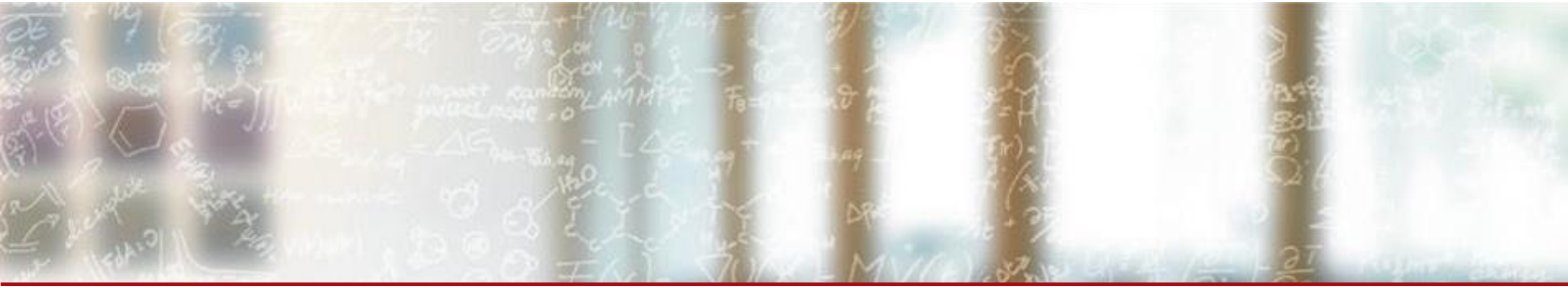
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



FirecREST v2: Lessons Learned from Redesigning an API for Scalable HPC Resource Access

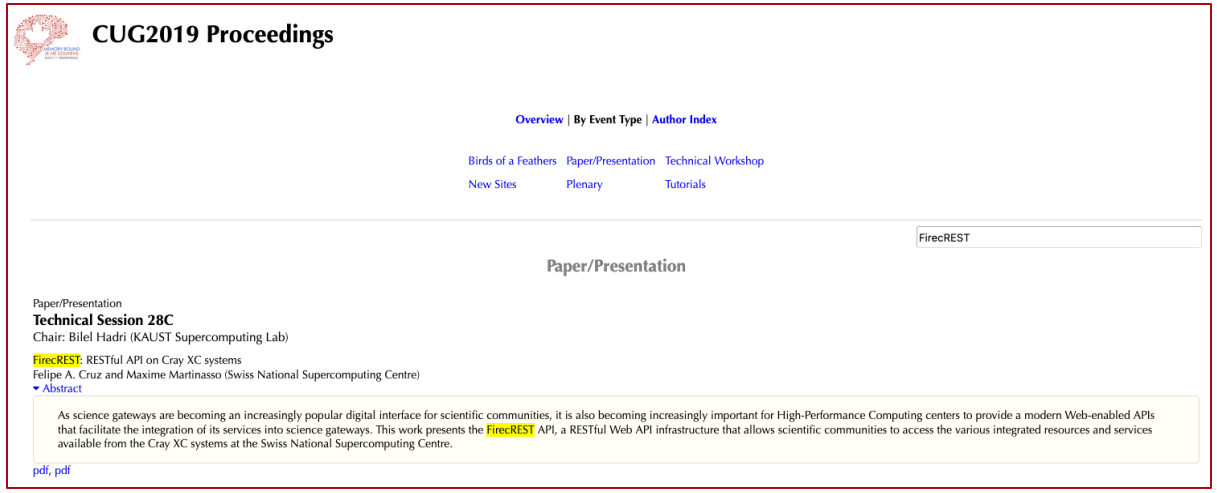
Elia Palme, **Juan Pablo Dorsch** (juanpablo.dorsch@cscs.ch), et al.
CSCS - Swiss National Supercomputing Centre
ETH-Zurich

CUG-25 Conference
New York, USA, May 8, 2025

The version 1 of FirecREST

FirecREST version 1

- FirecREST v1 was a response to
  - Enabling HPC workflows automation through a programmatic interface
  - Streamlining sysadmins' workflows
  - Cloud technologies entering the HPC market


