Abacus AI Deployment Guide for THANOS System

Overview

This guide covers deploying all THANOS orchestration agents and tools to Abacus AI platform.

Prerequisites

- Active Abacus AI account with project access
- Admin permissions for creating agents and functions
- Understanding of Abacus AI project structure

Step 1: Create Base Project

1.1 Create New Project

1. Navigate to Abacus AI Projects
2. Click "Create New Project"
3. Select "ChatLLM - Custom LLM Chat"
4. Name: "THANOS File Organizer"
5. Description: "AI-powered file organization system with one-click functionality"

1.2 Project Configuration

```
Project Settings:
  - Use Case: ChatLLM - Custom LLM Chat
  - Access Control: Everyone (or specific team members)
  - Documentation: Use Case Specific
```

Step 2: Create Data Pipelines

2.1 Upload File Metadata Rules

1. Go to "Datasets" section
2. Click "Create Dataset"
3. Upload `file_metadata_rules_v1.csv` (create if needed)

```
file_type,category,folder_structure,tags
jpg,photos,Photos/{year}/{month},image,photo
pdf,documents,Documents/{category},document,text
mp4,videos,Videos/{year},video,media
docx,documents,Documents/Word,document,office
xlsx,spreadsheets,Documents/Excel,spreadsheet,data
```

2.2 Configure Feature Groups

1. Navigate to "Feature Groups"

2. Click “Attach Existing Feature Group”
3. Create classification rules feature group

Step 3: Deploy Agent Tools

3.1 Guard Rail Tool

```
# Copy the guard-rail.py content
# Navigate to Functions in Abacus AI
# Create New Function -> Python Function
# Name: "guard_rail_tool"
# Paste the guard_rail_function code
```

Function Configuration:
- Name: `guard_rail_tool`
- Input Schema: `{"user_id": "string", "org_id": "string", "scope": "string", "tier": "string"}`
- Output Schema: `{"ok": "boolean", "warnings": ["string"], "quotas": {}, "permissions": {}}`

3.2 List Files Tool

```
# Copy the list-files.py content
# Create New Function -> Python Function
# Name: "list_files_tool"
```

Function Configuration:
- Name: `list_files_tool`
- Input Schema: `{"scope": "string", "cursor": "string", "limit": "number", "filters": {}}`
- Output Schema: `{"files": [], "total_found": "number", "next_cursor": "string"}`

3.3 Extract EXIF Tool

```
# Copy the extract-exif.py content
# Create New Function -> Python Function
# Name: "extract_exif_tool"
```

Function Configuration:
- Name: `extract_exif_tool`
- Input Schema: `{"file_key": "string"}`
- Output Schema: `{"file_key": "string", "has_exif": "boolean", "gps": {}, "camera": {}}`

3.4 Classify File Tool

```
# Copy the classify-file.py content
# Create New Function -> Python Function
# Name: "classify_file_tool"
```

Function Configuration:
- Name: `classify_file_tool`
- Input Schema: `{"file_key": "string", "metadata": {}, "text_content": "string"}`
- Output Schema: `{"primary_category": "string", "tags": [], "confidence": "number"}`

Step 4: Create Document Retrievers

4.1 URL Classification Retriever

1. Go to “Document Retrievers” section
2. Click “Create Document Retriever”
3. Name: “url_classification_retriever”
4. Configure to handle file classification rules

4.2 Content Analysis Retrievers

Create additional retrievers for:
- Image content analysis
- Document text extraction
- Metadata processing

Step 5: Deploy SnapOrchestrator Agent

5.1 Create Main Agent

1. Navigate to “Models” section
2. Click “Create Model”
3. Select “Custom Agent”
4. Name: “SnapOrchestrator”

5.2 Agent Configuration

```
# Upload the snap-orchestrator.yaml configuration
# Configure input/output schemas
# Link to deployed tools and retrievers
```

Agent Schema:

```json
{
  "input": {
    "job_id": "string",
    "user_id": "string",
    "org_id": "string",
    "tier": "string",
    "scope": "string",
    "dry_run": "boolean"
  },
  "output": {
    "summary": {
      "total_files": "number",
      "folders_created": "number",
      "files_moved": "number",
      "undo_bundle": []
    }
  }
}
```

5.3 Agent Logic

```python
# Main orchestration logic
def orchestrate_file_organization(input_data):
    # 1. Run guard_rail_tool
    # 2. Run list_files_tool
    # 3. For each file:
    #    - extract_exif_tool
    #    - classify_file_tool
    # 4. Generate folder structure
    # 5. Move files
    # 6. Generate summary
    pass
```

Step 6: Create Deployment

6.1 Deploy SnapOrchestrator

1. Go to “Deployments” section
2. Click “Start Deployment”
3. Select SnapOrchestrator model
4. Name: “thanos-orchestrator”
5. Configure API endpoints

6.2 Deployment Configuration

```yaml
Deployment Settings:
  - Name: "thanos-orchestrator"
  - Model: SnapOrchestrator
  - Memory: 4GB
  - CPU: 2 cores
  - Timeout: 30 minutes
  - Batch Size: 50 files
```

Step 7: Test Deployment

7.1 Test Individual Tools

```bash
# Test guard rail
curl -X POST https://your-deployment-url/guard_rail_tool \
  -H "Content-Type: application/json" \
  -d '{"user_id": "test", "org_id": "test", "scope": "/test", "tier": "Standard"}'

# Test file listing
curl -X POST https://your-deployment-url/list_files_tool \
  -H "Content-Type: application/json" \
  -d '{"scope": "/test/files", "limit": 10}'
```

7.2 Test Full Orchestration

```bash
# Test complete workflow
curl -X POST https://your-deployment-url/orchestrate \
  -H "Content-Type: application/json" \
  -d '{
    "job_id": "test-job-123",
    "user_id": "test-user",
    "org_id": "test-org",
    "tier": "Standard",
    "scope": "/test/unorganized",
    "dry_run": true
  }'
```

Step 8: Monitor and Scale

8.1 Monitoring Setup

1. Enable logging in Deployments
2. Set up alerts for failures
3. Monitor performance metrics

8.2 Scaling Configuration

```yaml
Auto-scaling:
  - Min instances: 1
  - Max instances: 5
  - Scale trigger: Queue length > 10
  - Scale down: Idle time > 5 minutes
```

Step 9: API Integration

9.1 Get API Endpoints

After deployment, note down:
- Orchestrator endpoint URL
- API authentication method
- Rate limits and quotas

9.2 Update Web App Configuration

Update the Next.js app environment variables:

```
ABACUS_ORCHESTRATOR_URL=https://your-deployment-url
ABACUS_API_KEY=your-api-key
ABACUS_PROJECT_ID=your-project-id
```

Troubleshooting

Common Issues

1. **Function timeout**: Increase timeout in deployment settings
2. **Memory errors**: Increase memory allocation

3. **API rate limits**: Implement proper request throttling
4. **Data pipeline errors**: Check dataset format and schemas

Debug Steps

1. Check function logs in Abacus AI
2. Test individual tools before full orchestration
3. Use dry_run mode for testing
4. Monitor resource usage during execution

Security Considerations

1. **API Keys**: Store securely, rotate regularly
2. **Access Control**: Limit deployment access to authorized users
3. **Data Privacy**: Ensure file content is not logged permanently
4. **Network Security**: Use HTTPS for all API calls

Next Steps: After successful deployment, integrate with the Next.js web application for complete THANOS system functionality.