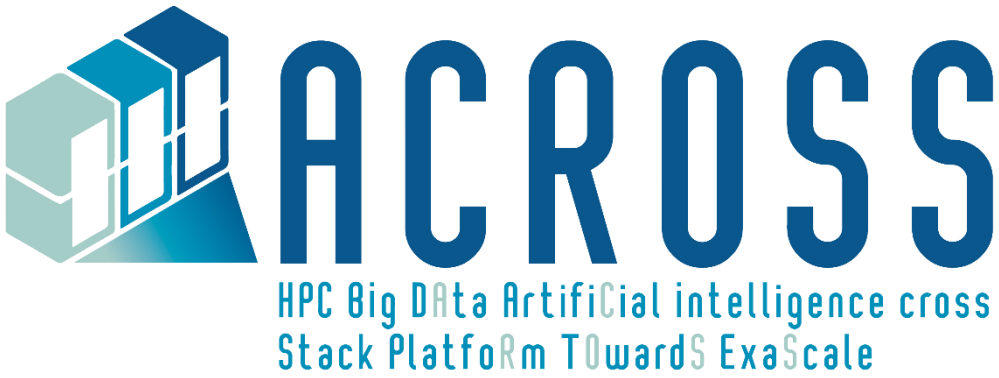

D6.2 – Extend the domain-specific object store to support Grand Ensemble and storm-resolving climatological simulations

Deliverable ID	D6.2
Deliverable Title	Extend the domain-specific object store to support Grand Ensemble and storm-resolving climatological simulations
Work Package	WP6
Dissemination Level	PUBLIC
Version	1.0
Date	2022-02-28
Status	Submitted
Deliverable Leader	MPI-M
Main Contributors	MPI-M, ECMWF

Published by the ACROSS Consortium

Document History

Version	Date	Author(s)	Description
0.1	2022-01-25	MPI-M (S. Willner)	Structure defined
0.2	2022-01-28	MPI-M (S. Willner)	Rough draft
0.3	2022-02-03	MPI-M (S. Willner, J. Segura)	Added architecture overview and further paragraphs
0.4	2022-02-08	MPI-M (S. Willner)	Added introduction and requirements
0.5	2022-02-18	ECMWF (E. Danovaro)	Added FDB and Polytope descriptions
0.6	2022-02-21	MPI-M (S. Willner, J. Segura)	Completed draft
0.7	2022-02-22	MPI-M (S. Willner)	Finalized version for review
0.8	2022-02-25	MPI-M (R. Budich, J. Segura)	Rephrasing and spell check
1.0	2022-02-28	MPI-M (S. Willner)	Incorporated reviewer feedback from Deltares and ECMWF

Table of Contents

Glossary

Acronym	Explanation
API	Application Programming Interface
DB	Data base
FDB	Field DataBase
GRIB	General Regularly-distributed Information in Binary form
HPC	High Performance Computing
HPDA	High Performance Data Analysis
HSM	Hierarchical Storage Management
HTTP	HyperText Transfer Protocol
ICON	ICOsahedron Non-hydrostatic – Earth system model framework
JSON	JavaScript Object Notation
MARS	Meteorological Archival and Retrievable System
MPI	Message Passing Interface
NVRAM	Non-Volatile Random Access Memory
POSIX	Portable Operating System Interface
RDMA	Remote Direct Memory Access
REST	REpresentational State Transfer
SLURM	Simple Linux Utility for Resource Management (job scheduler)
SSD	Solid State Disk
WMO	World Meteorological Society

List of figures

Executive Summary

Climate modeling increasingly strives for higher resolutions in its simulations in space and, such, in time. Accordingly, the amount of data handled and produced by competitive Earth system models is ever increasing, posing an increasing challenge to the hardware and software used in this context. Whereas computing power has become cheaper and more readily available in recent decades, input-output infrastructures have to keep up with the data generated. Accordingly, weather and climate forecasting is one of the three pilot projects in the ACROSS project.

MPI-M operates such a modern, competitive Earth system model, ICON, and desires to improve its data management workflow, especially for large ensembles of experiments as well as storm-resolving model runs. ECMWF, on the other hand, has decade-long experience of operating such a data workflow including its domain-specific object storage FDB. Whereas ECMWF uses its framework mainly for operational weather forecasts, MPI-M as a basic research institution has different requirements and access patterns for its applications. These requirements as well as the necessary steps for building such a data management workflow are laid out in this deliverable.

Position of the deliverable in the whole project context

This deliverable is part of Work Package 6 “Weather, Climate, Hydrological and Farming Pilot”. In particular, it reports on the continued work undertaken in Task 6.1 “Data and metadata modelling for object-store integration”. Overall, it defines and describes the Exploitable Result ER16. The first prototype directly improves the infrastructure as needed for the project partner Deltares and their WFLOW hydrological model in Task 6.2 “Hydrological workflow simulations of the Rhine and Meuse basins”. The lessons learned when adopting this framework to other infrastructures, including the necessary tests, as developed in WP2 and WP3 will pose additional feedback for these work packages.

Thus, this deliverable directly contributes to the objectives of WP6:

1. Improve the existing operational system for global numerical weather prediction, post-processing and data delivery by exploiting hardware-acceleration and data streaming/object store techniques to demonstrate exascale scalability.
2. Enable low-latency exploitation of climate simulations by integrating data delivery trough domain-specific object store
3. Develop and demonstrate an environment for user-defined in-situ data processing. The system will enable HPDA on multi-petabyte meteorological and climatological archives and data streams to enable data analytic workflows that improve insight to data.

Description of the deliverable

Deliverable D6.2 describes the efforts of developing a framework to incorporate the domain-specific object store (FDB) in a semantic data management workflow for the ICON Earth system model. In the first version this includes the context and requirements for this framework as identified in the beginning of the project as well as a description of the implementation and architecture of the framework being developed. In the updated second version, expected for 2022 Q2, the first results in terms of lessons learned and benchmarks on various infrastructures will be added.

1 Introduction

Deliverable D6.2 is part of Work Package 6 of the ACROSS project, which focuses on one of the three pilot projects in ACROSS, namely the “Weather, Climate, Hydrological and Farming Pilot”. As shown in Figure 1, WP6 itself has three major components: the time critical operation workflow around the weather model IFS, the high performance data analytics including hydrological and agricultural models using climate and weather data as input, and large-scale climate simulations based on the ICON Earth system model[1]. As a connecting element, the work package incorporates the high performance distributed object store FDB[2], which is to store data from the weather model (mainly time frames) as well as from the climate model (time series). This object store then functions as the data source to provide the downstream models with the needed input data.