CUG2019 Proceedings

Overview | By Event Type | Author Index

Birds of a Feathers Paper/Presentation Technical Workshop
New Sites Plenary Tutorials

FirecREST

Paper/Presentation

Paper/Presentation
Technical Session 28C
Chair: Bilel Hadri (KAUST Supercomputing Lab)

FirecREST: RESTful API on Cray XC systems
Felipe A. Cruz and Maxime Martinasso (Swiss National Supercomputing Centre)
Abstract

As science gateways are becoming an increasingly popular digital interface for scientific communities, it is also becoming increasingly important for High-Performance Computing centers to provide a modern Web-enabled APIs that facilitate the integration of its services into science gateways. This work presents the FirecREST API, a RESTful Web API infrastructure that allows scientific communities to access the various integrated resources and services available from the Cray XC systems at the Swiss National Supercomputing Centre.

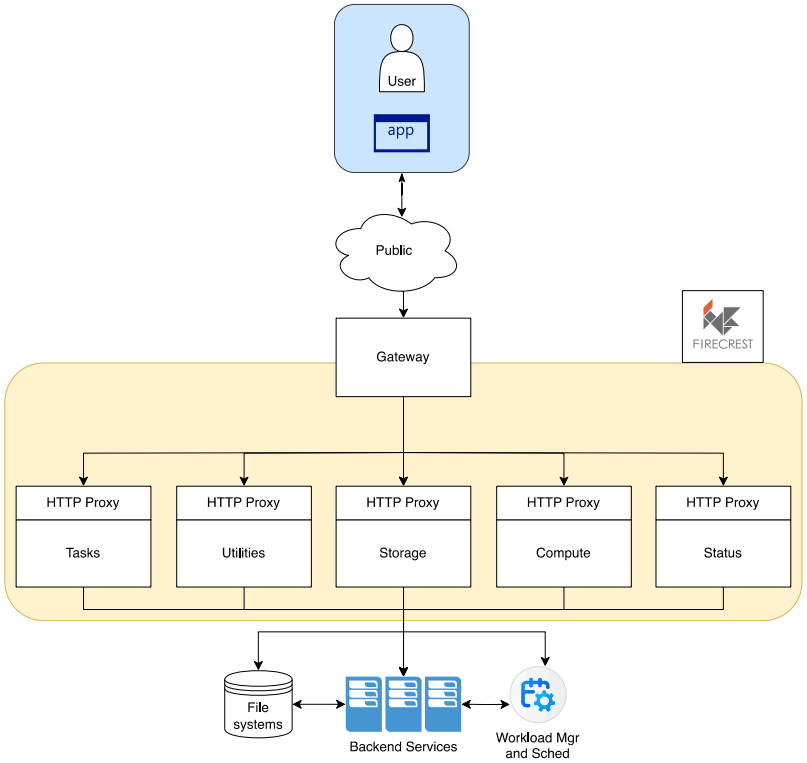
pdf, pdf

- FirecREST v1 provided
  - Lightweight HTTP access to HPC resources
  - Abstraction for workload managers and schedulers, clusters, and storage
  - User authentication and resource access authorization

- *After years of development and usage, an **evaluation** of the **architecture**, **technologies,** and **performance** was needed*

Version 1 Architecture

- The first version of FirecREST was based on a microservice architecture
- Each microservice is a container with a specific function (gateway, compute, storage, status, etc)
- Providing
  - Flexibility on maintenance and scalability
  - Fits in cloud architectures
  - Introduces some complexity in inter-container communication

