Accessing HPC resources via RESTful API

INFOCOMP 2022

Christian Köhler
christian.koehler@gwdg.de

June 29, 2022

hpc@gwdg.de
GWDG – Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Introduction

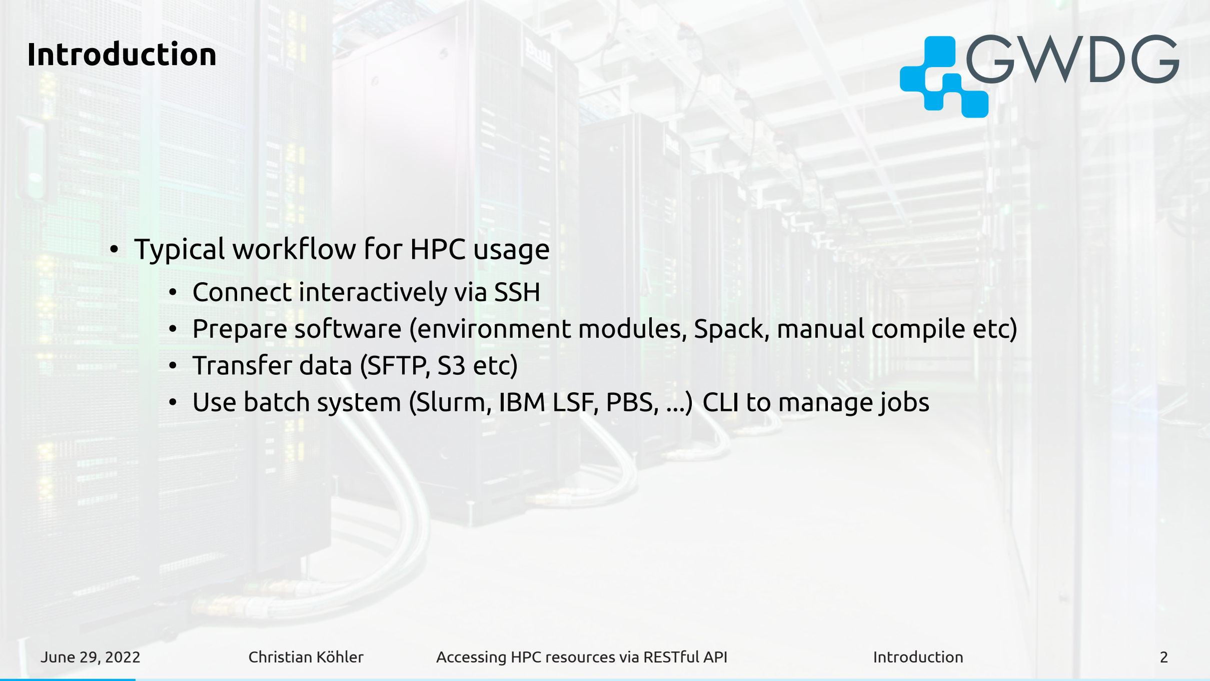
Architecture

Use Cases

Authentication

Summary & Outlook

Introduction



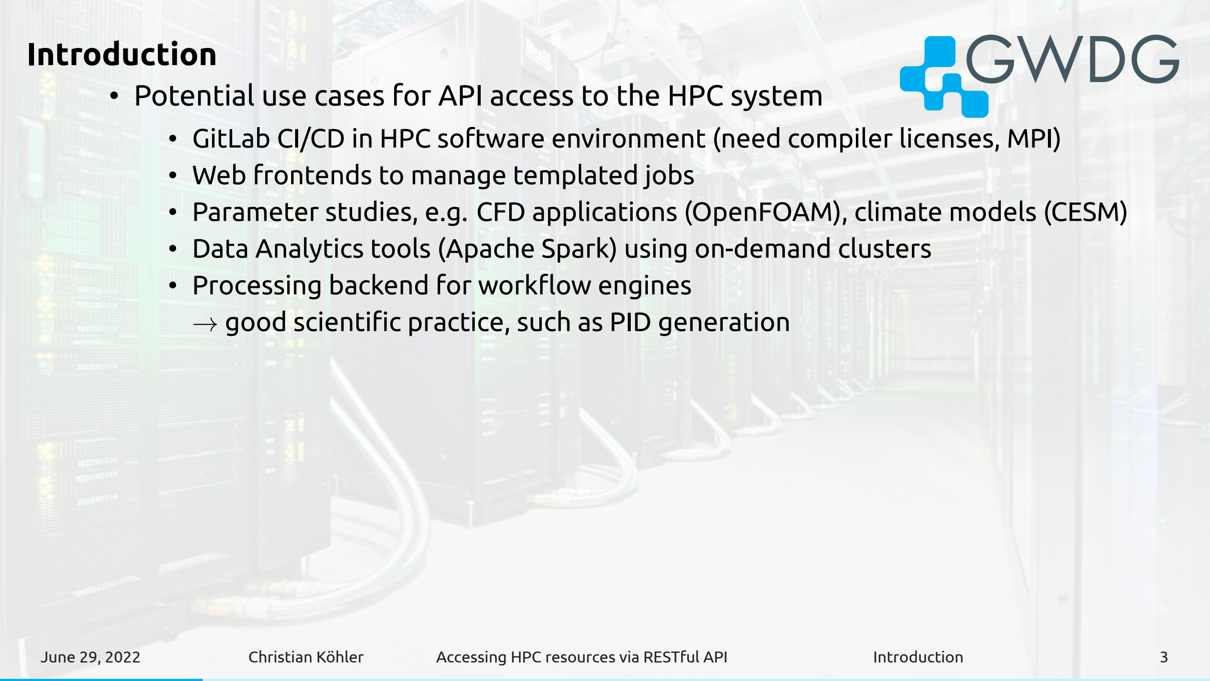
Introduction

- Typical workflow for HPC usage
  - Connect interactively via SSH
  - Prepare software (environment modules, Spack, manual compile etc)
  - Transfer data (SFTP, S3 etc)
  - Use batch system (Slurm, IBM LSF, PBS, ...) CLI to manage jobs

Introduction

- Typical workflow for HPC usage
  - Connect interactively via SSH
  - Prepare software (environment modules, Spack, manual compile etc)
  - Transfer data (SFTP, S3 etc)
  - Use batch system (Slurm, IBM LSF, PBS, ...) CLI to manage jobs

