A new CARTA controller for scalable deployment across regional centres

Angus Comrie Robert Simmonds Adrianna Pińska David Aikema

Inter-University Institute for Data Intensive Astronomy (IDIA)

Abstract

The Cube Analysis and Rendering Tool for Astronomy (CARTA) [1] is a modern visualisation tool with a client-server architecture, designed to support very large image data sizes.

The current implementation of CARTA allocates a single backend to each user. This works well on the desktop and in simple site deployments, but limits our ability to scale effectively to larger images in a high-performance computing environment, as the backend process can only reside on a single node. Launching a separate backend process per image allows us to distribute computation by allocating processes to multiple nodes, using heuristics based on the image size and available resources. Serving each image from an isolated process also limits the impact of hardware or software failures on an individual node.

An extension of this approach would also provide a secure and efficient method for serving files from different locations, such as SKA Regional Centres, allowing users to combine multiple data sources seamlessly in a single frontend environment. We present an outline of the architecture changes that have been implemented to support this model in a new CARTA controller software package, currently under development.

Introduction

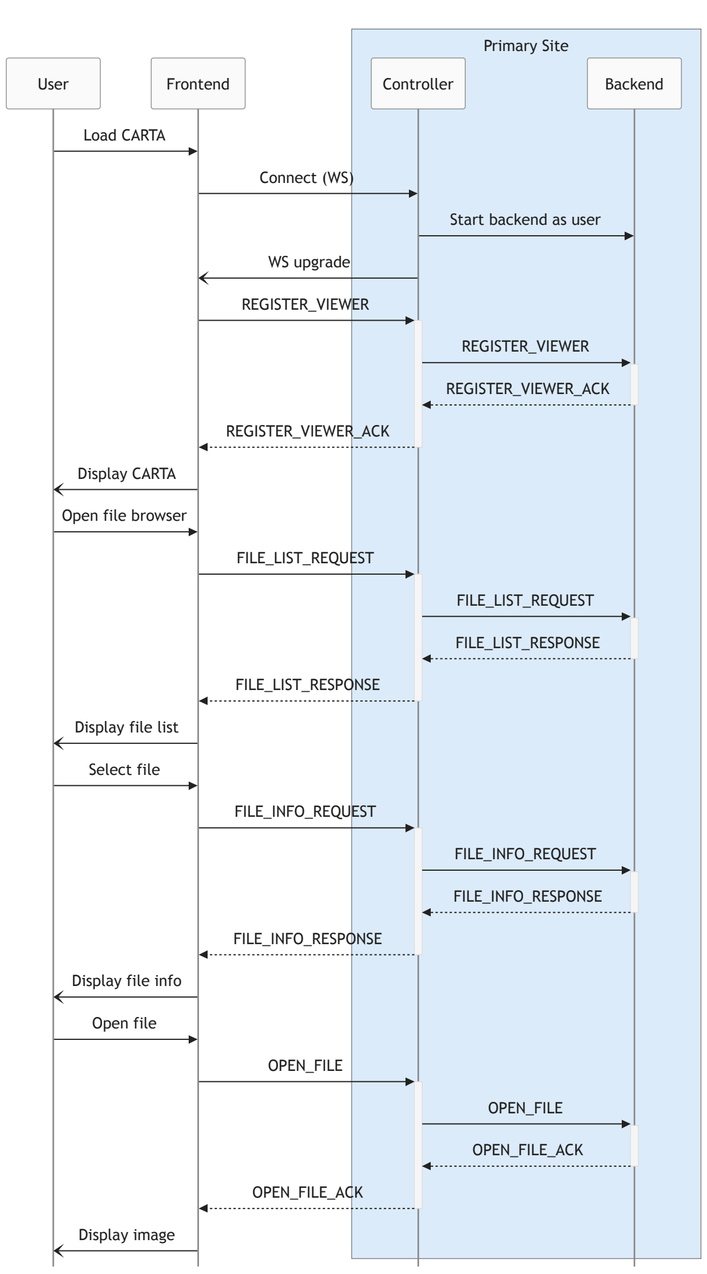


Figure 1. Connecting, browsing and opening a file with the current CARTA controller. On a user's first authenticated WebSocket connection, the controller starts a backend as that user, so that it can only read files that the user can access. The controller then proxies messages bidirectionally between the frontend and backend.