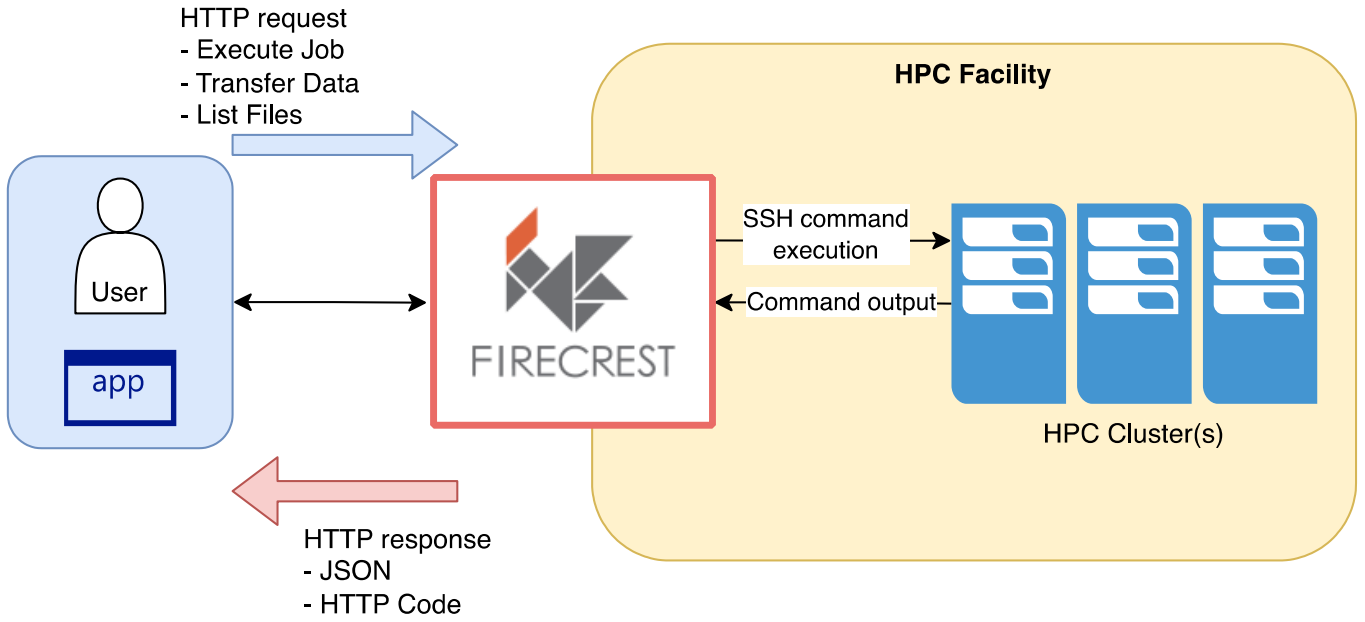


Version 1 Technologies

- HTTP Server
  - FirecREST was design as a **HTTP proxy to HPC** developed in Python's ***Flask*** Library
  - Flask is not design to handle large number of requests (ie, ~100 req/sec): it requires of an HTTP server
  - The HTTP server technology chosen was ***Gunicorn*** using ***multithreading***
  - ***Multithreading*** handles requests a separate threads: incoming request are reallocated in a new thread while waiting for others request to complete
  - Important: multithreading in Python processes requests sequentially due to GIL (Global Interpreter Lock) limitation

Version 1 Technologies – SSH authentication

- SSH connection
  - Command execution on FirecREST is done via SSH
  - In v1, each command is issued with an SSH certificate for a specific user, command, a short valid time using ***Paramiko*** library

