ORNL/LTR-2024/3680

# IRI Technology Landscape – A survey of re-usable components and methodologies

David M. Rogers

**January 2025**

**ORNL/LTR-2024/3680**

ORNL National Center for Computational Sciences

# IRI TECHNOLOGY LANDSCAPE – A SURVEY OF RE-USABLE COMPONENTS AND METHODOLOGIES

David M. Rogers

January 2025

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831
managed by
UT-BATTELLE LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

## ACKNOWLEDGMENTS

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

# 1. INTRODUCTION

This document describes technical implementation details on network access schemes connecting API-driven workflows to supercomputer centers. API-driven workflows are a central theme in connected computing, since they bring the “terminal”–“mainframe” access pattern present since the 1970s up to the task of interfacing with modern web browser technologies. Both security (HTTPS/TLS/IPSec/VPNs/public key cryptography/digital signatures) and network protocol stacks (HTTP-REST APIs, tokens, gRPC, SRTP) have evolved to the point where implementing API-driven workflows is possible using stable, secure off-the-shelf software.

There are three different types of access patterns covered: remote procedure calls, dataset-based file transfer, and P2P transfers (data streaming, in-memory/online, fast file transfer).

Security principles are covered first. Next, for each access pattern, an example implementation is provided using libraries implemented in publicly available, open-source software. The code recipes referenced by this document can thus serve as a starting point for building any type of IRI workflow involving the patterns above.

It is expected that further case studies will need to be published as individual technologies are evaluated, adapted, and run. This will be necessary because application-specific and site-specific configuration will always need to be performed by building software layers on top of the off-the-shelf references provided here. This adaptation process can be costly and difficult, so there is value in analyzing this step.

The end of this document briefly describes several IRI case studies which demonstrate early applications using the patterns above.

## 2. SECURITY PRINCIPLES

Three basic security principles are worth defining here:

- action attribution - Every action run by a server should be attributable to a known user. The server should also have some degree of certainty that the user wanted this action to take place.

APIs secured with user-specific tokens can attribute actions to those users. We assume HTTPS/TLS is used so that tokens are not intercepted over the network. However, there are several circumstances where a token could be sent when the user does not want the action to take place: i) if counterfit tokens can be generated, ii) if any system that stores tokens is misconfigured or compromised. Token expirations and renewals are commonly used to guard against these. There is also another solution, using biscuits – a type of token allowing the user to issue derived tokens with user-defined checks (e.g. limited to a certain group level, time, and client), in combination with client certificates – uniquely identifying the client on each connection.

- transparency/accountability - Every action run by a server should be logged, along with the chain of trust authenticating the client and authorizing that action.

This involves consistently utilizing logging servers, along with service monitoring (to ensure logs are being produced).

- policy enforcement - Every incoming connection to a network center should have some means of center-defined access control.

A policy-enforcement point is a point in the network protocol stack at which a grant/deny decision is made for a given request. For SSH, this involves checking the user's password or validating the user posesses a private key. For REST-APIs, this can involve checking a user token or validating the user posesses a private key. For direct network connections, this can involve a variety of mechanisms, including source IP and port checking, IPSec/VPN tunnels, and/or TLS/DTLS.

Example source code for REST-API access following all three principles above is available in many places:

- The certified package
- FirecREST's use of the Kong API gateway
- AWS Documentation on Security Credentials
- gRPC Authentication Guide / Google Cloud Auth Guide
- globus Authentication Reference
- Bluesky Tiled Security Guide

## 3. REMOTE PROCEDURE CALLS

This access pattern is synonymous with what is usually meant by REST-API access. Here, a server performs some action in response to a network request. This pattern is typically used to represent database records in a "REpresentational State Transfer" (REST) way. For a strong understanding of REST principles, seek to understand its resource naming conventions.

HPC centers might mis-interpret remote procedures to be batch jobs run with full user-level shell permissions on a compute cluster. However, this actually the exception, rather than the norm for APIs, since APIs provide much more fine-grained control of user intent and server actions.

With that in mind, there are actually very different security concerns that apply to web-enabled APIs. Foremost among those are cross-site scripting attacks. Here, a user accesses a legitimate API call to insert a maliciously crafted value (e.g. HTML or SQL code) into the database. Later, when the server accesses that value for the same or another user, it results in undefined behavior that generally produces unintended consequences. Long experience with Web 2.0 has shown that these kinds of attacks are almost always traceable to a type error – for example a string inserted directly into source code, or an invalid value that triggers a mis-handled exception. Modern applications employ HTML-quoting and standardized parsing, notably JSON with input validators like JSONSchema/pydantic, and ORM (rails / SQLAlchemy). Modern web-servers stop all processing on exceptions and return an HTTP-500 responses. Nevertheless, it is up to the client-request handling code to use thread-safe functions and validate all aspects of a request up-front to ensure that exceptions do not leave the server in an inconsistent state.