Current CARTA implementation

CARTA can be run by a single user on a desktop computer, or in a multi-user environment on a shared server or cluster. The current version of CARTA consists of multiple components:

Controller

- Written in TypeScript and executed in Node.js. Only used in multi-user deployments.
- Authenticates users, serves the frontend, spawns and manages backend processes, proxies the WebSocket connection between backend and frontend.
- Stores persistent user data in a MongoDB database; manages workspace sharing.
- Relays scripting requests to the backend.

Backend

- Written in C++. Runs on hardware as near to the data storage as possible. Slices, downsamples, and compresses image data to send to the frontend.
- One backend spawned per user; multiple sessions per backend; multiple images open per session.
- Basic HTTP server provides HTTP scripting API, serves the frontend, stores data (in flat files). No sharing features.

Frontend

- Web application written in TypeScript. Runs in the user's local browser.
- Communicates with the backend through a WebSocket connection; requests portions of image data on demand.
- Performs rendering tasks that don't require full-resolution data, such as rasterization, colour mapping, and contour rendering.

Multi-site considerations

Currently it isn't possible for a user to open images stored at multiple different sites in a single CARTA session.

- The user has to open multiple separate sessions, one per site.
- The browser windows can be displayed side by side, but the ability to integrate the images is limited.
  - To apply one region to all images, the user would have to export it from one session and import it into the others.
  - The scripting API could simplify repeating the same operations in multiple sessions.

New controller implementation

Changes to the software stack

- Node.js to Go
  - Go is a compiled language. This improves the controller's execution speed, and simplifies packaging and distribution.
  - TypeScript's ecosystem of third-party packages is vulnerable to security breaches. Go has a strong standard library; built-in concurrency.
- MongoDB to PostgreSQL
  - MongoDB was previously selected for its JSON support, but PostgreSQL now has a native JSON type.
  - Our data does not benefit from a non-relational database.
  - Easy to switch between different SQL implementations.

Separation into multiple services

- Monolithic service split into separate controller, process spawner, possibly authenticator.
- This reduces complexity and dependency requirements of each component.
- Optional components don't have to be included in every deployment.
  - Authentication methods could be split into plugins.
  - We plan to deploy a custom configuration on the desktop, replacing the backend's HTTP server. This would eliminate code duplication and simplify the backend. User data could be saved to SQLite.