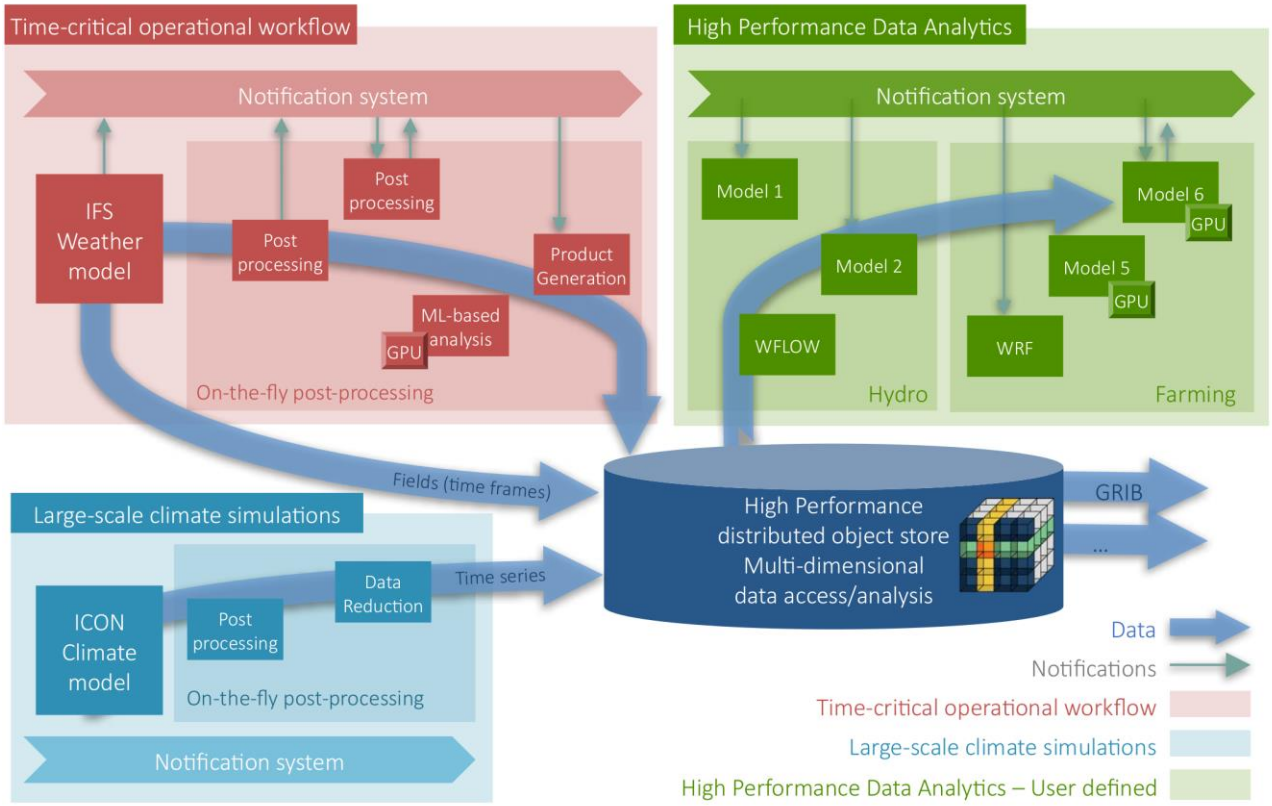

Figure 1 - Overview of the data flow in WP6 “Weather, Climate, Hydrological and Farming Pilot”. This deliverable defines the particular work on the framework for “Large-scale climate simulations” (light blue, lower left) and its adoption of the “High Performance distributed object store” (dark blue, center)

The focus of this deliverable is the module on large-scale climate simulations and the way the ICON climate model is included in the depicted workflow. In particular, this includes the identified requirements and the envisioned architecture to fulfill these requirements in the “BORGES” (**B**etter **ORG**anized **E**arth **S**ystem model data) data system. As the ACROSS endeavor is also focused on experimental setup, BORGES will be setup and continuously evaluated using benchmarks – it might, however, also be subject to further changes as the

1 Zaengl, G., D. Reinert, P. Ripodas, and M. Baldauf, 2015: The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core. Q. J. R. Meteorol. Soc., 141, 563–579. https://doi.org/10.1002/qj.2378

2 Smart, S., T. Quintino, and B. Raoult, 2019: A High-Performance Distributed Object-Store for Exascale Numerical Weather Prediction and Climate. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (*PASC '19*). https://doi.org/10.1145/3324989.3325726

project progresses. The lessons learned, first experiences made along the implementation, as well as the identified changes needed will be included in the second volume of this deliverable in project month M24.

This document is organized as follows. The next section, Section 2, introduces the context and requirements identified so far, from potential users of the data system to technical requirements towards data and efficiency. Section 3 describes the BORGES architecture as it is currently being implemented as a prototype for such a data system. The individual components of the BORGES data system are then described in more detail in Section 4. Section 5 as for the first version, this section will remain unused, but in the second volume, this section will include the results that would be collected in first two years of the project.

1.1 Scope

As part of Work Package 6 this deliverable functions as a progress report on Task 6.1 “Data and metadata modelling for object-store integration”. In particular, the BORGES data system described here is one of the Exploitable Results, ER16, which is to be developed as a demonstrator with a Technological Readiness Level of at least TRL6 by the end of the overall ACROSS project. The BORGES data system as described in this deliverable further is the foundation of the work on providing data for the hydrological model downstream as done in Task 6.2 “Hydrological workflow simulations of the Rhine and Meuse basins”. With the focus on the climate data part of WP6, this data system will further be tested on the ACROSS infrastructure in the course of Task 6.4 “Complex Weather and Climate workflows at pre-exascale”.

1.2 Related documents

ID	Title	Reference	Version	Date
D2.1	D2.1 – Summary of Pilots co-design requirements		v3.0	2021-08-31
D2.2	D2.2 – Description of key technologies and platform design		v0.8	2021-11-30
	ACROSS Exploitable results table		v5.0	2022-01-25

2 Context and requirements

This section describes the context in which the BORGES data system is being built including the most important requirements identified so far. Most of these are not specified to full extent yet as they are constantly evaluated in close contact to planned and potential users. Also, many of these are not time critical throughout the implementation of the first prototype, but should be kept in mind though not to hinder their fulfillment in the successive development into an operational system. The following also includes potential extensions as can be added to BORGES once it reaches an operational state – this is, however, out of scope of the ACROSS project itself.