→ **What if HPC is needed as a backend for other applications/services?**



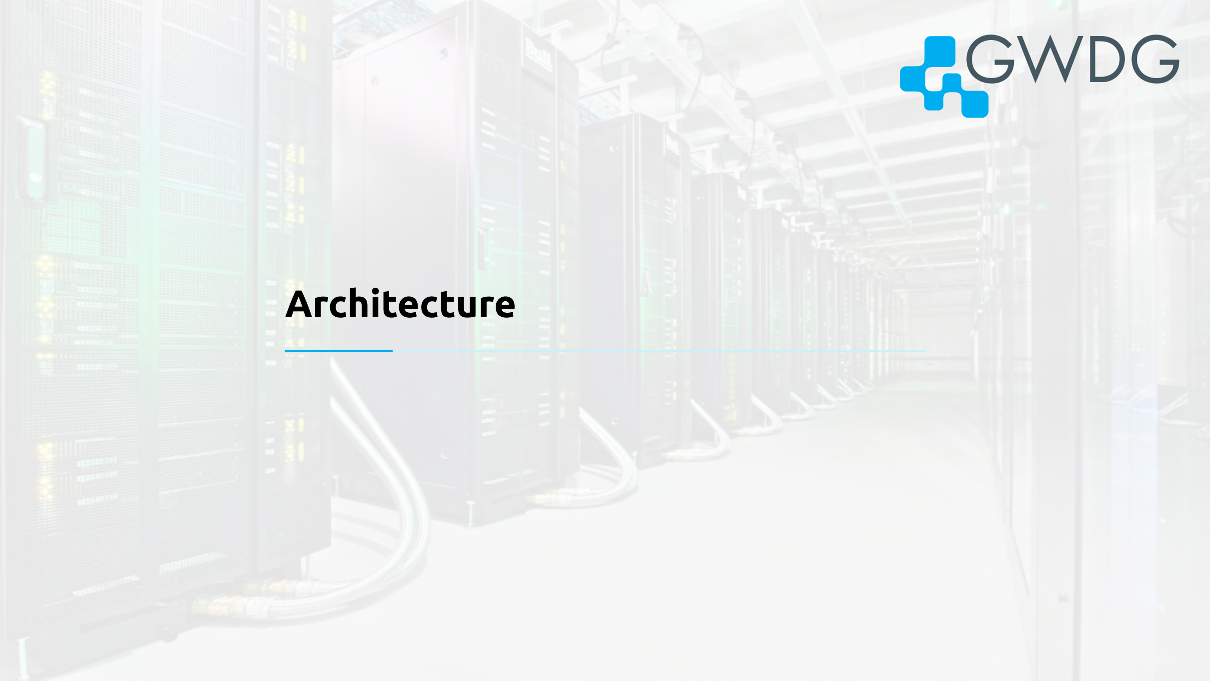
Introduction

- Potential use cases for API access to the HPC system
  - GitLab CI/CD in HPC software environment (need compiler licenses, MPI)
  - Web frontends to manage templated jobs
  - Parameter studies, e.g. CFD applications (OpenFOAM), climate models (CESM)
  - Data Analytics tools (Apache Spark) using on-demand clusters
  - Processing backend for workflow engines

    → good scientific practice, such as PID generation

Introduction

- Potential use cases for API access to the HPC system
  - GitLab CI/CD in HPC software environment (need compiler licenses, MPI)
  - Web frontends to manage templated jobs
  - Parameter studies, e.g. CFD applications (OpenFOAM), climate models (CESM)
  - Data Analytics tools (Apache Spark) using on-demand clusters
  - Processing backend for workflow engines
    → good scientific practice, such as PID generation
- **Design goals of HPCSerA**
  - Provide REST interface for HPC
  - Allow running smaller tasks on HPC frontends
  - Integration of smaller data transfers (e.g. code repositories)
  - Scheduler agnostic/adaptable
  - Easy integration while maintaining security
- Out of scope: initial setup and testing of HPC software



Architecture

Architecture

- REST API is accessed via HTTP(S) requests by
  - **Client** for submitting jobs, querying status (per service)
  - **HPC agent** to pull new jobs, feedback status, post results (per HPC site)

Architecture

- REST API is accessed via HTTP(S) requests by
  - **Client** for submitting jobs, querying status (per service)
  - **HPC agent** to pull new jobs, feedback status, post results (per HPC site)
- First prototype: simple static authentication model
  - How to easily revoke trust from some service/site?

  → Need individual authentication for each involved party, separation of projects

Architecture

- REST API is accessed via HTTP(S) requests by
  - **Client** for submitting jobs, querying status (per service)
  - **HPC agent** to pull new jobs, feedback status, post results (per HPC site)
- First prototype: simple static authentication model
  - How to easily revoke trust from some service/site?