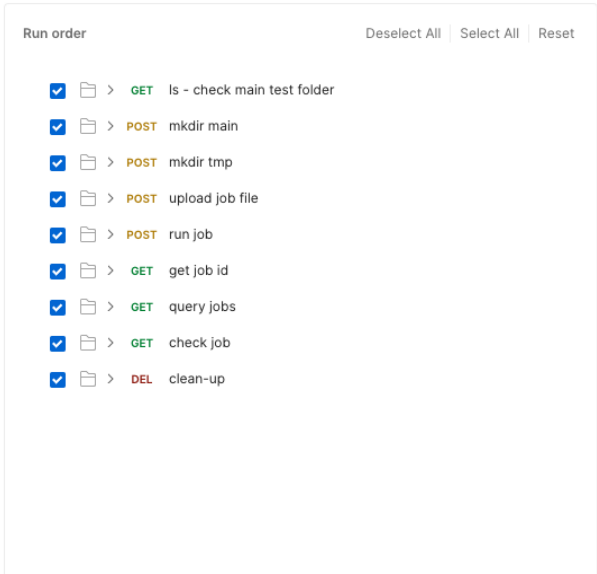


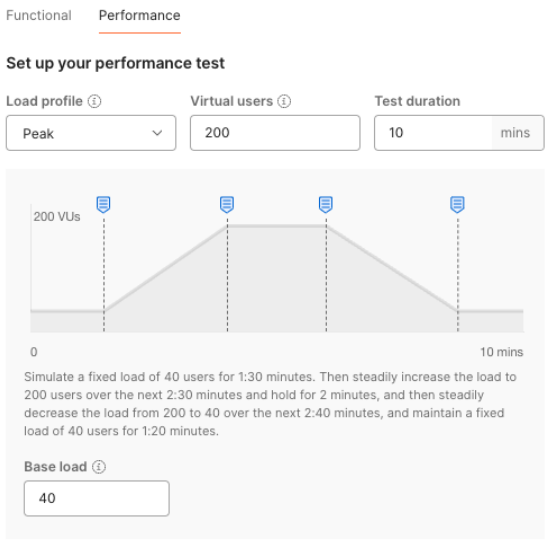
Version 1 Performance Evaluation

Version 1 Performance evaluation

- Test configuration
  - Representative set of requests (filesystem's operations, job scheduler, data transfer, etc)
  - Complete job lifecycle simulation (system setup, job submission, job monitoring, result retrieval)
  - Test performed on a CSCS Alps vCluster (HPE-Cray EX system)

- Test load simulation
  - Using ***Postman*** to simulate the number of requests and clients and dynamically increase them to simulate traffic peaks





Version 1 Performance evaluation

- In this evaluation, 3 types of analysis were performed
  - Log analysis
    - Measures the timestamp of individual API calls to identify bottlenecks in complete job simulations
    - Helps to uncover limitations on how HTTP server processes requests
  - Network traffic analysis
    - Examines problems on FirecREST interfacing backend services (HPC systems, IAM layer, Storage)
  - Resource usage analysis
    - Monitors the usage of resources in the backend systems (CPU and memory)
    - Key to find issues on SSH connections

Version 1 Performance evaluation

- Identified bottlenecks
  - Web server: multithreading presents scalability issues, since the number of threads depends on the server (OS and resources)
    - If the number of requests is larger than supported threads, the web server starts to drop requests
    - Additionally, it introduces small memory overheads
  - SSH Authorization
    - Each command executed via FirecREST-v1 creates one SSH connection, which is limited by the `MaxSessions` parameter in SSHd configuration
    - This is a limitation to execute multiple concurrent commands on the cluster
    - Moreover, processing SSH credentials increases CPU time and resources