2.1 Target users

In comparison to weather models, climate models are rarely run in an operational mode. While some standardized, almost operational efforts, such as the Coupled Model Intercomparison Project (CMIP), exist in the climate research community, most simulations target specific research questions and are run by smaller groups or individual researchers. Accordingly, a data system for climate model data has to account for a large number of different users with different behavior, requirements, workflows. Additionally, these users are highly versatile in their efforts to change the model, experiment with model output, and in general have a variety of interests that should not interfere with the system's coherence, nor should the system restrict users in their flexibility in carrying out research experiments and analyzing data (Requirement **RqU1 Multi-user**).

The data system also involves components in need of computational or memory resources. These should accordingly be included in the resource accounting system in the same way as the model run itself. In particular, basic post-processing such as data reduction or compression should happen in the user's HPC job (**RqU2 Resource accounting**). With several data back-ends for storage, however, the system will need to include its own data storage accounting overarching all back-ends. As a low priority requirement this is planned for after the demonstrator stage (extension ExU1 Storage accounting).

One of the key issues identified in this context is the organization of the various output data the climate model produces. Here, a semantic rather than a classical file-based organization of data sets is desired (**RqU3 Key-value access**). The FDB as storage library for the envisaged BORGES data system already provides key-value based storage and access to files by design. The interface between this library and the actual users accordingly should directly support this way of access and, at the same time, mimic libraries and interfaces users are already used to. For the first prototype, this should include two different perspectives of data access: cdo[3] and Intake[4] (**RqU4 Multi-UI**). The Climate Data Operators (cdo) are a tool for high-performance processing of climate data, which is highly popular in the climate research community. Its focus is on the processing itself with a vast set of operators. On the other hand, Intake is a Python library to catalog data across files and to retrieve these in a key-value type manner following a schema defining the catalog. Here, the library can be extended to use the climatological data system as a back-end and users can access it in a manner similar to (but also incorporating more complex queries than)

```
collection.search(experiment="IFS-NEMO-9km", variable="tas")
```

After a successful demonstration of the BORGES data system, it can then be extended to provide interfaces also for other tools and languages in the community (e.g., ParaView[5] and other visualization frameworks). These should, in particular, allow for access also across time scales, i.e., when data retrieval might be very slow such as when data is retrieved from tape storage (ExU2 Asynchronous access).

Post-processing and storing the respective data needs to happen in the HPC environment in which the data producing model, ICON, is running to ensure high throughput between the respective components. While on-site users, i.e. those with direct access to the HPC system to which the data store is attached, can benefit from

3 Schulzweida, U., 2021: CDO User Guide (Version 2.0.0). https://doi.org/10.5281/zenodo.5614769

4 Official website: https://intake.readthedocs.io

5 Official website: https://www.paraview.org

this high-throughput and more direct access, off-site users will need to use a dedicated interface via the internet (**RqU5 External interface**). In line with Task 6.2 the WFLOW model[6] will receive the necessary ICON data as input via this external interface.

2.2 Core data

As the BORGES framework is designed specifically with the ICON model in mind, the core data to be processed and stored in it focus on the ICON state vector. Conceptually, this encompasses all physical variables handled by the core model – "prognostic" variables –, though might also include intermediate quantities – "diagnostic" variables (at the current model state these are only loosely differentiated). By the design of the model these are on a 2-dimensional grid, but might vary over (discrete) time (steps) as well as along height levels. The same holds for data of this form which have been further processed, e.g., aggregated via time, interpolated to a different grid, etc., but which still are 2-dimensional grids. Overall, these potentially large data sets are

- Model input data such as an initial state to start the model from (**RqD1 Input**)
- Model output data, either raw or further processed (**RqD2 Output**)
- State vector snapshots for checkpointing (including technical variables needed for bit-identical reruns) (**RqD3 Checkpoints**)

Other data, such as experiment parameters or grid definitions, are handled as metadata (see next subsection). The conceptual relationship of these data sets is shown in Figure 2.

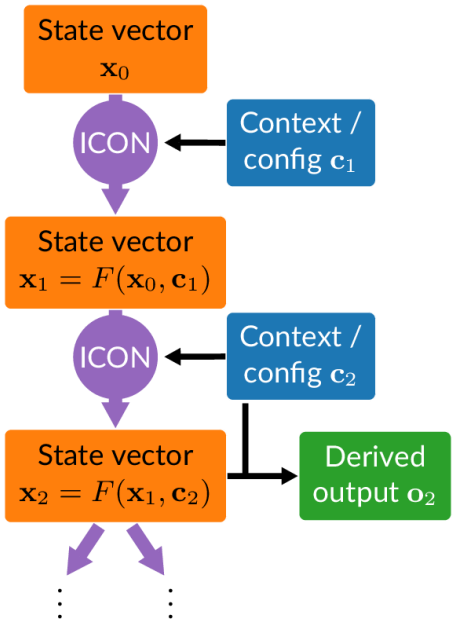

Figure 2 - Relation between ICON and the relevant data in BORGES.
ICON as a model transforms its physical state vector (including all prognostic variables) from one model time step to the next, according to a specific configuration and context. Also, further data can be derived from such a state vector, e.g. diagnostic variables, aggregations, etc. BORGES has to account for selected state vector snapshots (orange) and derived output (green) as potentially large core data sets. Also, it has to keep track of the relevant metadata of the context and configuration (blue) as well as the links between these data sets.

6 van Verseveld, W., M. Visser, H. Boisgontier, H. Bootsma, L. Bouaziz, J. Buitink, D. Eilander, and M. Hegnauer, 2022: Hydrological modeling framework in Julia. https://doi.org/10.5281/zenodo.5958258

Code: https://github.com/Deltares/Wflow.jl

The stream of core data is to be built on the GRIB2 format[7], which is also the format chosen as the primary data format in FDB. GRIB2 has specifically been designed and standardized to stream meteorological and climatological data. It is organized in messages or "fields" defining the data, its grid, and the data itself on a 2-dimensional grid. Following FDB, a GRIB message is the smallest indivisible unit in BORGES (**RqD4 Format**). However, to efficiently retrieve time series these might need to be split up, i.e. GRIB messages of sub-grids/sub-domains might be constructed from the full grids, to efficiently retrieve partial time series. This has been identified as a highly-used access pattern by climate researchers (**RqD5 Time series**). To efficiently fulfill this requirement FDB has to be adjusted accordingly. Once the overall framework has been established and allows for in-depth benchmarking, alternative data organizations other than FDB might be tested, in particular ways which include efficient chunking across several dimensions.

