

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

→ THE EUROPEAN SPACE AGENCY

ESA UNCLASSIFIED – Releasable to the Public



TAP is already 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 crashes
- 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

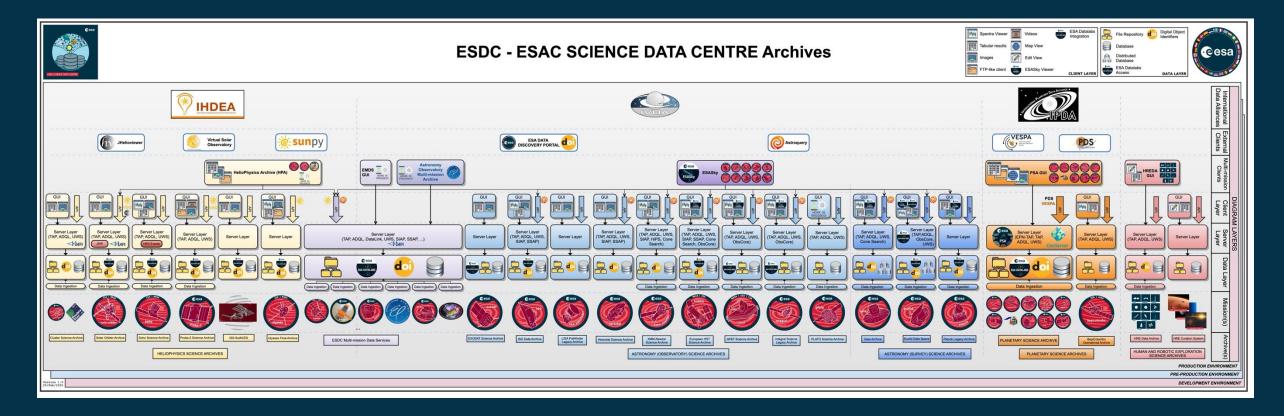
💳 💶 📕 🛨 💳 📰 📲 💶 💶 📲 💶 👬 💳 🗤 🖗 🔤 🔤 👘



- 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:
 - <u>ESASky</u>
 - ESDC Multi-mission Data Service (EMDS, currently just SMILE mission)
 - <u>HelioPhysics Archive</u> (HPA)

💳 📕 🚼 🧮 💳 🔚 📕 🗮 💳 📕 📕 💳 👬 💳 🛶 🚳 🛌 📕 🖬 🗮 💳 🔤 🛶 🖓







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

💳 📕 🚼 🧮 💳 🚼 📕 🏣 🔜 📕 🗮 💳 🗮 🔤 🗮 🔤 வ 🚳 🏹 📲 🔂 💳 📰 🐷 🐨 👘 🔸 The European Space Agenc'

• esa

Most of the advantages offered by a stateless service are aligned with what we want for our future TAP. However, **TAP is 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 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.

H= ___ III → HE EUROPEAN SPACE AGENCY



- 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 metadata management
 - Upload user datasets

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

Notifications

۲

- ۲

- QUOTA DB UPDATED EVENT

QUOTA FILE UPDATED EVENT

NOTIFICATION CREATED EVENT

NOTIFICATION REMOVED EVENT

SHARE_USERS_REMOVED_EVENT

Sharing

- SHARE_ITEMS_CREATED_EVENT
- SHARE ITEMS UPDATED EVENT
- SHARE ITEMS REMOVED EVENT
- SHARE GROUPS CREATED EVENT ۲
- SHARE_GROUPS_UPDATED_EVENT
- SHARE GROUPS REMOVED EVENT
- SHARE_USERS_CREATED_EVENT
- SHARE_USERS_UPDATED_EVENT

→ THE EUROPEAN SPACE AGENCY



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 singleserver configuration, trying to include the minimum changes possibles in this setup



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.



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