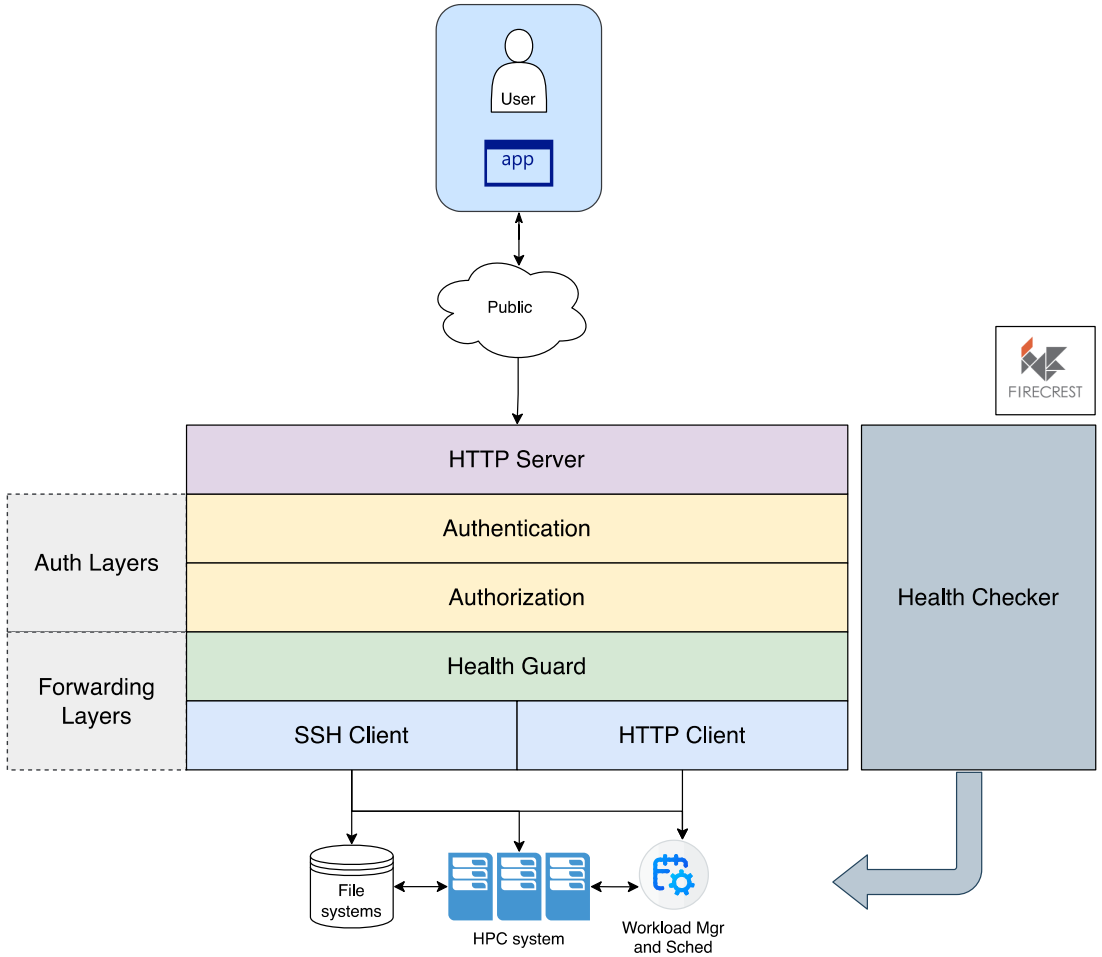
FirecREST v2 Architecture

FirecREST v2 Architecture

- Through the evaluation of version 1, we defined the core principles to design an improved version:
  - **Simplify architecture** by discarding microservice architecture to avoid issues on interconnection to simplify deployment and remove overheads
  - **Minimize interaction with underlying services** by reducing the number of communication with backend services (cluster, scheduler) to decrease latency and improve performance
  - **Implement Asynchronous I/O operations** by enabling asynchronous communication where possible to avoid bottlenecks on HTTP request/response and be independent of the system limitations

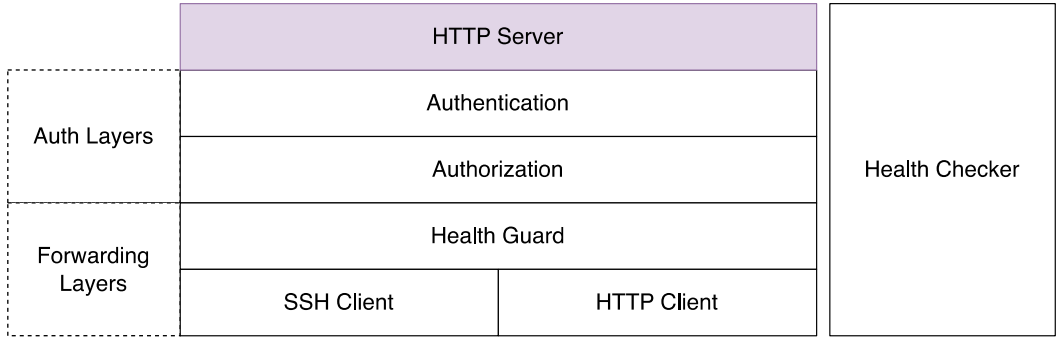
FirecREST v2 Architecture

- To apply this core principles, a new **layered architecture** has been proposed