A simple example of REST-code interfacing to a database using SQL-ORM is present in the FastAPI user docs and there are many resources on line. A full-stack example of a user-management system with containerized databses and OAuth2 authentication is available as well ref.

Open-Source REST-APIs for jobs on HPC include:

- psik-api

Provides a REST interface to compute jobs and input/output files associated with those jobs. It is also able to make callbacks when jobs change state (queued/active/completed/failed). Uses client public-key certificates for authentication and biscuit-tokens for authorization.

- FireCREST

Uses OpenID-connect tokens for client authorization.

- Tapis

Uses password-based authentication to tacc.tapis.io to obtain tokens for use with the Tapis API. Integration with AWS S3 storage is enabled by creating access credentials for tapis to access AWS S3 storage on a user's behalf. Integration with Globus Transfer is similarly arranged via creating access credentials for tapis to access Globus endpoints on the user's behalf.

Authorization in TAPIS is provided by a centralized "Security Kernel" microservice that associates JSON web tokens (JWT)-s with role-based permission graphs.

- Cromwell

Provides a REST API specifically for the workflow description language (WDL) that supports job submission to many compute backends (Google Cloud, AWS, SLURM, SGE, LSF, HTCondor). This is heavily used by the bioinformatics community and thus has strong support for containerized workflows built on conda environments.

Cromwell does not isolate users from one another, effectively mapping all authenticated users to the same system user ref.

Closed-Source REST-APIs for jobs on HPC:

- Superfacility API (note an open source client is provided)
  - command-line callbacks are now available in beta.
  - the server code is not available even in binary form
- Globus Compute
  - the server can be hosted locally, but globus maintains control over job submission and client authentication (which may be subject to a license subscription)

## 4. DATASET-BASED FILE TRANSFER

This access pattern is used to serve large datasets, allowing users to download/upload specific subsets / files on-demand. It is usually used as part of a workflow that expects smaller (<1 GB) files, and/or user interaction (e.g. uploads through a web browser).

These require a functioning client/server pair.

Open-Source cloud file upload/download:

- SFTP (scp/rsync)

Note: SFTP's UNIX access model (directory hierarchy with user/group/world:execute/read/write permissions) is widely considered unsuitable for cloud access because of 1) lack of accountability / tracking changes, and 2) inability to dynamically scope file access permissions (e.g. multi-group/role access with tokens)

- Rclone – extremely versatile, MIT-licensed software used by OSC's Open OnDemand
- XRootD – supports access control lists and POSIX access ref
- Plan9 server – NFS-like protocol for mounting remote filesystem via 9mount (UNIX-based access model)
- Rucio file transfer service
- Academic Torrents
- Named Data Networking Platform (trusted publisher model)

Closed-Source cloud file upload/download:

- AWS File SDK for Python
- Google Cloud Storage Utility
- Globus File transfer

## 5. P2P TRANSFERS / DATA STREAMING

This case covers data moving between a set of client and a set of server machines that is specific to a given data processing activity – such as real-time camera/particle detector feeds or modeling/simulation parameters like AI/ML inference parameters or experimental geometry information e.g. computer-aided designs for process engineering models. It is distinguished from dataset-based file transfer in that the data scope and lifetime are associated to a given experiment. Note, however, that there is some overlap, since the outputs from an experiment may be best served by dataset methods, and "offline" inputs for a compute job are traditionally manually prepared based on stored datasets.

The WebRTC protocol was specifically designed to handle secure, P2P data streaming in the modern, often connectivity-challenged web. It is deployed widely as the basis for audio and video streaming to web browsers.

HTTP file upload:

- python/fastAPI server
- python/httpx client
- python/aiohttp client

HTTP file download:

- python/fastAPI server
- python/httpx client
- python/aiohttp client

HTTP WebSockets:

- python/fastAPI server
- python/aiohttp client

Note that Psik-api implements the HTTP file upload and download protocols with logging, biscuit-authorization, and public-key client authentication.