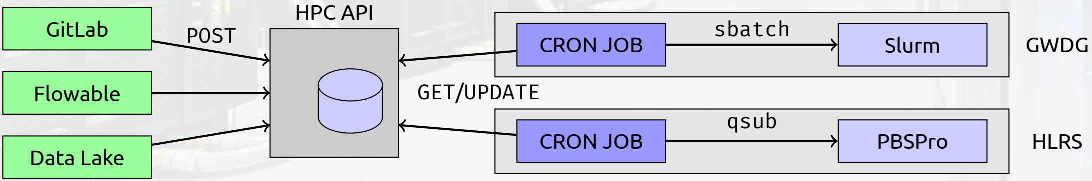
  → Need individual authentication for each involved party, separation of projects
- Current development project “HPCSerA” (**HPC Ser**vice **API**):
  API server, HPC agent, default client

Architecture components

- Main components: external services, API server, HPC systems

→ For example, in our use cases GitLab and OpenForecast: Scientific Compute Cluster of GWDG and HAWK at HLRS

- **HPC agent** can run as a cronjob/inside `screen` on the HPC frontend
  - Implicitly run shell commands for batch system interaction (defined in separate configuration file)
  - Job steps for login nodes/data movement (via `subJobTypes`)
  - Only need user privileges



API definition

- API service
  - Representational State Transfer (REST)
  - Access via HTTP(S) protocol
- OpenAPI 3.0 Specification
  - Definition in YAML format
  - Swagger Codegen: generate server/client codebase, documentation
  - Client SDK in our case: Python module
  - Machine readable

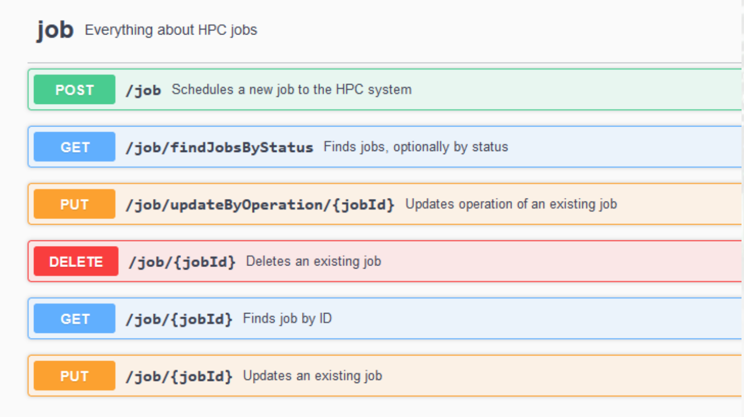
API definition

- API service
  - Representational State Transfer (REST)
  - Access via HTTP(S) protocol
- OpenAPI 3.0 Specification
  - Definition in YAML format
  - Swagger Codegen: generate server/client codebase, documentation
  - Client SDK in our case: Python module
  - Machine readable

swagger.yaml (abbreviated)

```
openapi: 3.0.0
 /job/{jobId}:
  get:
   summary: Finds job by ID
   description: Returns a single job
   parameters:
  — name: jobId
   responses:
    ”200”:
     description: Successful operation
     content:
      application/json:
       schema:
        $ref: ’#/components/schemas/Job’
```

API definition
HTML documentation



job Everything about HPC jobs

- POST `/job` Schedules a new job to the HPC system
- GET `/job/findJobsByStatus` Finds jobs, optionally by status
- PUT `/job/updateByOperation/{jobId}` Updates operation of an existing job
- DELETE `/job/{jobId}` Deletes an existing job
- GET `/job/{jobId}` Finds job by ID
- PUT `/job/{jobId}` Updates an existing job

Related work

- *NEWT (NERSC Web Toolkit)* - developed at LBNL
  - OAuth/LDAP/Shibboleth authentication

  → Trusting the authentication provider
- *FirecREST* - developed at CSCS
  - Microservice provides *SSH certificate*
  - SSH daemon of the HPC system has to be configured accordingly
- *slurmrestd* - part of Slurm workload manager
  - Use *MUNGE* service or JWT tokens for authentication

Use Cases

Use Cases

GitLab CI/CD, Workflow Engine

- GitLab CI/CD
  - Runner uses the default client to run jobs+YAML configuration
  - Setup with `.gitlab-ci.yml` in repository
    - never include credentials/tokens here!