FirecREST v2 Architecture

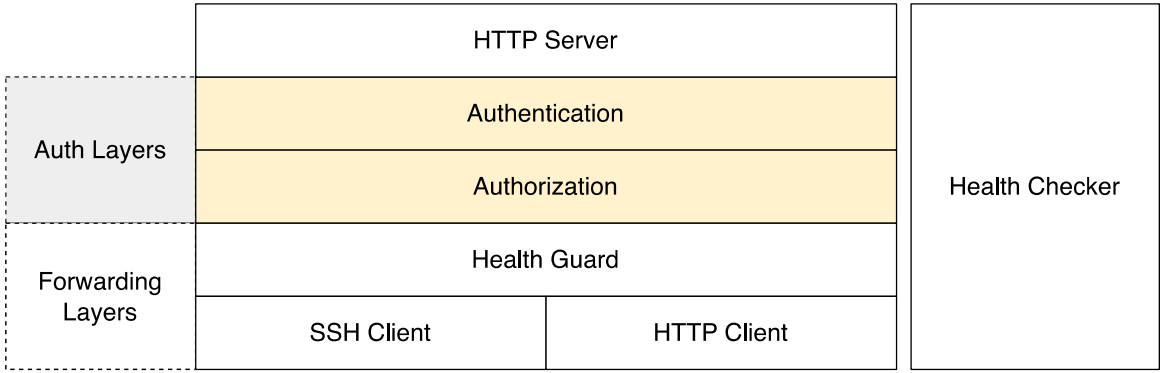
- HTTP Server
  - Framework and libraries selection for full Async performance
    - **FastAPI**: high-performance web framework for APIs on Python; with native asynchronous operation



    - **Uvicorn**: ASGI server (Asyncrhonous Server Gateway Interface)
    - **AsyncIO**: for asynchronous I/O bound operations, enables asynchronous HTTP communication to backend services
    - **AsyncSSH**: SSH client built on AsyncIO, enables asynchronous SSH connections to the clusters

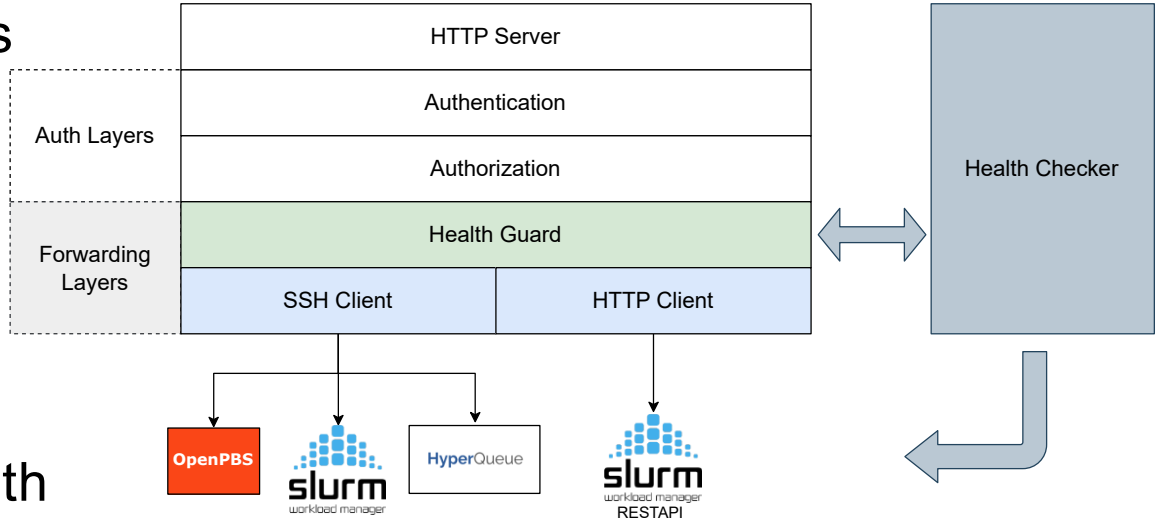
FirecREST v2 Architecture

- Auth Layers
  - Authentication (AuthN)
    - Complies with OIDC/OAuth2 standard
    - Offline JWT validation
  - Authorization (AuthZ)
    - After AuthN, it checks permissions on accessing resources
    - Enables JWT claim AuthZ
    - Supports ReBAC, ABAC, RBAC
    - Natively it supports OpenFGA