- WebRTC: peerbeam, gfile

### 5.1 NETWORK SECURITY ROADBLOCKS

Network security administrators create firewalls and IP traffic network addressing and routing roadblocks in order to protect unsecured services running on internal computers from receiving traffic they do not expect to handle. However, when a user runs an application that expects incoming network traffic, the network administrator needs to provide a means to allow this traffic and the user needs to provide a means to both log all access and secure their service against all unintended use. Many misunderstandings can appear at this step, since the simplest solution for security is to limit all access and the simplest solution for the user is to enable all access. Security is a shared responsibility, and operating network services requires transparency and accountability in both directions. Users need to be made aware of network-level access control mechanisms, and network administrators need to be made aware of logging and security policy enforcement methods used by user-level services.

Given this state of affairs, we now point out a list of work-arounds used in trial and demonstration phases of network services.

- User-level IP forwarding tunnels - these provide traffic forwarding between two computers who are not allowed to pass traffic directly
  - SSH port forwarding (`ssh -L` forwards connections to localhost on to remotely accessible servers, `ssh -R` forwards connections to the remote computer on to locally accessible servers).

    This utilizes an ssh connection from localhost to an SSH server to tunnel IP traffic. Note that if you are running a custom server on a shared login node, that server will be accessible to *all* users with access to that node unless you instruct the server to listen on a UNIX-domain socket. For that reason, you should start the server like:

    ```
    ssh -L 8001:./server.sock user@host
    serve-application --unix $HOME/server.sock
    ```

    instead of

    ```
    ssh -L 8001:locahost:8001 user@host
    serve-application --host localhost --port 8001
    ```

  - Websocket tunneling - like ssh-tunneling, it is possible to port-forward traffic through websockets using wstunnel. You will likely need to run your own wstunnel server at a publicly accessible address. This solution is rather heavy-handed, and should only be used when system adminstrators refuse to allow external traffic to route to your secure services.
  - Kubernetes port-forwarding - also available on openshift clusers, will redirect connections to the local computer to a service running within kubernetes. This can be used when adminstrators refuse to allow external traffic to route to your services running within Kubernetes [sic].
  - HAProxy - a generic TCP proxy that forwards traffic

  Note that HAProxy also allows client TLS certificate-based authentication on connections

  - NGinx can be setup to forward raw TCP/UDP traffic
  - socat / ncat - more programs that forward connections
- Application gateways - These include traffic forwarders specialized for specific protocols and applications (e.g. HTTP proxies). Additional type-safety performed at this step can prevent unintended access.
  - Traversal Using Relays around NAT service providing proxy services for data streams conforming to the real-time stream control protocol
    * A key reference here is the github.com/pion/turn server implementation library in golang

    Note that the existence of these internet standards illustrates the general state of cultural disparity between the IETF's vision of an ideal, connected, secure internet and common organizational practices around network security (breaking connectivity).
  - Web gateways / Reverse proxies - both HAProxy and NGinx can be configured to receive HTTPS traffic from clients and forward it to internal-only, HTTP servers.
  - HTTP proxies - there are many of these around, and organizations tend to use them to intercept and inspect outbound HTTP traffic. Because the client must redirect requests to the proxy server, special client setup is required to use them. Typically, command-line programs like `curl` use these when configured with `--proxy`, `$http_proxy`, or `$https_proxy`.

    - SOCKS5 proxy - these can be started with `ssh -D` and forward arbitrary IP traffic. Because the client must redirect requests to the proxy server, special client setup is required to use them. See, for example PySocks [Note: unmaintaned code].
    - Message-forwarding - Because each message-passing API encounters the need for traffic forwarding, there are effectively as many of these as there are message-passing APIs. There are variations on reliability guarantees and message sizes/rates depending on the intent of the message-passing scheme itself.
        * NNGstream
        * ZeroMQ Example
        * RabbitMQ
        * Kafka Message Brokers
- IPSec tunnels and VPNs
    - Wireguard - defines a VPN specifically between a given set of hosts This needs to be set up on each computer involved in the VPN.
    - IPSec tunnels are typically setup on dedicated router hardware by the organization maintaining that router (e.g. Arista IPSec docs)

# 6. APPLICATIONS USING INTEGRATED RESEARCH INFRASTRUCTURE

## 6.1 DIII-D DATA ANALYSIS MONITORING SYSTEM

The DIII-D project operates a tokamak fusion reactor by setting up, running, and analyzing a sequence of “shots", where each shot is a plasma emission event. The control and monitor setup for this system is setup as a group of independent systems (plant monitoring, plasma controls, tokamak diagnostic outputs, etc.) which are all able to be queried and displayed via database and web API-s.ref

The Data Analysis Monitoring System (DAM) is a webpage that accesses all these settings and diagnostic interfaces and displays them to the user. New types of data analysis routines (with both local and remote compute backends) are added by including AJAX-calls from the web page which query out to remote resources for data analysis status and outputs. Verbose outputs like high-performance compute job and error logs can be linked to, allowing the system operators to drill down into any errors or anomalies as needed.