With the huge amounts of core data to be stored in BORGES, not all of the data can be stored on disks. In particular, the system should allow for storing data on tape archives, in an as seamless as possible way. Its back-end should be organized accordingly in a hierarchy – from fast SDD as caches, via disk for large, often-accessed data, to tape stores (**RqD6 Hierarchy of back-ends**). While data can be archived to tape explicitly by users, the system should also put an expiry date on data by default, unless users proactively archive their data (which could still rely on disk as long as it is regularly accessed) (**RqD7 Time-to-live**).

2.3 Metadata

Keeping track of experiment metadata such as model parameters and other input data are a crucial part of a sound data provenance strategy. Several (mostly individual) efforts have been undertaken to create databases for organizing such data at MPI-M. Unfortunately, such a database has to keep up with the changing target of ICON as a model under active development. So far, researchers organize their experiment metadata on their own in individual ways. For all core data processed and stored in BORGES a complete-as-possible set of metadata should be handled by the system, including (also compare Figure 2):

- General metadata (**RqM1 General**)
    - Creation time
    - Creating user
    - Core data size
- The process that produced the data (**RqM2 Process**)
    - Technical context: Machine, compiler, code version, binary, ...
    - Process (model) configuration: Initial state, parametrization, ...
- Links to other data in BORGES that were used in the process (**RqM3 Relations**)
- User-specific metadata guiding the exploration of the data stored (**RqM4 User-specific**)
    - Experiment name
    - Experiment version
    - Description
    - Tags
    - Access permissions

7 World Meteorological Organization (WMO): General Regularly distributed Information in Binary form: Edition 2, https://old.wmo.int/extranet/pages/prog/www/DPS/FM92-GRIB2-11-2003.pdf

In a later state, core data should also be versioned. In fact, if the relations between data sets are included in the system consistently, they make up a graph similar to versioning systems such as Git[8]. Adding proper version support, however, would require further features (ExM1 Versioning).

Consistency in meaning of parameters is a further aspect of experiment metadata. Here, the meteorological and climatological community has managed to establish and stick to a standardization procedure for decades, organized by the WMO. These mostly come in the form of CF-conventions[9] (e.g., used in NetCDF[10]) and standardized tables for the GRIB format. As GRIB2 is the format chosen for BORGES, this standard will be locally extended (i.e., add MPI-M-specific definitions) and needs to be checked when data is stored via the data management framework (**RqM5 Parameter consistency**).

2.4 General requirements

Additionally, general requirements for BORGES have been identified. With the currently already high data volume it has to efficiently handle streams of high volume and velocity (**RqG1 High throughput**) with reasonable access times for the most important data to users (**RqG2 Low latency**). Ideally, the overall structure enables the system to be easily adjusted for even larger data sizes (**RqG3 Scalable**). With limited resources available and to make these rather strong requirements reachable, data reduction and compression are indispensable on the medium run (**RqG4 Storage-efficient**). The system should thereby try to make use of the existing hardware infrastructure and not demand additional or specialized resources (though, of course, several are tested in Task 6.4, but can not be taken into account yet for a more medium-term operational system) (**RqG5 Use existing hardware**).

At this stage, these requirements are still rather vague; their further specification and, if possible, a quantification will be done in the exploratory process. As exploring the possibilities and limits of these dimensions are a major focus in this project, the system has to be flexible enough for such an endeavor. This, however, needs to be balanced with the target of a fully usable prototype successfully serving the needs of a reasonably-sized user base. Accordingly, from the very beginning high value needs to be put on the separation of concerns inside the system and on the challenge to clearly define a strict interface between the inside of the system (storage, data organization, housekeeping) and the user. This includes a code base separate from the core ICON model and, most importantly, a high modularization (**RqG6 Modularization**). In order to gather insights into several approaches regarding data organization and storage in the BORGES data system, it has to be monitored and subjected to constant performance evaluation and benchmarks (**RqG7 Monitoring**).

8 Official website: https://git-scm.com

9 Hassell, D., J. Gregory, J. Blower, B.N. Lawrence, and K.E. Taylor, 2017.: A data model of the Climate and Forecast metadata conventions (CF-1.6) with a software implementation (cf-python v2.1), Geosci. Model Dev., 10, 4619–4646, https://doi.org/10.5194/gmd-10-4619-2017

10 Official website: https://www.unidata.ucar.edu/software/netcdf

3 Architecture

This section introduces the architecture envisioned for the BORGES data system under the requirements identified in the previous section. It thereby tries to create clear boundaries between the individual components and libraries involved. This is in particular important as this project emphasizes the exploration and evaluation of solutions to the requirements. Thus, the end-product is a dynamic, moving target which is prone to changes along the implementation and constant evaluation. Also, actual access and usage patterns will need to be monitored once a prototype has been established and the system to be adjusted accordingly. All components need to be abstracted from their particular implementation in a careful manner such that they can be altered and substituted for by alternative solutions if needed. Overall, care has been taken in the design of the architecture so far in order not to take one-way decisions in this early stage but have sol grounds to base on with sufficient flexibility for the implementation.