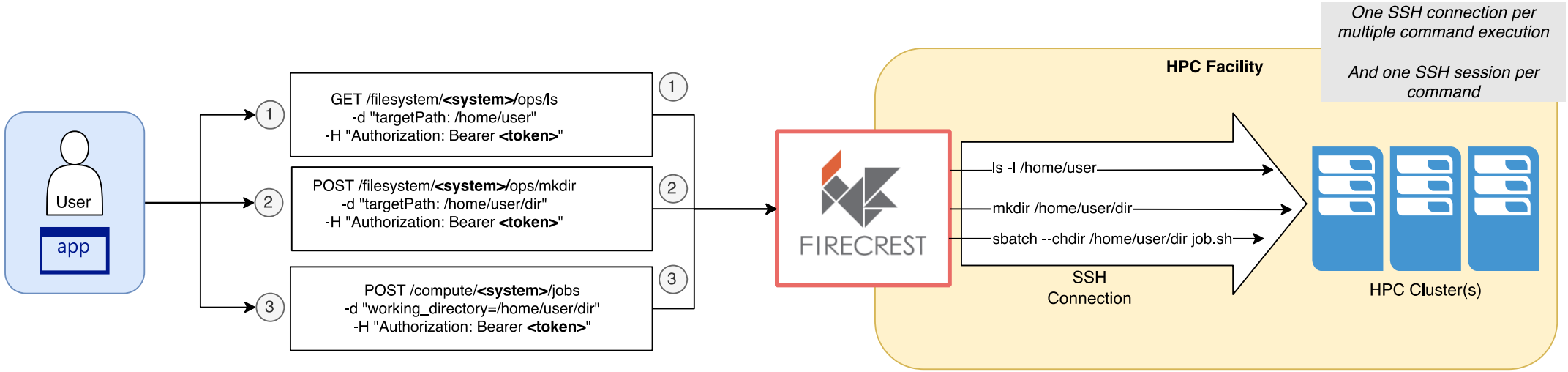
FirecREST v2 Architecture

- Forwarding Layers
  - Health Guard
    - Monitors the health of target resources
    - Requests are only forwarded when a resource is ready
    - Reduces latency issues
  - Clients
    - Clients are organized depending the abstraction (scheduler, filesystems) with a standard interface
    - For instance, scheduler abstracts SLURM, SLURM API, OpenPBS, etc
    - Protocols: routing the request depending the type of client (SSH or HTTP)



FirecREST v2 Architecture

- SSH client improvement
  - The v2 SSH Connection Pool outperforms v1 on command execution
  - Instead of using one SSH connection+channel per command, this new approach re-uses an SSH connection to execute several commands on a row.
  - Each connection in the pool is closed after a time of inactivity, leaving it available for the next user.
  - *This feature requires to adjust the `MaxSessions` setting in SSHd Config*



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Benchmarking

Benchmarking

- Testing under real use case conditions
  - **AiiDA** is a workflow manager that provides high-throughput calculations on HPC
  - The team from **Material Clouds** is working actively in the `aiida-firecrest` Python library

Benchmarking

- Testing under real use case conditions
  - **AiiDA** is a workflow manager that provides high-throughput calculations on HPC
  - The use case selected was I/O bound operations to measure latency
    - Up to 1000 small files (1KB) were generated on the server and were retrieved via FirecREST using v1 and v2 endpoints.
    - API requests were made with `AsyncClient` of the `httpx` Python library
  - The CSCS' Alps HPE-Cray system Eiger was used
  - Requests were made from an external network to CSCS

Results and conclusions

Benchmarking

- Results

Number of files	FirecREST v1	FirecREST v2	FirecREST v2 (w/SSH connection pool)
1	1.5 ± 0.1 [s]	0.8 ± 0.02 [s]	**0.4 ± 0.3 [s]**
10	13.7 ± 1.1 [s]	2.7 ± 0.05 [s]	**0.5 ± 0.2 [s]**
100	129.6 ± 1.5 [s]	19.5 ± 3.4 [s]	**1.5 ± 0.7 [s]**
1000	*Too long (> 1000 [s])*	176.3 ± 4.5 [s]	**15.5 ± 8.9 [s]**

Times in seconds measures the time between the request and the file being retrieved
Each test was executed 5 times. Results show mean value and standard deviation

