
    card_payments: {requested: true},
    transfers: {requested: true},
  },
  business_type: 'individual',
  business_profile: {
    url: `https://africanmarket.com/vendor/${vendor.id}`,
    mcc: '5812', // Eating establishments
  },
});
```

Security Configuration

```
// Verify webhook signatures
const endpointSecret = process.env.STRIPE_WEBHOOK_SECRET;
const sig = request.headers['stripe-signature'];

try {
  const event = stripe.webhooks.constructEvent(request.body, sig, endpointSecret);
  // Process the event
} catch (err) {
  console.log(`⚠️  Webhook signature verification failed.`, err.message);
  return response.status(400).send(`Webhook Error: ${err.message}`);
}
```

5. Email Service - SendGrid (Required)

Setup Instructions

1. **Create SendGrid Account**
   - Visit SendGrid.com (https://sendgrid.com)
   - Complete account verification
   - Add payment method for production
```

2. **Domain Authentication**
   bash
   ```
   # Add DNS records for domain authentication
   CNAME: s1._domainkey.your-domain.com → s1.domainkey.uXXXXXXX.wlXXX.sendgrid.net
   CNAME: s2._domainkey.your-domain.com → s2.domainkey.uXXXXXXX.wlXXX.sendgrid.net
   CNAME: em1234.your-domain.com → u1234567.wl123.sendgrid.net
   ```

3. **Create API Key**
   - Navigate to Settings → API Keys
   - Create key with Mail Send permissions
   - Restrict key to specific IP addresses

4. **Configure Sender Identity**
   bash
   ```
   # Verify sender email
   FROM_EMAIL=noreply@your-domain.com
   SUPPORT_EMAIL=support@your-domain.com
   ```

5. **Environment Configuration**
   bash
   ```
   SENDGRID_API_KEY=SG.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   FROM_EMAIL=noreply@your-domain.com
   SUPPORT_EMAIL=support@your-domain.com
   ```

Email Templates Setup

```
// Create dynamic templates in SendGrid
const templates = {
  welcome: 'd-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  orderConfirmation: 'd-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  passwordReset: 'd-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  vendorWelcome: 'd-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
};
```

6. SMS Service - Twilio (Optional)

Setup Instructions

1. **Create Twilio Account**
   - Visit Twilio.com (https://twilio.com)
   - Complete phone verification
   - Add payment method

2. **Purchase Phone Number**
   ```bash
   # Search for available numbers
   curl -X GET "https://api.twilio.com/2010-04-01/Accounts/$TWILIO_ACCOUNT_SID/
   AvailablePhoneNumbers/US/Local.json" \
   -u $TWILIO_ACCOUNT_SID:$TWILIO_AUTH_TOKEN

# Purchase number
curl -X POST "https://api.twilio.com/2010-04-01/Accounts/$TWILIO_ACCOUNT_SID/
IncomingPhoneNumbers.json" \
–data-urlencode "PhoneNumber=+15551234567" \
   ```

-u $TWILIO_ACCOUNT_SID:$TWILIO_AUTH_TOKEN
```

1. **Environment Configuration**
   `bash`
   `TWILIO_ACCOUNT_SID=ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
   `TWILIO_AUTH_TOKEN=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
   `TWILIO_PHONE_NUMBER=+15551234567`

## 7. Maps Service - Google Maps (Required)

### Setup Instructions

1. **Create Google Cloud Project**
   ```bash
   # Create project
   gcloud projects create africanmarket-maps

# Enable APIs
gcloud services enable maps-backend.googleapis.com
gcloud services enable places-backend.googleapis.com
gcloud services enable geocoding-backend.googleapis.com
```

1. **Create API Key**
   ```bash
   # Create API key
   gcloud alpha services api-keys create –display-name=”AfricanMarket Maps API Key”

# Restrict API key
gcloud alpha services api-keys update \
–api-target=service=maps-backend.googleapis.com \
–api-target=service=places-backend.googleapis.com \
–api-target=service=geocoding-backend.googleapis.com
```

1. **Environment Configuration**
   `bash`
   `GOOGLE_MAPS_API_KEY=AIzaSyxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
```

Security Configuration

```
// Restrict API key usage
const restrictedKey = {
  restrictions: {
    browserKeyRestrictions: {
      allowedReferrers: [
        "https://your-domain.com/*",
        "https://www.your-domain.com/*"
      ]
    },
    serverKeyRestrictions: {
      allowedIps: [
        "YOUR_SERVER_IP"
      ]
    },
    androidKeyRestrictions: {
      allowedApplications: [
        {
          packageName: "com.africanmarket.app",
          sha1Fingerprint: "SHA1_FINGERPRINT"
        }
      ]
    },
    iosKeyRestrictions: {
      allowedBundleIds: [
        "com.africanmarket.app"
      ]
    }
  }
};
```

8. Error Monitoring - Sentry (Recommended)

Setup Instructions

1. **Create Sentry Account**
   - Visit Sentry.io (https://sentry.io)
   - Create organization and project
   - Choose JavaScript/Next.js platform

2. **Configure Project**
   ```bash
   # Install Sentry CLI
   npm install -g @sentry/cli

# Login to Sentry
sentry-cli login

# Create project configuration
sentry-cli projects create africanmarket
```

1. **Environment Configuration**
   `bash`
   `SENTRY_DSN=https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx@oXXXXXX.ingest.sentry.io/XXXXXXX`
   `SENTRY_ORG=your-org-name`
   `SENTRY_PROJECT=africanmarket`