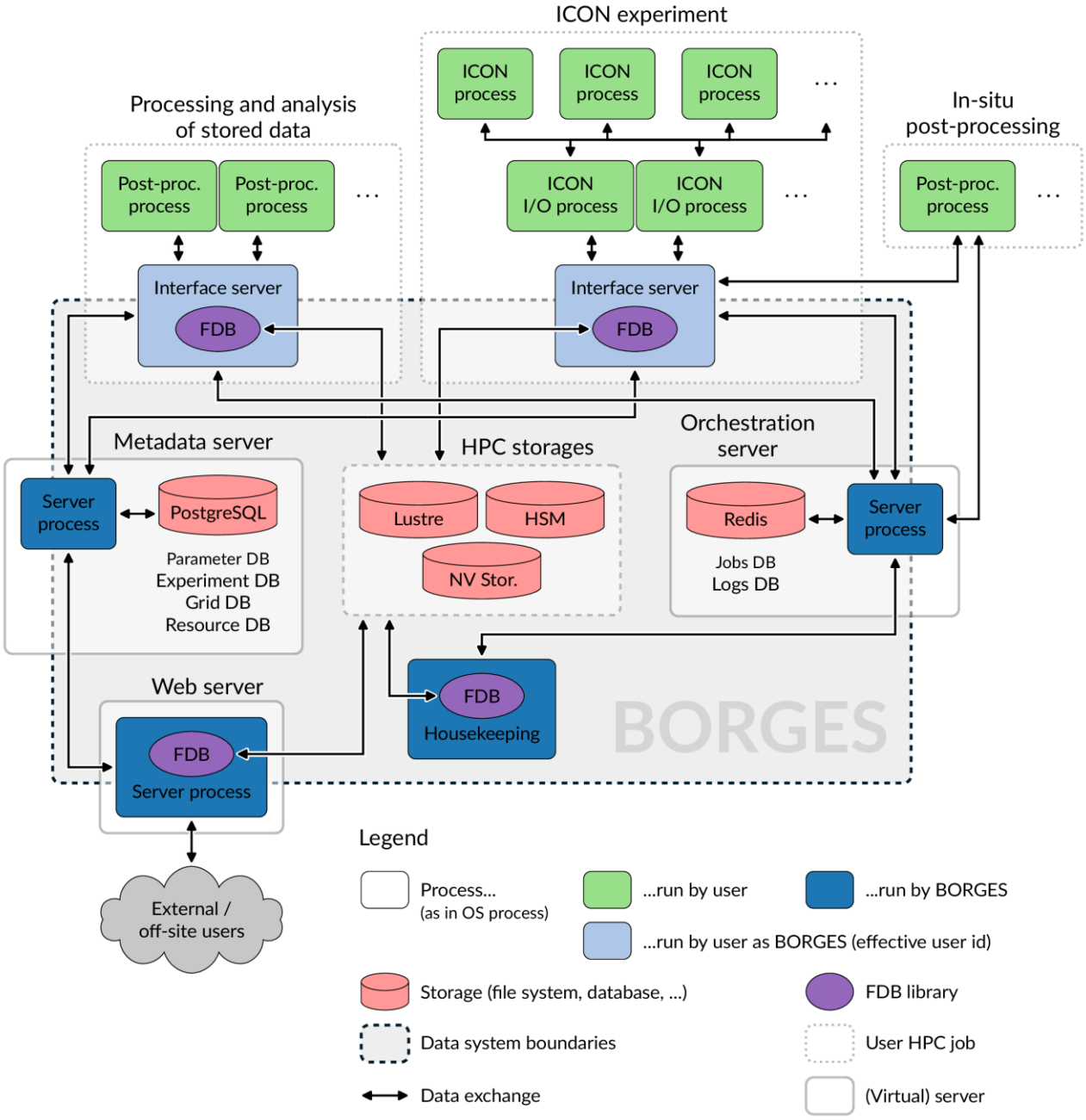

Figure 3 - Component view of the data system architecture.
BORGES consists of a dynamically (light blue) and statically (dark blue) started processes across machines.

An overview of the architecture is given in Figure 3 from a component perspective focusing on the involved processes – programs under execution. Each of these processes runs on one machine in the extended HPC environment (on the HPC itself or machines and virtual servers directly in connection to it). When several processes belong to an application or HPC job, they have been bundled with a dashed line. Whereas the gray shaded box includes all processes that are part of the BORGES data system, some of them can be accessed by the user and thus constitute the interface – no other exchange of data is possible, in particular, no direct access from the user to the underlying data (e.g. files). Thus, the system clearly separates user processes from the system's internals via several defined interfaces, which is a necessary step to ensure consistency of data inside system across its different parts. Additionally, this allows the seamless monitoring and improvement of internal mechanisms without affecting the user directly.

The user thereby has mainly two rules. For one, they run the ICON model (potentially highly modified for the individual researcher's needs) as primary data source accessing the system in a writing manner. The generated data enter the data system via an interface server process striving to use the HPC hardware in an optimized way to ensure high-throughput into and via the system to the underlying storage. Such process thus is effectively part of both, the user's HPC job and the BORGES system, deriving from a binary separate from the model and running with a dedicated user effectively (see details below). On the other hand, users access data in a reading manner; in fact, especially for large experiments the reading volume and number of reading users is expected to far exceed the writing access patterns. As there exists a plethora of different analysis workflows and technologies used, BORGES aims to provide a wide range of possible direct usage. In particular, the format of retrieved data should range from "raw" GRIB messages (e.g. when retrieving initial or checkpointing data for the model or for streaming to large post-processing applications) to more convenient formats such as xarray[11] for people used to a Python or Julia-based workflow. Access thereby is eased by dedicated small convenient interface libraries for the most common programming languages and frameworks in use. Similar, a channel for accessing the data from outside the HPC environment itself is crucial to enable sharing some of the data across institutes (this will be tested with the WFLOW model as a first "prototype user").

Beyond core data access, users (or their processes) need to be able to access metadata. In a writing manner this only affects registering experiments and their metadata and creating new, individualized parameters. Reading access, on the other hand, allows for interactive exploration of the data catalog as well as the standardized parameters. So far, these access patterns are planed to be served via the dedicated libraries as with the core data, but might later be extended by further convenient interfaces, e.g. an interactive web front-end.

In its data flow BORGES tries to find an efficient compromise between a centralized database architecture and computing jobs distributed on the HPC cluster. It is centralized in the sense that access to the system is only possible via the defined interfaces (e.g., a shared library would not fulfill this requirement). However, the processes for this interface are distributed among user HPC jobs as well as classical service processes on virtual or dedicated hardware servers.

Though not differentiated in Figure 3, the communication between processes varies among different components – large streams of core data and smaller but more frequent exchange of metadata and requests. The protocols for these communication flows try to follow appropriate standards and is differentiated between internal communication (in the system as well as where high throughput and low latency is required) and external or low throughput but convenient access. For the former BSON[12] or Protobuf[13] will be used for requests (both being a standard in highly efficient inter-process communication across machine architectures and programming languages) and GRIB for inter-process core data exchange. The latter includes all relevant metadata; where necessary extending the "local extension" section 2 of the GRIB standard. For more convenient access (but with lower throughout and more overhead) the system follows common standards in providing REST APIs over HTTP, which can easily be accessed across a variety of programming languages and frameworks.