Benchmarking

- Results

Number of files	FirecREST v1	FirecREST v2	FirecREST v2 (w/SSH connection pool)
1	1.5 ± 0.1 [s]	0.8 ± 0.02 [s]	**0.4 ± 0.3 [s]**
10	13.7 ± 1.1 [s]	2.7 ± 0.05 [s]	**0.5 ± 0.2 [s]**
100	**129.6 ± 1.5 [s]**	**19.5 ± 3.4 [s]**	**1.5 ± 0.7 [s]**
1000	*Too long (> 1000 [s])*	176.3 ± 4.5 [s]	**15.5 ± 8.9 [s]**

~ 85x faster than v1

- **Results for this high-throughput I/O workflow have shown an impressive improvement of v2 over its predecessor, and moreover using the SSH connection pool**

Times in seconds measures the time between the request and the file being retrieved
Each test was executed 5 times. Results show mean value and standard deviation

Conclusions

- Version 2 of FirecREST has proven to be a substantial improvement to the first version of the API while supports high-throughput regime use cases

- This was achieved from a thorough analysis used to identify the bottlenecks in version 1 which led us to select a new architecture and technologies

- The key enhancements can be listed as
  - End-to-end asynchronous request forwarding (from HTTP to SSH)
  - Health monitoring reduces latency, improves response times and adapts to the needs of the HPC facility
  - SSH connection pool enhance the communication between the API and SSH backend services (scheduler, filesystems operations, data transfer, etc)

- Besides the improvements on performance
  - Architecture is simpler
  - Abstraction layers provide better integration with different vendors

Acknowledgements

FirecREST v2: Lessons Learned from Redesigning an API for Scalable HPC Resource Access

Elia Palme
elia.palme@cscs.ch
CSCS – Swiss National Supercomputing Centre
Lugano, Switzerland

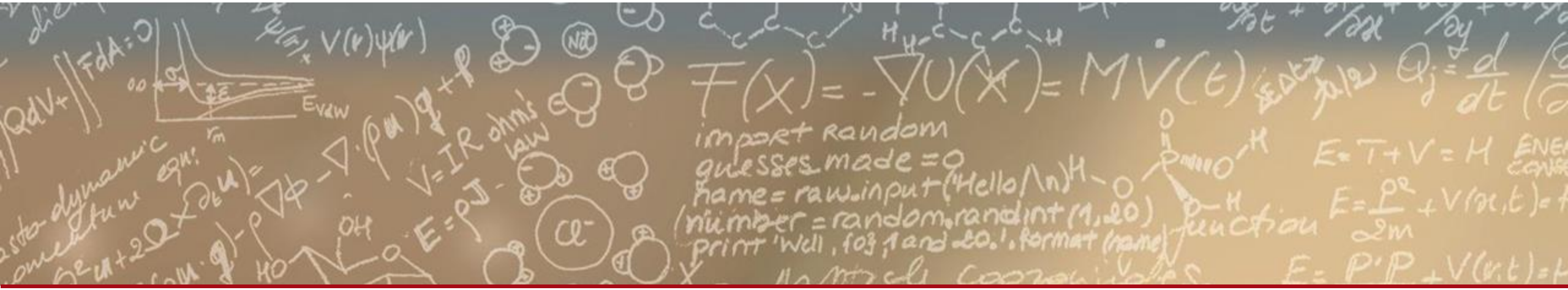
Juan Pablo Dorsch
juanpablo.dorsch@cscs.ch
CSCS – Swiss National Supercomputing Centre
Lugano, Switzerland

Ali Khosravi
Giovanni Pizzi
ali.khosravi@psi.ch
giovanni.pizzi@psi.ch
PSI Center for Scientific Computing, Theory, and Data
5232 Villigen PSI, Switzerland

Francesco Pagnamenta
Andrea Ceriani
Eirini Koutsaniti
Rafael Sarmiento
Ivano Bonesana
Alejandro Dabin
CSCS – Swiss National Supercomputing Centre
Lugano, Switzerland

FIRECREST

eth-cscs.github.io/firecrest-v2



Thank you for your attention.