ESAC TAP stateless

IVOA June 2025 Interoperability Meeting
Jose Osinde (Starion for ESA)
Ignacio Leon (AURORA Technology B.V.)
C. Rios (Starion for ESA, M. Henar (Starion for ESA), J. Ballester (Starion for ESA)
R. Parejo (Starion for ESA), R. Bhatawdekar (ESA)
SCO-08: Archives Software Development

ESAC
Camino Bajo del Castillo s/n, Urb. Villafranca Del Castillo
28692 Villanueva de la Cañada (Madrid) Spain

Preparing TAP+ for higher loads and complexity

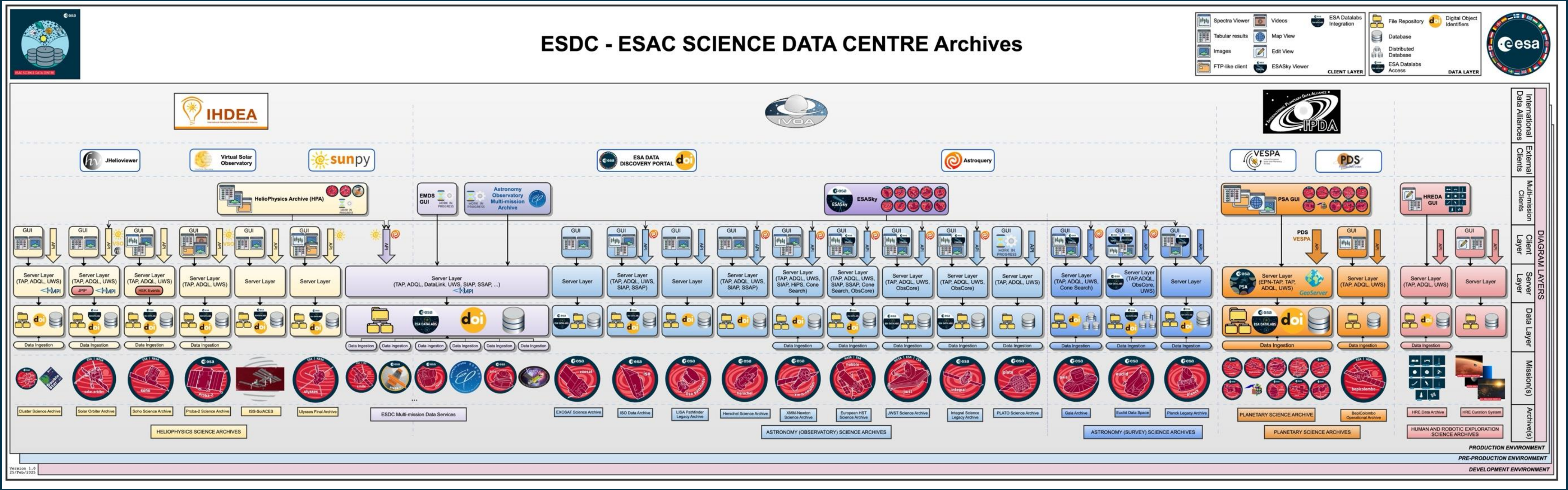
TAP is becoming a fundamental component of the ESDC Archives. Looking ahead, we need to take a step forward and ensure a service that offers:

- Improved robustness
- Fault tolerance in case one of server crash
- Better support of high load peaks.
    - In a standard TAP service most of the load typically resides in the backend (the database). ESDC provides additional functionality like on-the-fly data combination and formats conversions that add significant load to the TAP server itself
- Scalability to handle more simultaneous requests from different users
- Additional flexibility to deal with new requirements

What is the scenario we want to dealt with?

- Archives are growing in size, user base, and number of features:
    - Repositories in the order of PetaBytes
    - Request for on-the-fly data conversions
    - Compressing and packing data before download
- Users are increasingly demanding integrated solutions that allow easy combination of data from multiple sources. This is currently done in two ways:
    - Combine data from multiple missions within a single archive increasing it complexity
    - Combine data from different TAP services within the same UI:
        - ESASky
        - ESDC Multi-mission Data Service (EMDS, currently just SMILE mission)
        - HelioPhysics Archive (HPA)

What is the scenario we want to dealt with?