To ensure efficiency (in terms of memory and processing) as well as robustness, most of these components are developed in typed compiled languages, specifically, C or C++ as standard in HPC software engineering; only relying on standard and common libraries as far as possible. The deployment of the data system will be based on the Spack[14] framework. This allows to define compilation and installation of the respective components in a system-independent manner. However, many of the components will need to be manually

11 Official website: https://xarray.pydata.org

12 Official website: https://bsonspec.org

13 Official website: https://developers.google.com/protocol-buffers

14 Official website: https://spack.readthedocs.io

configured for the specific system. This allows the further evaluation of the data system on different systems as part of Task 6.4 “Complex Weather and Climate workflows at pre-exascale”.

4 Components

This section describes the components of the BORGES architecture as presented in Figure 3 in more detail. While some of these constitute the basis (ICON, FDB), others are still to be more defined in the course of their implementation.

4.1 ICON

The ICON (ICOsahedral Nonhydrostatic) model is developed in collaboration between MPI- M and the German Weather Service (DWD), and includes several other working groups like the German Climate Computing Center (DKRZ), the Karlsruhe Institute of Technology (KIT), the Swiss Federal Institute of Technology in Zürich (ETH) and the Swiss National Supercomputing Centre (CSCS). Its forecasts are used in more than 30 national weather services as lateral boundary conditions for limited-area forecasts. In BORGES, ICON constitutes the major source of data and is the main purpose target for the system's design, in which all necessary specialization is guided by ICON's current state and planned development.

The ICON model is not only used for weather forecast by the weather services but also for climate research and to investigate climate change, its variability and predictability. Recently, two prominent threads of ICON models in the Max-Planck institute for meteorology have caught attention and momentum.

ICON is developed in four branches, atmosphere in the Earth system (ICON-AES), numerical weather prediction (ICON-NWP), ocean in the Earth system (ICON-OES) and land in the Earth system (ICON-LES). And the final ICON version integrates the stable components of all branches[15]. These ICON branches are connected in a unified coupled branch, the ICON-ESM (earth system model) branch.

All the parts in ICON-ESM interchange information to create a complete earth system model, ICON-LES communicates with ICON-AES through the water flow, heat and momentum, and the carbon content. It further communicates with ICON-OES providing the data of water flow. ICON-OES then interchanges information with ICON-AES about the water flow (e.g. evaporation) as well as heat and momentum. And finally, ICON-AES interchanges information about carbon content with the ocean through a model, like the Hamburg Carbon Cycle Model (HAMOCC) which is an ocean bio-geochemistry model.

Lately, efforts to make ICON more accessible to other users, resulted in the development of the toolbox "make experiments!" (mkexp). This is a compilation of tools needed when preparing ICON experiments using the MPI-M ESM in a single configuration file. This allows even less advanced users to make complex ICON runs like coupled CMIP6-style runs.

4.2 FDB5

The Field DataBase (FDB) developed at ECMWF is a domain-specific object-store designed for fast archival and retrieval of meteorological and climatological data (both observations and model outputs). With a decades-long history, the library is currently in its fifth generation (FDB5) and has been fully overhauled from the previous one. In BORGES, the FDB constitutes the major sink for stream data with several back-ends for actual storage.

The FDB is both a software library and a service which has (in various forms) been in use for many years and have undergone substantial evolution. As a library, it provides the final output stage for the I/O stack used by the Integrated Forecasting System (IFS, ECMWF's forecast model), as well as controlling meteorological data output for other tools. In current use it defines data structures and usage patterns to store meteorological data on a high-performance parallel filesystem, currently Lustre.

As a service at ECMWF, the FDB acts as a layer within the MARS ecosystem[16], storing recently generated meteorological data as an in-HPC hot cache of data. This data can be directly accessed using the FDB by

15 Prill, F., D. Reinert, D. Rieger, and G. Zängl, 2020: ICON Tutorial: Working with the ICON model. Deutscher Wetterdienst (DWD), https://www.dwd.de/EN/ourservices/nwv_icon_tutorial/pdf_volume/icon_tutorial2020_en.pdf

16 Official website: https://confluence.ecmwf.int/display/UDOC/MARS+user+documentation

other components within operational and research weather forecasting pipelines, but it is also served by instances of the MARS server software sitting on nodes within the HPC facility, to make data within this cache available efficiently to clients outside of the HPC facility.

In accordance with ECMWF's outlook on data handling and curation, all write and read operations, and data exploration are metadata driven, using scientifically meaningful metadata. Operations are both driven by and validated according to a well defined schema.

In the context of the ACROSS project, we aim to extend the functionality of this object store and to decrease our coupling to specific hardware solutions. In particular, we aim to reduce our dependence on POSIX filesystems and any other globally-namespaced storage system. We also wish to exploit the development of upcoming data stores. The current version of the FDB software is released publicly[17].

In existing operational use, the FDB library is able to act on globally visible parallel storage resources (currently a Lustre filesystem). As the volume of data produced and consumed continues to grow, this poses extreme requirements on the capacity of the parallel filesystem — especially in terms of metadata handling. We can neither assume that the performance of globally-namespaced parallel filesystems will scale with our data needs, nor that they would be financially sensible propositions if they did. For this reason we have decoupled the internal FDB architecture in two main components: the Catalogue, responsible for metadata management and data indexing, and the Store, responsible for bulk data archival.

In the context of ACROSS we are going to consider a hybrid deployment on hierarchical data stores to benefit from high-IOPS offered by NVMe- and PMEM-based products for the FDB Catalogue, and large capacity and bandwidth of massive parallel file systems for the FDB Store.

4.2.1 Configuring FDB

This project will investigate the impact of drastically different data access patterns on read and write. In fact climatological simulations are generated in time-steps and accessed in time-series, thus requiring a data transposition step. Therefore, the impact of different data organization, for now different FDB schemata, to assess write and read performances in such different scenarios will be evaluated.