Multi-backend mode

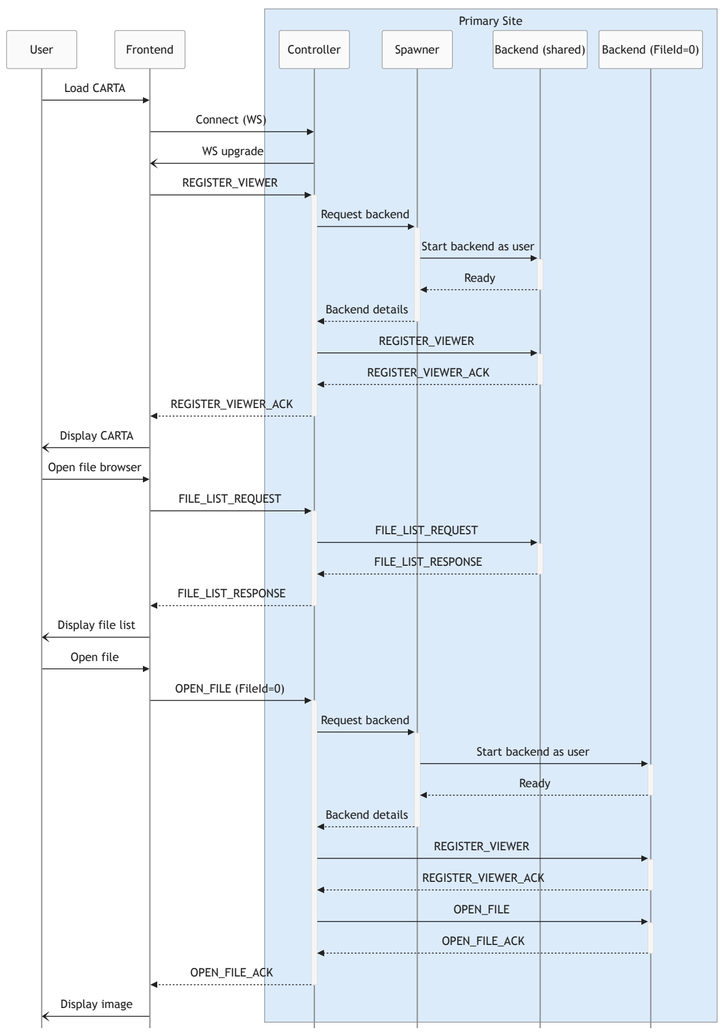


Figure 2. Connecting, browsing and opening a file with the Go controller in multi-backend mode. On viewer registration, the spawner service starts a shared backend for the file browser. Opening a file starts a dedicated backend for that file. The controller routes each message to the right backend by file ID.

Current approach: one backend per user

- Multiple sessions can connect; multiple images open per session; multiple concurrent reads per image.
- Problems:
  - HDF5 does not support concurrent reads from multiple threads, even of different files. The thread-safe build of the library uses a global lock that only allows access to one thread at a time.
  - Thread-safe use of shared objects adds implementation complexity and impacts speed.
  - An error produced by one image can crash multiple sessions.

In development: one backend per image

- If the same image is opened multiple times, in one session or multiple sessions, each copy gets a separate backend.
  - Controller spawns and destroys backends on demand, and routes requests from frontend to backends. Frontend doesn't manage backends directly.
  - Region coordinate transformations for spatial matching to move from backend into frontend, removing the need for backend to access metadata of other images.
  - An empty backend can be used to produce directory listings and file metadata for the file browser.
- Benefits:
  - True concurrent reads from HDF5 files.
  - Backend can progressively be refactored and simplified as unneeded functionality is removed.
  - CARTA is more resilient: one backend crash affects only one image.
  - A controller in a cluster environment can allocate backends to different nodes or different regions or zones within a site, depending on resource requirements and proximity of data storage. Data for the same image could be read from copies in multiple locations.
- Future extension: use multiple backends per image to eliminate all multi-threaded reads within one process.

Scripting interfaces

Current implementation

- The user can execute JavaScript snippets in the browser to control elements of the frontend.
  - This interface is intended for interactive use.
- A Python library focusing on non-interactive workflows is also available.
  - Python scripts can send serialized requests to an HTTP API in the backend, which relays them to the frontend.
  - In site deployments, the controller acts as a proxy between the script and the backend.

Planned changes

- The HTTP API will move to the controller, which will also be available on the desktop.
- The controller will relay scripting requests to the frontend through a WebSocket connection.
- This model is simpler to integrate with multi-backend mode and eventually with multiple sites.