ESAC SCIENCE DATA CENTRE
ESDC - ESAC SCIENCE DATA CENTRE Archives
Spectra Viewer
Tabular results
Images
FTP-like client
Videos
Map View
Edit View
ESASky Viewer
ESA Datalabs Integration
CLIENT LAYER
File Repository
Database
Distributed Database
ESA Datalabs Access
Digital Object Identifiers
DATA LAYER
IHDEA
IVOA
IPDA
JHelioviewer
Virtual Solar Observatory
sunpy
ESA DATA DISCOVERY PORTAL
Astroquery
VESPA
PDS
HelioPhysics Archive (HPA)
EMDS GUI
WORK IN PROGRESS
Astronomy Observatory Multi-mission Archive
ESASky
PSA GUI
HREDA GUI
GUI
API
VSO
PDS VESPA
Server Layer (TAP, ADQL, UWS)
JPIP
HEK Events
Server Layer
Server Layer (TAP, ADQL, DataLink, UWS, SIAP, SSAP, ...)
Server Layer (TAP, ADQL, UWS, SIAP, SSAP)
Server Layer (TAP, ADQL, UWS, SIAP, HiPS, Cone Search)
Server Layer (TAP, ADQL, UWS, SIAP, SSAP, Cone Search, ObsCore)
Server Layer (TAP, ADQL, UWS, ObsCore)
Server Layer (TAP, ADQL, UWS, Cone Search)
Server Layer (TAP,ADQL, ObsCore, UWS)
Server Layer (EPN-TAP, TAP, ADQL, UWS)
PSA
GeoServer
ESA DATALABS
doi
Data Ingestion
Cluster Science Archive
Solar Orbiter Archive
Soho Science Archive
Proba-2 Science Archive
ISS-SolACES
Ulysses Final Archive
ESDC Multi-mission Data Services
EXOSAT Science Archive
ISO Data Archive
LISA Pathfinder Legacy Archive
Herschel Science Archive
XMM-Newton Science Archive
European HST Science Archive
JWST Science Archive
Integral Science Legacy Archive
PLATO Science Archive
Gaia Archive
Euclid Data Space
Planck Legacy Archive
PLANETARY SCIENCE ARCHIVE
BepiColombo Operational Archive
HRE Data Archive
HRE Curation System
HELIOPHYSICS SCIENCE ARCHIVES
ASTRONOMY (OBSERVATORY) SCIENCE ARCHIVES
ASTRONOMY (SURVEY) SCIENCE ARCHIVES
PLANETARY SCIENCE ARCHIVES
HUMAN AND ROBOTIC EXPLORATION SCIENCE ARCHIVES
International Data Alliances
External Clients
Multi-mission Clients
Client Layer
Server Layer
Data Layer
Mission(s)
Archive(s)
DIAGRAM LAYERS
PRODUCTION ENVIRONMENT
PRE-PRODUCTION ENVIRONMENT
DEVELOPMENT ENVIRONMENT
Version 1.0
25/Feb/2025

Advantages of a stateless service

A stateless service does not store any client session information between requests. Each request is independent and self-contained. This architectural approach offers several key advantages:

- Scalability. Stateless services are easier to scale horizontally (add more instances) because any instance can handle any request—no need to track session state.
- Resilience & Fault Tolerance .If a service instance fails, another can take over immediately without loss of session information.
- Better Load Balancing. Load balancers can distribute requests freely across instances without worrying about session affinity (a.k.a. "sticky sessions").
- Improved Performance. No need to read or write session data, so responses can be faster, especially in high-throughput scenarios.
- Easier Maintenance and Upgrades

TAP+ is not a stateless service

Most of the advantages offered by a stateless service are aligned with what we want for our future TAP. However, **TAP is currently not a stateless service**, and this is mainly because key data is stored in server memory, tightly coupled to the servlet container:

- A session with associated privileges and quotas is created for any logged user and is kept in memory.
- We must track jobs associated with each user — even anonymous ones — including job status and results, all of which are held in memory.
- Events, notifications, and status updates are also cached in the server's memory.

This architecture works for a single-instance setup but **prevents horizontal scaling and load balancing**.

Our goal is to evolve TAP into a service that can be fully integrated into a load-balanced environment.

To achieve this, all session, job, and event-related information must be moved from in-memory storage to a shared database. This will allow multiple instances to access the same data reliably, enabling scalability. It also opens the door to containerizing and deploying the entire service within an orchestrated environment, such as a cloud platform.