  → GitLab secrets

Use Cases
GitLab CI/CD, Workflow Engine

- GitLab CI/CD
  - Runner uses the default client to run jobs+YAML configuration
  - Setup with `.gitlab-ci.yml` in repository
    - never include credentials/tokens here!

    → GitLab secrets
- Workflow Engine
  - Idea in Open Forecast project: combine open data with supercompting
  - *Flowable* is a graphical (BPMN) workflow tool
  - “HTTP task” allows integration with the API
  - Batch job is a *Singularity* container
    - pulled from GitLab container registry
    - started with individual parameters

Use Cases

Data Lake

- Storage of raw data, metadata + job manifests for processing
- Job manifests specify the entire environment:
  - Container image, dependencies, shell environment (+annotations)

  → Stored in the data lake, indexed and searchable
- Data Lake acts as a **Client** for HPCSerA
- **HPC agent**
  - Runs preprocessing script:
    build dependencies from Git, log versions, download input data
  - Submits HPC job to the batch system (container binary is being kept)
  - Postprocessing: Ingest of created artifacts into the data lake
- Resulting data and container image are stored in the Data Lake
- Provenance data can be collected without application overhead

Authentication

Authentication workflow
Requirements

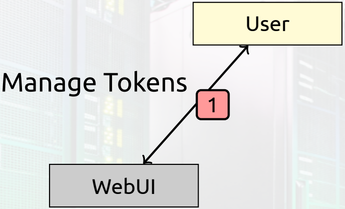
- Users should be able to self-manage access tokens
- Security concern: “Are we exposing our system?”
  - Access is granted per user, project, and client
  - Authorization via tokens should not grant arbitrary permissions

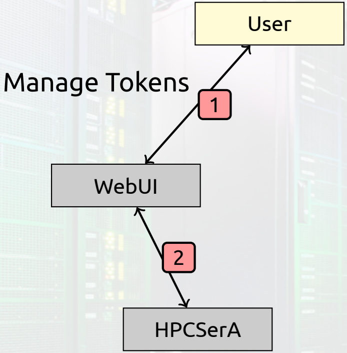
  ⇒ Limit by role

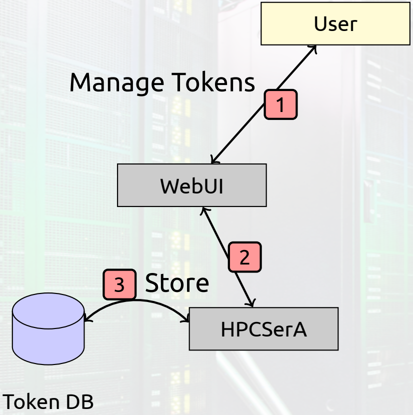
Authentication workflow

Improvement over static authentication model

1. User creates tokens via WebUI
2. WebUI uses HPCSerA as backend
3. Access tokens are stored in database
4. **Client** uses the token for a request
5. Token is validated against the database
6. On-demand token creation: Ask Auth app instead
7. Get user decision on token → step 3
8. **HPC Agent** uses token to retrieve/update jobs
9. Ingest of new code to HPC: Ask Auth app
10. Run task/job on HPC, interact with Batch system