This design is particularly illustrative of a general pattern for IRI workloads. Application teams develop custom, API-enabled monitoring and execution interfaces. These interfaces need to access status, database and compute resources provided on both internal and external networks.

From the perspective of interacting with an HPC facility, web-enabled workflows like this could be as simple as providing a compute and files API. Additional functionality which is important to the team includes real-time feedback, which could be enabled using 1) HTTP-REST API callbacks originating from nodes running a compute job, 2) streaming data outputs to the web interface enabled either using RTP/SCTP or

## 6.2 ESGF2-US

The Earth System Grid Federation (ESGF) is a collaboration that develops, deploys, and maintains software infrastructure for the management, dissemination, and analysis of model output and observational data.ref This work connects a community of data creators, consumers, and external scientific fields, as well as removes barriers to scientific investigations that will advance Earth System Science.

ESGF2-US is building and deploying modernized infrastructure that builds on recent technology trends including web application design, user computation via notebooks, container-orchestration, and analysis-ready, cloud-optimized data formats like Zarr. Like DIII-D, ESGF2-US hosts a web application to interact with the datasets. However, this web application is focused on data search, publication, and transfer rather than real-time parameterization and data acquisition.

One of the key innovative activities evolving from this project is the intake-esgf python client library for accessing ESGF’s holdings. This client accomplishes integration with both reproducible workflows and notebooks, includes local data caching capabilities, and interfaces with computational back-ends for data analysis. Technically, the data is pulled from both ESGF metadata servers (ESGF catalog), globus shared-data endpoints, and HTTP downloads.

Data analysis can be accomplished by any python code importing intake-esgf. Hence, analysis can be executed using both globus compute and rooki, as well as any compute backend capable of executing python code and accessing the internet to download ESGF data.

The Rooki service differs from the compute APIs mentioned in the sections above in that it is custom-built for the climate science community. It is a client library for interacting with the rook compute service. That compute service uses PyWPS, which wraps a python-flask web application. The application itself runs unprotected, with the expectation that it will be run behind an HTTPS reverse-proxy. Note that this maps all users to a single identity. This may be appropriate given that the server does not maintain significant user-associated state, however this also makes users’ outputs visible to all other users.

## 6.3 LIGHT SOURCES COLLABORATION

The Advanced Photon Source at ANL has developed an APS Data Management system that provides a python client API, command-line interface, and web-based graphical user interface to beamline scientists to run compute jobs to analyze their data. The server-side processing is done through pre-defined functions that are pre-registered with a centralized Globus Compute service. Users send requests to run those functions to the service, and the compute nodes receive run instructions from the service. This has enabled a series of applications including Bragg peak-finding, an "ALCF Community Data Coop" data portal, structure reconstruction from Laue microdiffraction data using the cold package, ptychography reconstruction using the ptychodus package, tomography using TomocuPy, and x-ray photocorrelation spectroscopy using boost_corr.

- Empowering Scientific Discovery Through Computing at the Advanced Photon Source Parraga et. al., 2023
- Demonstrating Cross-Facility Data Processing at Scale with Laue Microdiffraction Prince et. al., 2023

The ALS microtomography beamline (BL8.3.2) at LBNL is performing structural inference (reconstruction) using compute resources at both NERSC and ALCF. Their workflows are run by an internally deployed Prefect service. The beamline's data acquisition system (DAQ) contains code which triggers Prefect to perform experimental data copy to HPC, data transformation at HPC, and result copy from HPC steps. The experimental data is sent to HPC in HDF5 format, while the results are returned as reconstruction models with various resolutions – tuned for visualization at the APS beamline.

Experimental end-stations at NSLS-II are building a full software stack for interfacing with experimental controls (ophyd), caching and serving experiment data (suitcase, databroker/intake, tiled), and Bayesian optimization. Connectivity between these applications is HTTP-REST API based, and diagrammed at blueskyproject.io.

