Implementing STAC Data Organization in Azure Orbital

Shane Trimbur

January 6, 2025

Abstract

This whitepaper outlines a detailed implementation plan for organizing STAC (SpatioTemporal Asset Catalog) data using Azure Orbital. The steps described provide a structured approach to ingest, process, and manage geospatial data, leveraging Azure's cloud-native capabilities. This document references the Microsoft Azure Orbital documentation available at Azure Orbital Documentation.

1 Introduction

SpatioTemporal Asset Catalog (STAC) is an open standard for indexing and discovering geospatial data. Azure Orbital allows users to manage satellite data efficiently, offering features for processing and storing data at scale. This paper details the implementation of a STAC-compatible data organization pipeline using Azure Orbital.

2 Prerequisites

The following prerequisites are necessary for the implementation:

- Azure subscription with Orbital service enabled.
- Familiarity with STAC specifications (https://stacspec.org).
- Azure Storage and Azure Data Lake Gen2.
- Azure Functions or Logic Apps for automation.
- Python SDK for STAC operations.

3 System Architecture

The architecture for organizing STAC data in Azure Orbital is depicted in the diagram below (adapted from the Azure Orbital Documentation):

Image 2: Azure STAC Pipeline

4 Data Flow Explanation

The data flow in Azure Orbital for organizing STAC data is structured into key phases: **Data Acquisition, Data Ingestion, Metadata Generation, Cataloging, and Data Discovery**. The detailed flow is as follows:

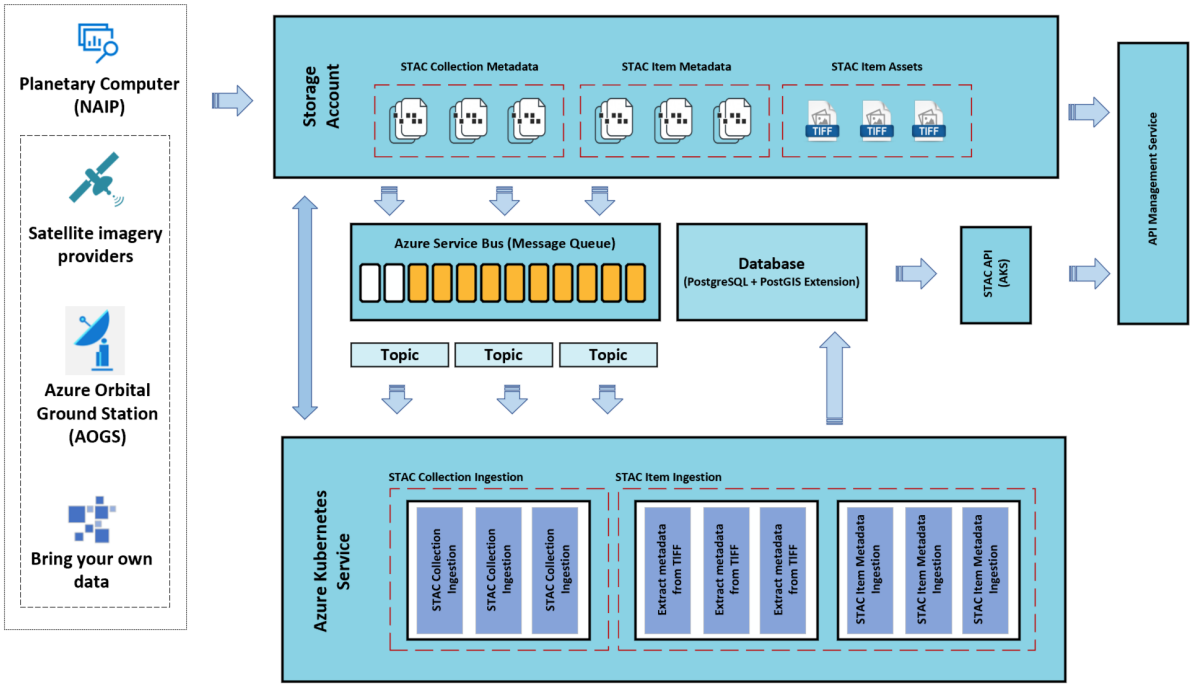

Figure 1: Pipeline architecture diagram showing individual components and interactions.

4.1 1. Data Acquisition

This phase involves acquiring satellite imagery data from multiple sources, including:

- Planetary Computer (NAIP)
- Satellite Imagery Providers
- Azure Orbital Ground Station (AOGS)
- User-provided data

The raw data is uploaded to Azure Storage for further processing.