Session handling

- TAP library is now based on the Spring framework and introduce Spring Session for standardized session management
- Session data is now persisted in a shared relational database, enabling consistent user sessions across multiple TAP instances
- Some or the pros and cons associated with this approach are:

Pros

- Enables session sharing across multiple application instances (Scalability).
- Sessions survive application/server restarts or crashes (High Availability)
- By storing session data externally, services can be easily deployed in containers and integrated into cloud environments

Cons

- Each session read/write involves a DB query, adding overhead compared to in-memory sessions (Increased Latency)
- Requires database schema setup, tuning, and ongoing maintenance (Higher Complexity)
- Session handling becomes dependent on database uptime and performance (Dependency on DB Availability)

Such trade-offs are actually common to all the features presented in the following slides.

Database access

- Transactional queries are already safe in a multi-client environment
- We now need to review queries related to Data Manipulation Language (DML)
    - This focuses on data content, not on structure (DDL), permissions (DCL), or transaction control (TCL).
- DML operations include: SELECT, **INSERT**, **UPDATE** and **DELETE**.
- Even when focusing only on DML, if we intend to share the same database across different TAP instances, we must carefully review all methods using these operations to ensure proper transaction handling where required.
- Main use cases include:
    - Register new tables in the tap schema (including table name, columns and types)
    - Job management

UWS events

This information needs now to be shared between the different instances

Jobs

- JOB_CREATED_EVENT
- JOB_UPDATED_EVENT
- JOB_REMOVED_EVENT

Login

- LOGIN_IN_EVENT
- LOGIN_OUT_EVENT

Quota

- QUOTA_DB_UPDATED_EVENT
- QUOTA_FILE_UPDATED_EVENT

Notifications

- NOTIFICATION_CREATED_EVENT
- NOTIFICATION_REMOVED_EVENT

Sharing

- SHARE_ITEMS_CREATED_EVENT
- SHARE_ITEMS_UPDATED_EVENT
- SHARE_ITEMS_REMOVED_EVENT
- SHARE_GROUPS_CREATED_EVENT
- SHARE_GROUPS_UPDATED_EVENT
- SHARE_GROUPS_REMOVED_EVENT
- SHARE_USERS_CREATED_EVENT
- SHARE_USERS_UPDATED_EVENT
- SHARE_USERS_REMOVED_EVENT

Service start/stop I

A TAP service must now consider the possibility of being configured with multiple instances.

- Startup clean-up procedures need to be redefined to support this setup:
    - How many instances should run in parallel
    - Can this be adjusted in real time? How does this impact the overall service?
    - How should pending jobs be handled? Who is responsible for cleaning them up, and when?
- Which model should we choose: Master/slave architecture or a decentralized one
- At the same time, we must maintain backward compatibility with archives running on a single-server configuration.

Service start/stop II

When a TAP instance starts:

- It assigns itself a unique ID
- It clears out old entries from its internal list of instances — other TAP instances not active anymore.
- It goes through all unfinished tasks and checks:
  - If the task was owned by an inactive or now-missing instance, the service will take over and decide what to do

Potential risk:

- If multiple service instances start at the same time, they might try to take over the same task. To avoid this:
  - Each task takeover is done carefully, step-by-step, to make sure only one instance takes control:
    1. The task is assigned to the new instance.
    2. Then it is restarted, cancelled, or deleted (depending on the configured policy)
    3. The system moves to next task but not before it make sure the overall system is in sync again

Behind-the-scenes improvements:

- Every task now keeps track of which instance is managing it.
- Whenever a task's status changes, that tracking information is updated to reflect who is in charge in the new status.

System tests

The architecture is now more complex, and so are the test required to cover the different use cases. We are now dealing with issues similar to those encountered in multi-thread environments:

- The setup must launch multiple instances in parallel
- We need to test interactions between two or more TAP instances sharing the same database. Typical use cases include:
    - Ensure that tables registered in one instance are visible to the others
    - Retrieving and manage jobs created from different instances
    - Handling shared resources (tables, results), and more
- Race conditions must be carefully considered and tested
- Another non-trivial scenario is to testing the service during startup and shutdown sequence, as described in the previous slide

Questions / feedback

Thank you for your attention