- Introduction of the Sirepo-Bluesky interface and its application to the optimization problems Rakitin et. al., 2020

## 6.4 GRETA/DELERIA FRAMEWORK

The Gamma-Ray Energy Tracking Array (GRETA) project operates a series of gamma-particle detectors that are used for understanding nuclear structure and reactivity, isotope stability, and subatomic particle physics ref.

During experimental operation, the detectors produce high-rate data that must be processed through signal analysis and event assembly steps. In order to facilitate running these analysis routines on OLCF's HPC resources, the DELERIA project uses three key components:

1. A webpage displaying a control interface to containerized applications running across multiple distributed computers (JANUS control interface)
2. A forward buffer which stores detector outputs sent via UDP packets as they await processing by compute nodes
3. Experiment monitoring software which logs (and presumably responds to) significant data findings as well as status changes of elements within the compute infrastructure.

The forward buffer utilizes a similar to NNG-stream. Its operation requires the ability to send high-rate, low-latency data on a path from the detectors to the compute nodes. This is not easy to accomplish with standard compute cluster setups, which do not expose network access to compute nodes. All the monitoring and control software is most easily accomplished with json-formatted HTTP-REST API calls, or an equivalent technology such as Google protocol-buffer formatted gRPC or equivalent ZMQ or NNG messages.

## 6.5 NERSC-NESAP ACTIVITIES

The NERSC Science Acceleration Program (NESAP) partners with development teams to better integrate those applications with new hardware and workflow technologies as they are tested and deployed at NERSC. Several of these projects connect to work at other DOE experimental facilities, such as SLAC National Accelerator Lab and the Advanced Light Source at LBNL.

- Real-time XFEL data analysis at SLAC and NERSC: A trial run of nascent exascale experimental data analysis Blaschke et. al., 2024
- Streamlined Web-Based Data Analysis at the ALS using BilboMD BilboMD server / Pelikan, Hura, and Hammel, 2009
- Streaming Large-Scale Microscopy Data to a Supercomputing Facility github.com/OpenChemistry/distiller / Welborn et. al., 2024, Welborn et. al., 2024
- Graph Neural Network-based Tracking as a Service Zhao et. al., 2023

For the examples above, the general pattern is similar to DIII-D. An application-specific webpage front-end, often connected to an experiment-tracking database, connects to the NERSC SuperFacility API in order to create compute jobs and display their outputs. Additional streaming data components are enabled via network routing that allows compute nodes to connect directly to experimental data sources. In this case, the ZeroMQ message passing protocol is used with an application gateway (the NCEM central aggregator).

## 6.6 LCLSTREAM

The LCLStream project enables HPC centers to run data analysis of both live and archived data from experimental end-stations at SLAC National Accelerator Laboratory's Linac Coherent Light Source (LCLS). Users of a given beamline end-station help define appropriate data reduction and compression routines to run locally at the data source. This compressed data stream is sent in the form of a stream of small, in-memory, hdf5-formatted data packets using the LCLStream client library, combined with the LCLStream-API. These are buffered through NNG-Stream, which stores the data until it is pulled for computations by the running HPC job.

In order to enable authentication and orchestration of this workflow, several packages mentioned above, as well as some novel, re-usable components were also created:

- certified plus signer - solving the federated identity and authentication problem by wrapping standard HTTP API-s
- planner_api - solving the experiment/HPC time co-scheduling problem
- rendezvous - solving the "synchronized start" problem
- psik-api - providing a simple, secure, and user-runnable, compute and file API

## 7. CONCLUSIONS

We are seeing a common, emerging pattern of application teams implementing and managing their own, site-local web interfaces and experiment-tracking databases. These are driving an increased call for for on-demand compute.

There are ample off-the-shelf software implementations available both for creating secure web interfaces and providing compute and file resources. What is missing is a universally acknowledged authentication and credential authorization mechanism, programming support for building appropriate connecting layers between facility-provided compute interfaces and application-required experiment management and tracking interfaces, and networking, security and regulatory frameworks for controlling and monitoring access to these systems. These latter activities will require significant investment by compute centers to configure, test, document, customize, and ultimately deploy on available software and hardware. In addition, the user community utilizing these technologies needs to work together to minimize the cost of developing and testing client interfaces to this infrastructure as well.

It is recommended that working committees on authentication/authorization, interfaces, and outreach and engagement fully explore the implications of running the software stacks used by emerging IRI applications. These will ground their discussions in key use cases, as well as provide valuable additions, interoperable components, and feedback and improvements to well-defined software stacks.