4.2 2. Data Ingestion

The ingestion phase processes the uploaded data into a STAC-compatible format. The key tasks are:

1. The **Engineer** uploads *STAC Collection JSON* for collections or *STAC Item Metadata* for items into Azure Storage.
2. Azure Service Bus triggers an event message to inform downstream services that new data is ready for processing.
3. The **Processor (AKS)** subscribes to the Service Bus topic, retrieves the messages, and processes the data:
    - For collections: Add STAC Collection to the PostgreSQL database.
    - For items: Upload raster data and add STAC Item metadata.

Practical Steps:
- Use Azure CLI or Azure Portal to set up Storage, Service Bus, and AKS instances.
- Ensure proper Service Bus topics and subscriptions are configured for ingestion triggers.

4.3 3. Metadata Generation

Metadata for both STAC Collections and Items is generated and stored in the PostgreSQL database:

- STAC Collection JSON defines the collection-level metadata.
- STAC Item JSON adds individual asset metadata (raster data, timestamp, coordinates).
- Both metadata types are validated against STAC specifications.

Practical Steps:
- Use the `pystac` library to generate and validate STAC metadata:

```
from pystac import Item
item = Item(id="example-id",
            geometry={"type": "Point", "coordinates": [125.6, 10.1]},
            bbox=[125.6, 10.1, 125.6, 10.1],
            datetime="2024-12-15T00:00:00Z",
            properties={})
item.save_object()
```

4.4 4. Cataloging

The processed metadata is added to a STAC Catalog for organization and indexing:

1. PostgreSQL serves as the backend database to store STAC metadata.
2. The **Processor (AKS)** updates the STAC catalog dynamically.
3. Data is normalized using appropriate STAC endpoints.

Practical Steps:
- Create a catalog using Python:

```
from pystac import Catalog
catalog = Catalog(id="example-catalog", title="Example Catalog")
catalog.add_item(item)
catalog.save_object()
```

4.5 5. Data Discovery

The **Analyst** queries the STAC API to retrieve metadata and assets:

1. Queries can be run on STAC Collections or Items to filter specific data.
2. Results are returned in a STAC-compatible format, facilitating easy access and use.

Practical Steps:
- Query the STAC API endpoints using tools like Postman or Python requests:

```
import requests
response = requests.get("https://example.com/stac/collections")
print(response.json())
```

4.6 6. Detailed Explanation of STAC Data Pipeline Images

Two critical diagrams, **azure_stac_dataflow.png** and **azure_stac_pipeline.png**, illustrate the internal workings and processes for organizing STAC data efficiently.

Image 1: azure$_{s}tac_{d}ataflow.png$ The *data flow diagram* outlines end-to-end workflows:

1. **Input Sources:** Satellite imagery providers, Azure Orbital Ground Station, Planetary Computer, and user-provided data feed raw data into Azure Storage.
2. **Message Queueing:** Azure Service Bus acts as an event-driven queue, managing communication between ingestion triggers, data processors, and databases.
3. **Processing Layers:** Kubernetes on AKS processes metadata and raster assets, dynamically validating and transforming raw data into STAC-compatible formats.
4. **Catalog Management:** PostgreSQL + PostGIS store and index metadata to enable scalable geospatial queries.
5. **API Integration:** A deployed STAC API on AKS exposes endpoints for analysts, facilitating streamlined querying and data discovery.

Image 2: azure$_{s}tac_{p}ipeline.png$ The *pipeline architecture diagram* focuses on individual components and the interaction between them:

- Azure Storage organizes **STAC Collection Metadata, Item Metadata, and Raster Assets (TIFFs)**.
- Azure Service Bus Topics provide message queueing for collection ingestion, metadata extraction, and item processing tasks.
- **AKS Processors** extract metadata dynamically from raster data and ingest them into the STAC-compatible format.
- PostgreSQL with PostGIS stores geospatial data, enabling advanced spatial queries.
- API Management Services expose the data to analysts via the STAC API.

Step-by-Step Implementation Guide for Both Diagrams

1. **Set Up Azure Storage:** Upload raw imagery, JSON metadata, and raster assets.
2. **Deploy Service Bus Topics:** Automate queue setup for task triggers.
3. **Provision AKS:** Deploy containerized Python STAC processors for metadata extraction.
4. **Integrate PostgreSQL with PostGIS:** Enable spatial indexing and storage for metadata.
5. **Deploy the STAC API:** Use AKS to expose a scalable API endpoint.
6. **Test the Pipeline:** Upload sample imagery and validate metadata ingestion and querying.

5 Conclusion

Organizing STAC data in Azure Orbital enables scalable and efficient geospatial data management. This whitepaper provides a step-by-step guide to implement the solution, leveraging Azure services and STAC standards for seamless integration.

6 References

- Azure Orbital Documentation: https://learn.microsoft.com/en-us/azure/orbital/organize-stac-data
- STAC Specification: https://stacspec.org
- Azure CLI Documentation: https://learn.microsoft.com/en-us/cli/azure/