Performance Monitoring Setup

```
// Configure performance monitoring
Sentry.init({
  dsn: process.env.SENTRY_DSN,
  environment: process.env.NODE_ENV,
  tracesSampleRate: 0.1,
  profilesSampleRate: 0.1,
  beforeSend(event) {
    // Filter out sensitive data
    if (event.exception) {
      event.exception.values.forEach(exception => {
        if (exception.stacktrace) {
          exception.stacktrace.frames.forEach(frame => {
            delete frame.vars;
          });
        }
      });
    }
    return event;
  }
});
```

9. Analytics - Google Analytics (Recommended)

Setup Instructions

1. **Create Google Analytics Account**
   - Visit Google Analytics (https://analytics.google.com)
   - Create account and property
   - Choose GA4 (Google Analytics 4)

2. **Configure Data Streams**
   `bash`
   `# Web stream configuration`
   `Website URL: https://your-domain.com`
   `Stream name: AfricanMarket Web`
   `Enhanced measurement: Enable all`

3. **Environment Configuration**
   `bash`
   `GOOGLE_ANALYTICS_ID=G-XXXXXXXXXX`

Enhanced E-commerce Setup

```
// Configure enhanced e-commerce tracking
gtag('config', 'G-XXXXXXXXXX', {
  custom_map: {
    'custom_parameter_1': 'user_type',
    'custom_parameter_2': 'vendor_category'
  }
});

// Track purchase events
gtag('event', 'purchase', {
  transaction_id: order.id,
  value: order.total,
  currency: 'USD',
  items: order.items.map(item => ({
    item_id: item.id,
    item_name: item.name,
    category: item.category,
    quantity: item.quantity,
    price: item.price
  }))
});
```

10. Push Notifications - Web Push (Optional)

Setup Instructions

1. **Generate VAPID Keys**
   ```bash
   # Install web-push library
   npm install -g web-push

# Generate VAPID keys
web-push generate-vapid-keys –json
```

1. **Environment Configuration**
   ```
   bash
   VAPID_PUBLIC_KEY=BGxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   VAPID_PRIVATE_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   VAPID_EMAIL=admin@your-domain.com
   ```

2. **Service Worker Setup**
   ```
   javascript
   // Register service worker
   if ('serviceWorker' in navigator && 'PushManager' in window) {
     navigator.serviceWorker.register('/sw.js')
       .then(registration => {
         console.log('SW registered: ', registration);
         return registration.pushManager.subscribe({
           userVisibleOnly: true,
           applicationServerKey: urlBase64ToUint8Array(VAPID_PUBLIC_KEY)
         });
       })
       .then(subscription => {
   ```

```
        console.log('User is subscribed:', subscription);
      });
  }
```

Service Integration Testing

Automated Testing Script

```bash
#!/bin/bash
# File: scripts/test-external-services.sh

echo "Testing external service integrations..."

# Test database connection
echo "Testing database connection..."
if psql $DATABASE_URL -c "SELECT version();" > /dev/null 2>&1; then
    echo "✅ Database connection successful"
else
    echo "❌ Database connection failed"
fi

# Test Redis connection
echo "Testing Redis connection..."
if redis-cli -u $REDIS_URL ping | grep -q PONG; then
    echo "✅ Redis connection successful"
else
    echo "❌ Redis connection failed"
fi

# Test Cloudinary
echo "Testing Cloudinary..."
if curl -f "https://api.cloudinary.com/v1_1/${CLOUDINARY_CLOUD_NAME}/image/upload" > /
dev/null 2>&1; then
    echo "✅ Cloudinary accessible"
else
    echo "❌ Cloudinary not accessible"
fi

# Test Stripe API
echo "Testing Stripe API..."
if curl -f -H "Authorization: Bearer ${STRIPE_SECRET_KEY}" "https://api.stripe.com/v1/
balance" > /dev/null 2>&1; then
    echo "✅ Stripe API accessible"
else
    echo "❌ Stripe API not accessible"
fi

# Test SendGrid API
echo "Testing SendGrid API..."
if curl -f -H "Authorization: Bearer ${SENDGRID_API_KEY}" "https://api.sendgrid.com/v3/
user/profile" > /dev/null 2>&1; then
    echo "✅ SendGrid API accessible"
else
    echo "❌ SendGrid API not accessible"
fi

echo "External service testing completed."
```

Cost Optimization

Service Cost Estimates (Monthly)

- **Database (AWS RDS t3.medium)**: $50-80
- **Redis (AWS ElastiCache t3.micro)**: $15-25
- **Cloudinary (Free tier + usage)**: $0-50
- **Stripe (2.9% + $0.30 per transaction)**: Variable
- **SendGrid (40,000 emails free tier)**: $0-15
- **Google Maps (Pay-as-you-go)**: $10-50
- **Sentry (Developer plan)**: $26
- **Total Estimated Monthly Cost**: $100-300

Cost Optimization Strategies

1. **Use managed service free tiers**
2. **Implement caching to reduce API calls**
3. **Optimize database queries to reduce compute**
4. **Use CDN to reduce bandwidth costs**
5. **Monitor usage patterns and scale accordingly**

Security Best Practices

API Key Management

```
# Use environment variables for all secrets
# Never commit API keys to version control
# Rotate API keys regularly
# Use least privilege access principles
# Monitor API key usage and set up alerts
```

Network Security

```
# Use VPC/private networks where possible
# Implement IP whitelisting
# Use SSL/TLS for all communications
# Enable encryption at rest and in transit
# Regular security audits and penetration testing
```

Compliance Considerations

- **PCI DSS**: For payment processing
- **GDPR**: For EU user data
- **CCPA**: For California user data
- **SOC 2**: For overall security practices

This external services guide should be updated whenever new services are added or existing service configurations change.