Analytics Service Integration Guide

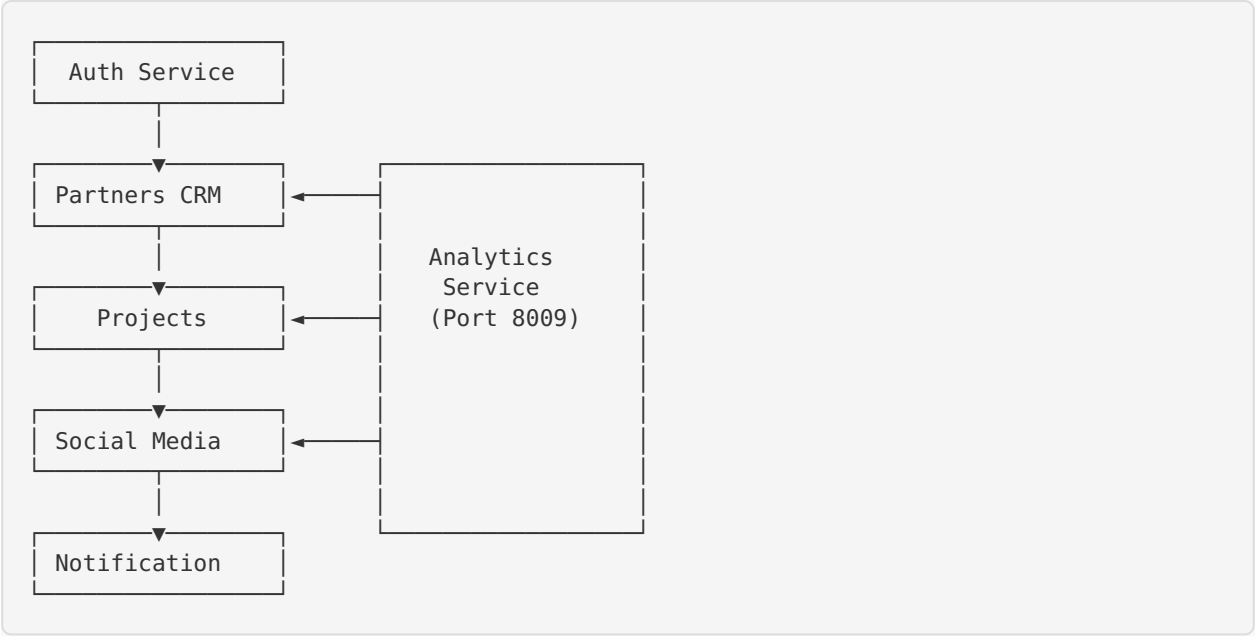
Overview

This guide explains how to integrate the Analytics Service with other Mission Engadi microservices and external applications.

Table of Contents

Service Architecture

The Analytics Service integrates with 6 microservices:



Ports:
- Auth Service: 8002
- Partners CRM: 8003
- Projects: 8004
- Social Media: 8007
- Notification: 8008
- Analytics: 8009

Integration Methods

1. Push Integration (Recommended)

Concept: Services send metrics to Analytics Service in real-time.

Advantages:
- Real-time data
- Lower load on source services
- Immediate insights

Implementation:

```
import httpx

async def send_metric(service_name: str, metric_type: str, metric_name: str, value: fl
oat):
    """Send metric to Analytics Service."""
    async with httpx.AsyncClient() as client:
        await client.post(
            "http://analytics-service:8009/api/v1/metrics",
            json={
                "service_name": service_name,
                "metric_type": metric_type,
                "metric_name": metric_name,
                "value": value,
                "timestamp": datetime.utcnow().isoformat()
            },
            headers={"Authorization": f"Bearer {get_service_token()}"}
        )

# Usage
await send_metric("partners_crm", "donation", "new_donation", 100.00)
```

2. Pull Integration

Concept: Analytics Service pulls data from source services.

Advantages:
- No changes to source services
- Centralized sync logic
- Can handle historical data

Implementation:

Analytics Service automatically pulls data via scheduled sync operations.

3. Event-Based Integration (Future)

Concept: Services publish events to message queue (Kafka/RabbitMQ).