A first proposition for a schema serving the basic requirements of storing ICON output is:

```
[ type, expid, expver
   [ category, parameter
      [ level, date, time ]]]
```

This bundles high-level keys (type such as "model output" or "initial data", experiment id, and version) into individual folders. To keep the number of generated files low while allowing parallel output from several ICON I/O servers (which yield data from the same time step but for different variables), variables (per GRIB standard broadly defined by category and parameter) are bundled into individual files. In each file the written GRIB messages are indexed by their further dimension, i.e. level, if applicable, and time (here separated as date and time of day). This way, reading a time series for a single variable involves accessing only one individual file. With an extension of sub-grid access (see below) and with the help of a grid database, reading time-series on sub-grids can further be accelerated as the relevant positions inside the file can be directly targeted. Also, this way, the schema strives for fewer but larger files in order to make better use of Lustre's performance capabilities.

The relevant keys are as specifically tailored for the data used and generated by the ICON model as possible, and only as general as really needed (e.g., ICON has a fixed way of defining height levels). Note, however, that this schema in this forms focuses on ICON output in the first stage and will be adjusted to properly separate input and checkpointing data.

17 Code repository: https://github.com/ecmwf/fdb

With the FDB as an application-level data store, moving data between back-ends has to be organized by a housekeeping component. This process will, in particular, move and distribute data among FDB configurations on a Lustre filesystem and on the hierarchical HSM storage (including cache disks and a large tape archive), potentially with additional more specialized storage hardware. Keeping track of the actual FDB storage instance/schema used is thereby a responsibility of the metadata server in communication with the interface or web server reading or writing the stream. Thereby, all accesses to FDB are encapsulated as part of processes which only allow for an interface abstract from the actual data storage library and model internally.

4.2.2 Adjustments to FDB

In the context of the ACROSS project, we will develop and assess the benefit of multiple FDB back-ends based on traditional POSIX file systems, high-performance object stores such as Intel DAOS and cloud-friendly solutions (i.e. Catalogue on PostgreSQL and Store on Ceph/Rados). In particular, the development of a back-end for the HSM data store – the current standard at MPI-M for tape archival –, is a high priority for the BORGES data system. At ECMWF, the effort on tape archival is currently focused on MARS, which re-organizes data for optimizing data access from tapes.

Depending on benchmarking results, most probably sub-grid access including indexing of the data inside each GRIB message will need to be implemented into FDB. Currently, only full messages (i.e. global grids) can be retrieved which is likely to constitute a bottle neck in efficient time-series oriented storage pattern and access. ECWMF is currently working on such sub-grid access, however, the GRIB standard, in particular the compression methods used, might make byte-addressability difficult and more elaborate chunking mechanism might need to be implemented. With further attempts to establish a chunking mechanism for efficient compression and fast retrieval of data slices across dimensions, alternatives to FDB might be needed to be evaluated. For very large messages and to distribute I/O work efficiently among the full process chain from source (model) to sink (disk) the actually stream of individual messages might need to be implemented, i.e. to stream partial messages and not have full messages in memory.

Further, and more technically, as the FDB library is encapsulated by dedicated interface processes, it is most desirable to keep it as well as its full configuration inside the respective binary, i.e. to statically link it and control the configuration via a dedicated interface rather than via files. In the current version, this also requires small adjustments to the way the library is built and interfaced.

4.3 Interface servers

The BORGES interface server processes are a crucial component in the data system as they are the middleware between high-throughput readers and writers and the data system itself. As such they constitute the distributed aspect of the BORGES system and its interface as each respective reading and/or writing HPC job starts one of these servers to access the data system. All data exchange with BORGES then passes the respective interface server.

Each interface server process thereby is tasked with ensuring consistency in all data entering the BORGES system, directly communicating with the metadata server and the storage back-end via the FDB library (as part of each process). In that, it is the canonical point for adding further compression and data reduction, which is to be explored in the course of the project. Additionally, the interface server is a pivot for data format conversion when being connected to user convenience libraries (e.g. converting from and into xarray format).

As one of the first components in the data chain after the actual data source, the model, high throughput is particularly important for the communication with the interface server. Therefore, the interface makes use of Remote Direct Memory Access (RDMA) capabilities of the MPI (Message Passing Interface) standard, which should follow optimal hardware usage when implemented correctly by the HPC vendor. As a binary separate from the core model binary nevertheless taking part in the respective MPI communication world, it requires the Dynamic Process Management feature of MPI version 2. This should be available on all modern HPC clusters and is also useful to efficiently connect different interface servers to each other (e.g., when later in-situ post-processing needs to be added to a running simulation).

Though running as part of the user's HPC job, interface server instances run as a dedicated BORGES user. With a code base separate from ICON and the post-processing applications with which it interfaces, the interface server processes derive from a separate binary. This is controlled by the system administrators to keep the interface clean and out of users' reach. With the SUID bit set[18] and owned by the "BORGES" user, it runs under the user's real id, but with "BORGES" as effective user. Thereby, all file access is done in the "BORGES" user name; no permission for access is directly granted to the real user. This constitutes an essential pillar in making sure the data consistency is controlled by the system itself only.

Additionally, each interface server serves as a data broker or multi-caster providing model output as well as data stored inside or currently streamed into the BORGES data system to further post-processing processes. These can range from first preliminary post-processing for further aggregation or data reduction to sophisticated analyses as defined by the researchers. For finding the respective interface server their instances register themselves in the orchestration server (see below).

Currently, the output stream from several computing nodes tasked with simulation on particular sub-domains of the full model area are combined in dedicated ICON I/O server processes. In a later stage these tasks could also be fulfilled by the BORGES interface servers, which could benefit from being able to directly store streams with the same sub-domain distribution.

4.4 Metadata server

The metadata server serves as the system's single point of authority regarding metadata on the parameter and grid definitions, experiments and other relations between the actual core data sets in the large storage back-ends. For that, it incorporates a classical relational database, namely PostgreSQL[19]. To also include data access logic, this database is never accessed directly by processes outside the metadata server but via a dedicated server process.

The Parameter DB part of the server stores the definition and metadata (description, unit, etc.) of parameters as per the GRIB format. This includes standard parameters as defined across institutions ("centers") by the WMO, the MPI-M-wide locally defined parameters, as well as user-specific definitions.

To organize data among simulations and researchers, the Experiment DB constitutes a register of experiments (i.e. model simulations) including their parametrization, links to initial data or checkpoints, and user-defined metadata to target a high level of data provenance. Here, the data system includes a database with a schema as flexible as possible with defined and registered experiment IDs. Output of these experiments will then be linked to this database as long as they are also stored via the data system. Every experiment output stored will need to be equipped with a unique experiment ID (either chosen by the user or automatically generated; registered in the database), re-run experiments (e.g. with a new ICON version) will use the same ID but with a different experiment version. The flexible schema makes use of the JSON storage and indexing capability of PostgreSQL. Specifically, the namelists as used for model configuration in ICON can be represented in a structured JSON form. These can be searched in the database, but the structure needs to be known by the searching user rather than the database itself. This allows to represent the model configuration across different model versions without altering the schema of the database itself for every configuration structure change.