User
Manage Tokens
1
WebUI


User
Manage Tokens
1
WebUI
2
HPCSerA



User

1 Manage Tokens

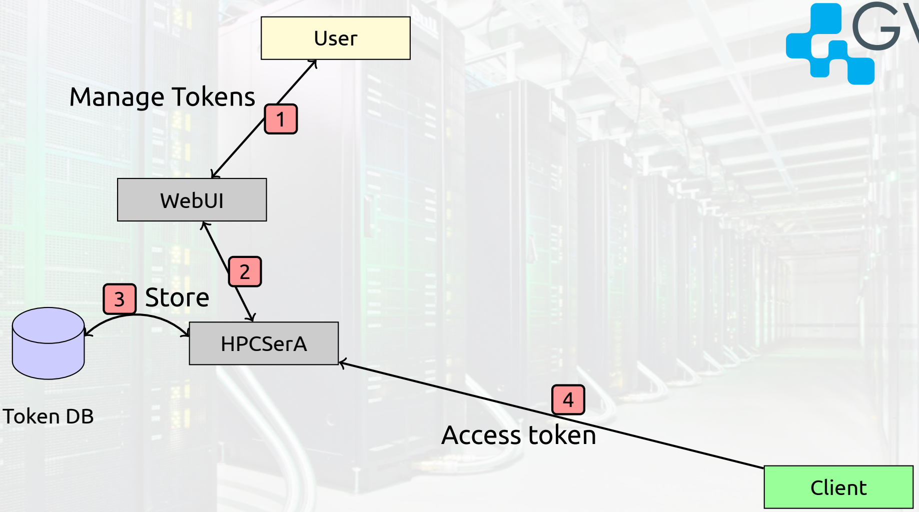
WebUI

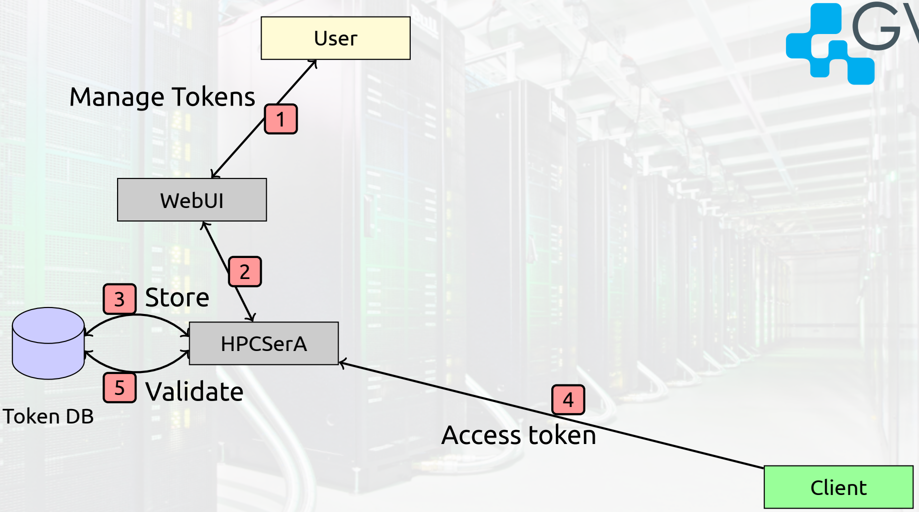
2

3 Store

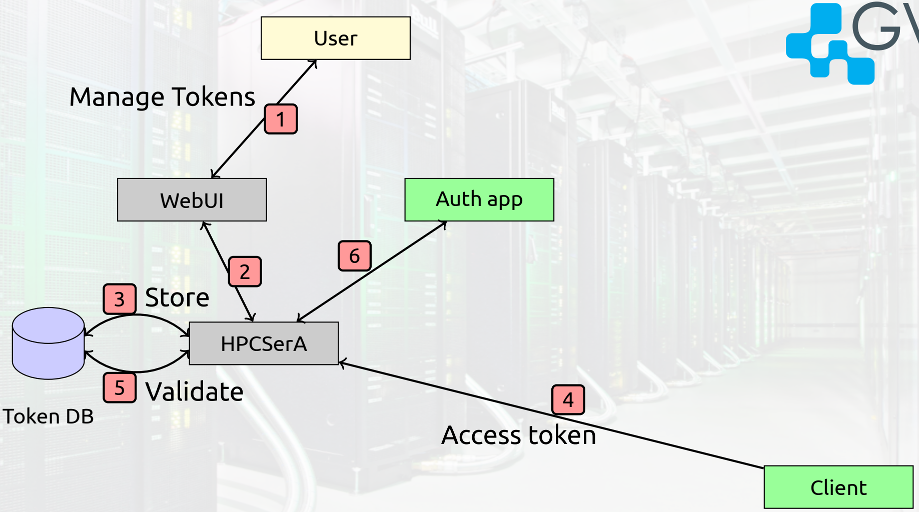
HPCSerA

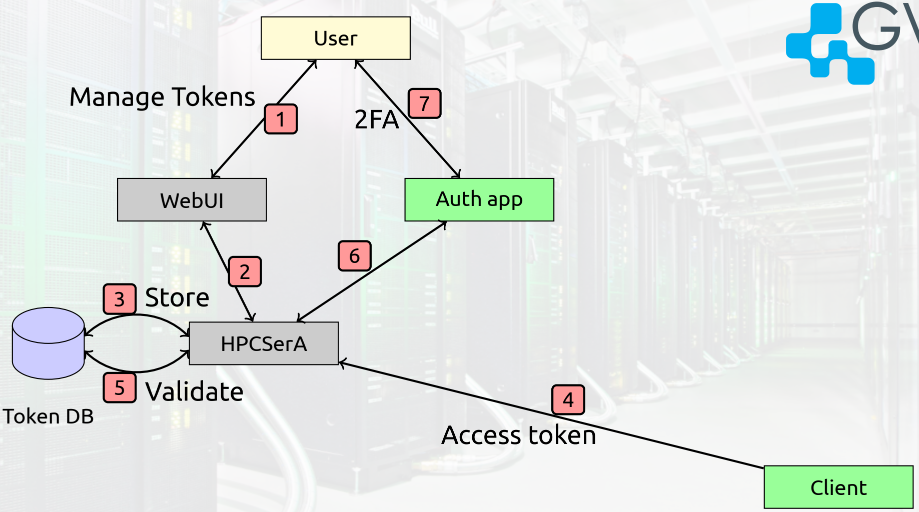
Token DB