Advantages:
- Decoupled architecture
- Scalable
- Event replay capability

Service-to-Service Integration

Partners CRM Integration

Metrics to Track:
- New partner registrations
- Donations received
- Partner engagement
- Communication history

Example Integration:

```python
from app.services.analytics_client import AnalyticsClient

class PartnerService:
    def __init__(self):
        self.analytics = AnalyticsClient()

    async def create_partner(self, partner_data: dict):
        # Create partner logic
        partner = await self.db.create(partner_data)

        # Send metric to analytics
        await self.analytics.send_metric(
            service_name="partners_crm",
            metric_type="partner",
            metric_name="new_partner",
            value=1.0,
            dimensions={
                "partner_type": partner.type,
                "country": partner.country
            }
        )

        return partner

    async def record_donation(self, donation_data: dict):
        # Record donation logic
        donation = await self.db.create(donation_data)

        # Send donation metric
        await self.analytics.send_metric(
            service_name="partners_crm",
            metric_type="donation",
            metric_name="donation_received",
            value=donation.amount,
            dimensions={
                "partner_id": donation.partner_id,
                "currency": donation.currency
            },
            meta={
                "transaction_id": donation.transaction_id
            }
        )

        return donation
```

Projects Integration

Metrics to Track:
- Project creation/completion
- Beneficiaries reached
- Project milestones
- Impact metrics

Example:

```python
class ProjectService:
    async def complete_project(self, project_id: str):
        project = await self.db.get(project_id)
        project.status = "completed"
        await self.db.update(project)

        # Send completion metric
        await self.analytics.send_metric(
            service_name="projects",
            metric_type="project",
            metric_name="project_completed",
            value=1.0,
            dimensions={
                "project_type": project.type,
                "duration_days": (project.end_date - project.start_date).days
            }
        )

        # Send beneficiary metric
        await self.analytics.send_metric(
            service_name="projects",
            metric_type="beneficiary",
            metric_name="beneficiaries_reached",
            value=project.beneficiaries_count
        )
```

Social Media Integration

Metrics to Track:
- Posts created
- Engagement (likes, comments, shares)
- Reach and impressions
- Platform-specific metrics

Example:

```python
class SocialMediaService:
    async def create_post(self, post_data: dict):
        post = await self.db.create(post_data)

        await self.analytics.send_metric(
            service_name="social_media",
            metric_type="social_post",
            metric_name="post_created",
            value=1.0,
            dimensions={
                "platform": post.platform,
                "content_type": post.content_type
            }
        )

        return post

    async def update_engagement(self, post_id: str, engagement_data: dict):
        # Update engagement logic
        await self.db.update_engagement(post_id, engagement_data)

        # Send engagement metrics
        for metric, value in engagement_data.items():
            await self.analytics.send_metric(
                service_name="social_media",
                metric_type="engagement",
                metric_name=f"post_{metric}",
                value=value,
                dimensions={"post_id": post_id}
            )
```

Notification Integration

Metrics to Track:

- Notifications sent
- Delivery status
- Channel performance
- User engagement

Example:

```python
class NotificationService:
    async def send_notification(self, notification_data: dict):
        # Send notification logic
        result = await self.send(notification_data)

        # Track notification sent
        await self.analytics.send_metric(
            service_name="notification",
            metric_type="notification",
            metric_name="notification_sent",
            value=1.0,
            dimensions={
                "channel": notification_data["channel"],
                "template_id": notification_data.get("template_id")
            }
        )

        # Track delivery status
        if result.delivered:
            await self.analytics.send_metric(
                service_name="notification",
                metric_type="notification",
                metric_name="notification_delivered",
                value=1.0,
                dimensions={"channel": notification_data["channel"]}
            )
```

Client Integration

JavaScript/TypeScript Client

Installation:

```
npm install axios
```

Client Implementation:

```typescript
import axios, { AxiosInstance } from 'axios';

class AnalyticsClient {
  private client: AxiosInstance;

  constructor(baseURL: string, token: string) {
    this.client = axios.create({
      baseURL,
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    });
  }

  async createMetric(data: MetricData) {
    const response = await this.client.post('/api/v1/metrics', data);
    return response.data;
  }

  async getPartnerStatistics(startDate?: string, endDate?: string) {
    const response = await this.client.get('/api/v1/analytics/partners/statistics', {
      params: { start_date: startDate, end_date: endDate }
    });
    return response.data;
  }

  async getDashboard(dashboardId: string) {
    const response = await this.client.get(`/api/v1/dashboards/${dashboardId}`);
    return response.data;
  }

  async getDashboardData(dashboardId: string) {
    const response = await this.client.get(`/api/v1/dashboards/${dashboardId}/data`);
    return response.data;
  }
}

// Usage
const analytics = new AnalyticsClient(
  'http://localhost:8009',
  'your-jwt-token'
);

const stats = await analytics.getPartnerStatistics('2024-01-01', '2024-12-31');
```