Multi-site support

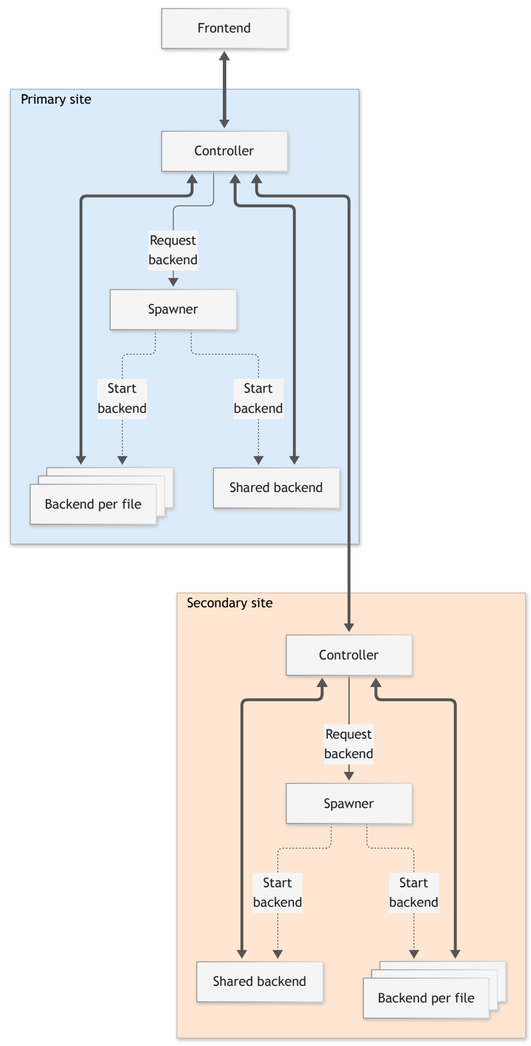


Figure 3. Components of a multi-site deployment. The primary controller connects to the secondary site's controller the first time it is needed, and reuses that connection afterwards. The secondary controller starts its own backends and becomes an additional step in the proxy chain. Thick lines are WebSocket connections.

Motivation

- Distributed systems such as the SKA Regional Centres (SRCs) hold data at multiple sites, and will need a way to serve CARTA users efficiently across them.
- Performance depends on keeping the backend as close to the data as possible: it reads full-resolution data, and sends only compressed, downsampled subsets to the frontend.

Proposed approach

- The user connects to a single primary controller, which connects to the controller at a secondary site when needed.
- Each secondary controller spawns backends at its own site, next to the data.
- SRCs are connected to each other by high-bandwidth links, far faster than a user's connection to a remote site. Coordinating data streams in a single controller is simpler than relying on the frontend to manage connections to multiple separate controllers.

Hybrid mode

- Users could connect their local CARTA installation to a remote controller, combining local and remote files in a single session.

Authentication and authorization

Current CARTA controller

- Supports PAM, LDAP, Google, and OIDC authentication.
- Designed to enable a user to access a single site through an interactive web browser session. A long-lived refresh token is used to retrieve short-lived access tokens, for use as bearer tokens.
- Support for non-interactive access to the scripting API was only added later, and currently is incompatible with OIDC authentication.

Revised approach

- Scripting API access is more deeply integrated, and supported by all authentication methods, including OIDC.
- A user will be able to generate a keypair and register the public key with the CARTA controller at a known endpoint. The controller will return a generated URL through which the user can log into their account and authorize the link. A non-interactive process will then be able to use the private key to request access tokens (for example, to connect to the scripting API).
- Registered public claims [2] (introduced by RFC 8693 [3]) will allow for the issuance of scope-limited tokens, as well as for token delegation.
- Token delegation and scoping can enable direct communication between multiple remote sites during certain tasks, eliminating the need to route all communications through the user. This shifts traffic from the user's potentially slower connection to the high-bandwidth links between large-scale computing facilities.

Acknowledgements

This poster is dedicated to the memory of Professor Rob Simmonds, who sadly passed away in July 2026. He was instrumental in the development of this work.

We acknowledge the use of the ilifu cloud computing facility – www.ilifu.ac.za, a partnership between the University of Cape Town, the University of the Western Cape, Stellenbosch University, Sol Plaatje University and the Cape Peninsula University of Technology. The ilifu facility is supported by contributions from the Inter-University Institute for Data Intensive Astronomy (IDIA – a partnership between the University of Cape Town, the University of Pretoria and the University of the Western Cape), the Computational Biology division at UCT and the Data Intensive Research Initiative of South Africa (DIRISA).

cartavis.org

References

[1] Angus Comrie, Kuo-Song Wang, Yu-Hsuan Hwang, Adrianna Pińska, Pamela Harris, Kuan-Chou Hou, Zhen-Kai Gao, David Aikema, Po-Sheng Huang, Marcin Sokolowski, Michaela van Zyl, and Rob Simmonds. Carta: The cube analysis and rendering tool for astronomy, June 2026.

[2] Internet Assigned Numbers Authority (IANA). JSON Web Token (JWT) Claims Registry. Web registry. Accessed: 2026-10-30.

[3] M. Jones, A. Nadalin, B. Campbell, J. Bradley, and C. Mortimore. OAuth 2.0 Token Exchange. RFC 8693, January 2020.

angus@idia.ac.za rob@idia.ac.za adrianna@idia.ac.za david@idia.ac.za