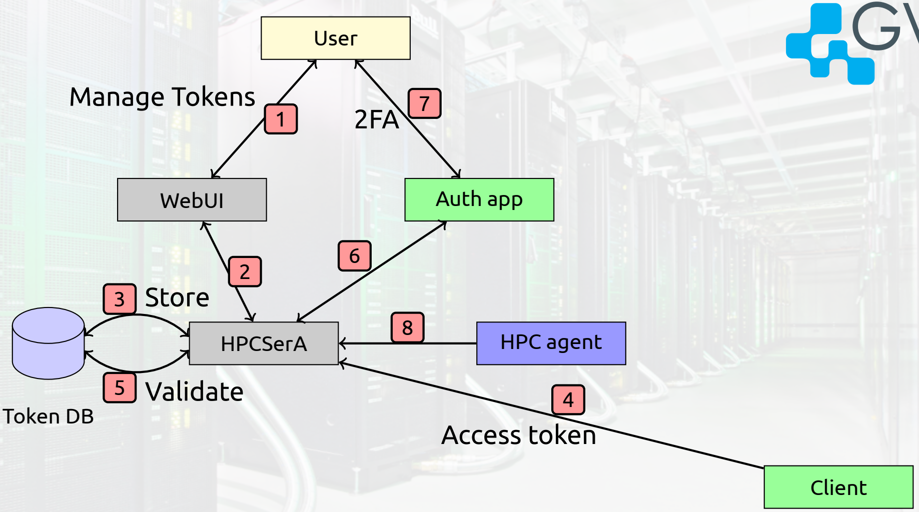

User
Manage Tokens
1
WebUI
2
3
Store
HPCSerA
Token DB
4
Access token
Client


User
Manage Tokens
1
WebUI
2
3
Store
HPCSerA
5
Validate
Token DB
4
Access token
Client

GWDG


User
Manage Tokens
1
WebUI
Auth app
2
6
3
Store
HPCSerA
Token DB
5
Validate
4
Access token
Client