Python Client

Installation:

```
pip install httpx
```

Client Implementation:

```python
import httpx
from typing import Optional, Dict, Any
from datetime import date

class AnalyticsClient:
    def __init__(self, base_url: str, token: str):
        self.base_url = base_url
        self.headers = {
            "Authorization": f"Bearer {token}",
            "Content-Type": "application/json"
        }

    async def create_metric(self, data: Dict[str, Any]):
        async with httpx.AsyncClient() as client:
            response = await client.post(
                f"{self.base_url}/api/v1/metrics",
                json=data,
                headers=self.headers
            )
            return response.json()

    async def get_partner_statistics(
        self,
        start_date: Optional[date] = None,
        end_date: Optional[date] = None
    ):
        params = {}
        if start_date:
            params["start_date"] = start_date.isoformat()
        if end_date:
            params["end_date"] = end_date.isoformat()

        async with httpx.AsyncClient() as client:
            response = await client.get(
                f"{self.base_url}/api/v1/analytics/partners/statistics",
                params=params,
                headers=self.headers
            )
            return response.json()

# Usage
analytics = AnalyticsClient("http://localhost:8009", "your-jwt-token")
stats = await analytics.get_partner_statistics()
```

Data Sync

Automatic Sync

Analytics Service can automatically sync data from source services:

Configuration (.env):

```
SYNC_ENABLED=true
SYNC_INTERVAL_MINUTES=60
PARTNERS_CRM_URL=http://partners-crm:8003
PROJECTS_URL=http://projects:8004
SOCIAL_MEDIA_URL=http://social-media:8007
NOTIFICATION_URL=http://notification:8008
```

Manual Sync

Trigger sync via API:

```
curl -X POST http://localhost:8009/api/v1/sync \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "service_name": "partners_crm",
    "sync_type": "incremental"
  }'
```

Trigger full aggregation:

```
curl -X POST http://localhost:8009/api/v1/sync/aggregate-all \
  -H "Authorization: Bearer $TOKEN"
```

Best Practices

1. Metric Naming

Use consistent naming conventions:

- Use snake_case: `new_partner`, `donation_received`
- Be descriptive: `email_sent` not `sent`
- Include action: `partner_created`, `project_completed`

2. Dimensions

Use dimensions for categorical data:

```
dimensions={
    "partner_type": "individual",
    "country": "USA",
    "campaign": "winter-2024"
}
```

3. Metadata

Use metadata for additional context:

```
meta={
    "transaction_id": "txn_123",
    "user_agent": "mobile-app",
    "source": "web"
}
```

4. Error Handling

Always handle analytics errors gracefully:

```python
try:
    await analytics.send_metric(...)
except Exception as e:
    logger.error(f"Analytics error: {e}")
    # Don't fail the main operation
```

5. Batching

Batch metrics when possible:

```python
metrics = [
    {"metric_name": "metric1", "value": 1.0},
    {"metric_name": "metric2", "value": 2.0}
]
await analytics.bulk_create_metrics(metrics)
```

6. Async Operations

Use async for non-blocking analytics:

```python
# Fire and forget
asyncio.create_task(analytics.send_metric(...))

# Or use background tasks in FastAPI
background_tasks.add_task(analytics.send_metric, ...)
```

Troubleshooting

Connection Issues

Problem: Cannot connect to Analytics Service

Solution:
1. Check service is running: `curl http://localhost:8009/api/v1/health`
2. Verify network connectivity
3. Check firewall rules
4. Verify service URL in configuration

Authentication Issues

Problem: 401 Unauthorized errors

Solution:
1. Verify JWT token is valid
2. Check token expiration
3. Ensure correct Authorization header format
4. Verify service has required permissions

Sync Issues

Problem: Data not syncing from source services

Solution:
1. Check sync status: `GET /api/v1/sync/status`
2. Review sync logs
3. Verify source service URLs
4. Check authentication between services
5. Trigger manual sync to test

Performance Issues

Problem: Slow analytics responses

Solution:
1. Check database indexes
2. Review query performance
3. Consider data retention policies
4. Implement caching
5. Scale horizontally if needed

Support

For integration support:
- **Documentation**: See API_DOCUMENTATION.md
- **Issues**: GitHub Issues
- **Email**: support@engadi.org