Especially when reading sub-grids the Grid DB will come in handy as a reference for the most common grids used by ICON experiments. Here, the structure of the particular grid will be stored for fast mapping from latitude-longitude coordinates to the respective indices of grid points as a hint on where to find respective data in GRIB messages with that grid fast. Also, this reduces the metadata overhead of storing non-standard grids in GRIB message headers (which are repeated for all messages on this grid, i.e. for all time steps and height levels).

18 https://en.wikipedia.org/wiki/Setuid

19 Official website: https://www.postgresql.org

In a later stage, resource accounting (in particular storage use across users and projects for the different back-ends and caches) will be done in a dedicated Resource DB.

4.5 Orchestration server

The orchestration server has mainly two tasks. For one, it is the central point of authority on currently running interface servers enable later in-situ post-processing applications to find and connect to the desired experiment output stream easily. Further, it is the main source for all log and monitoring data. In the prototyping and evaluation stage, it will store measurements from the respective components on their usage and performance as well as logging events.

Pragmatically, both tasks use a Redis[20] database for their respective data. As a high-performant in-memory store Redis is a common choice for fast unstructured or log data streams. These streams are accepted by a dedicated server process and are accessed only by the interface servers for attaching to each other where needed as well as by system administrator for monitoring and diagnostics.

4.6 Housekeeping

As a central element of the core of the BORGES data system, the housekeeping process has the following tasks:

- Check consistency and, if necessary, clean residuals in the involved databases and catalogs
- Remove outdated data, e.g. after the time-to-live on disk has ceased
- Move data between back-ends, either after explicit user requests (e.g. to archive data) or to efficiently cache accessed data
- Depending on the actual design of the web/external interface, it might also serve as means for asynchronously retrieving requested data from tape, similar to a worker in Polytope (see below)

It is run by the system in a regular interval and can be run on a specific (virtual) server or directly as a classical HPC process (to ensure high throughput to the storage back-ends, e.g. via a self-rescheduling SLURM job). The orchestration server has the control of starting and observing this process. As such it accesses the underlying data as the same dedicated data system user as the DB server processes. Depending on further design and requirements, these tasks might be undertaken by more than one housekeeping process.

4.7 Web/external server

As an interface to BORGES from outside its HPC environment and institutional intranet, the web server allows external users to access data stored in the system. This will follow the standard HTTP protocol with a REST API which is most common for these kinds of access patterns. For regular web access, in particular, the web connection is the main bottleneck rather than such a server process itself. This can later be extended to efficiently make use of existing high-speed connections between institutions to provide access to the data system in a manner as if running on the same institution though the actual data is produced and/or stored in another.

A possible candidate for such a system is the Polytope system described below, whose suitability for BORGES will be evaluated in the course of the project.

4.7.1 Polytope as a potential component

In the context of the LEXIS EU Project, ECMWF has developed Polytope, a software system for management of curated meteorological and climatological data, exposing a RESTful API. This enable the exchange of weather and climate data among distributed data centers. Due to the sizes of data involved, downloading

20 Official website: https://redis.io

requests to Polytope will be asynchronous, returning a poll-able URL where the client may check the progress and finally retrieve their data.

Figure 4 shows the current technical design of Polytope. Each instance of Polytope may be directly connect to a MARS or FDB server, while all instances are federated, and thus can cooperate in order to efficiently retrieve and serve data to the users minimizing the impact on the operational archive. Each instance of Polytope may have an additional local FDB acting as Data Store for archival of user data and as a cache for data coming from MARS. Furthermore, in the local FDB instance we can archive post-processed versions of the original data, thus further reducing the time required to deliver data to the users.

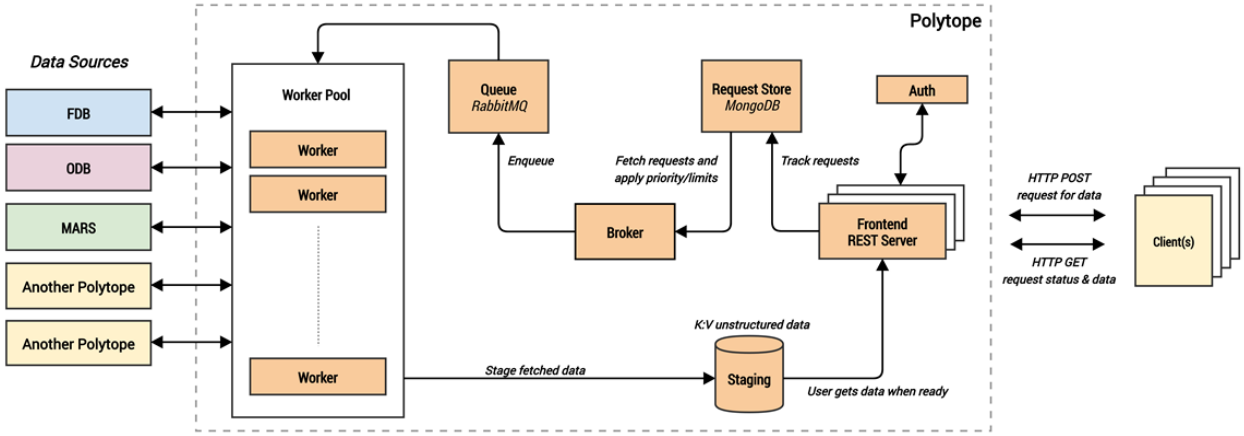

Figure 4 - Polytope architecture

Polytope can serve different collections of data (e.g. forecast data, climate data), that are represented as separate end-points. HTTP requests to each end-point will include authorization metadata in the header and data-specific metadata in the message content. Data will be indexed using multiple scientifically meaningful keys. Endpoints are provided to query collection contents and their data schemata. To simplify adoption and integration of Polytope, it includes a Python client, which can be executed as a CLI application or as a simplified library in Python scripts.

5 Results

This section will be updated in the second version of this deliverable (at project month M24) to include lessons learned in the implementation, benchmarks, and first experiences with the diverse ACROSS architectures.