User
Manage Tokens
1
2FA
7
WebUI
Auth app
2
6
3
Store
HPCSerA
5
Validate
Token DB
4
Access token
Client



User

Manage Tokens

1

2FA

7

WebUI

Auth app

2

6

3 Store

HPCSerA

8

HPC agent

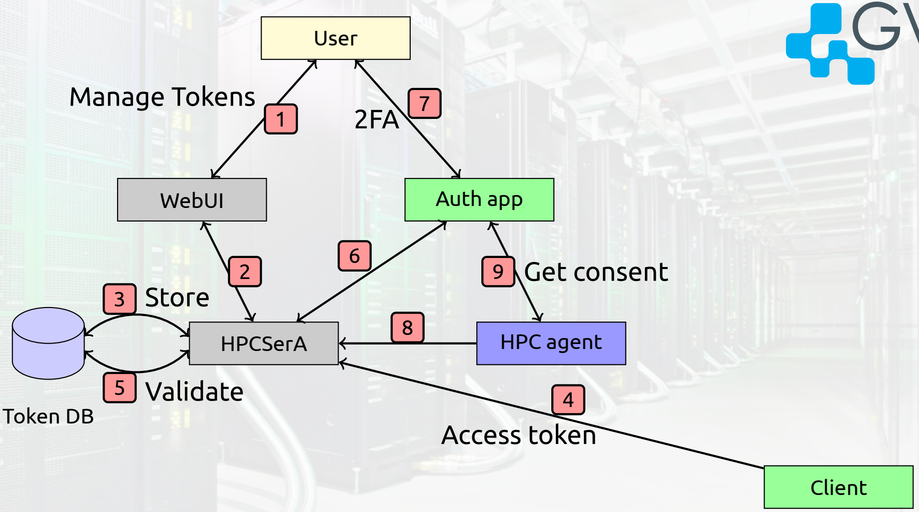
5 Validate

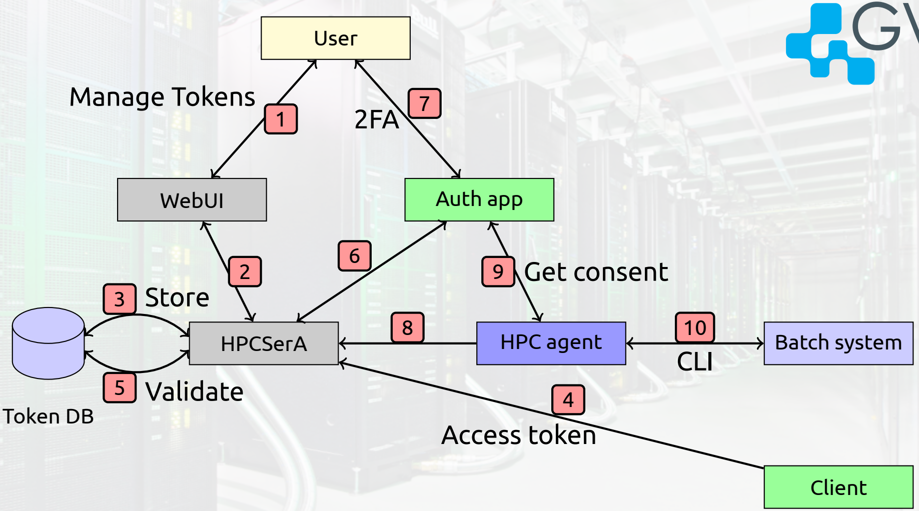
Token DB

4

Access token

Client


User
Manage Tokens
1
2FA
7
WebUI
Auth app
2
6
9
Get consent
3
Store
HPCSerA
8
HPC agent
5
Validate
Token DB
4
Access token
Client



User

Manage Tokens

1

2FA

7

WebUI

Auth app

2

6

9

Get consent

3

Store

HPCSerA

8

HPC agent

10

Batch system

CLI

5

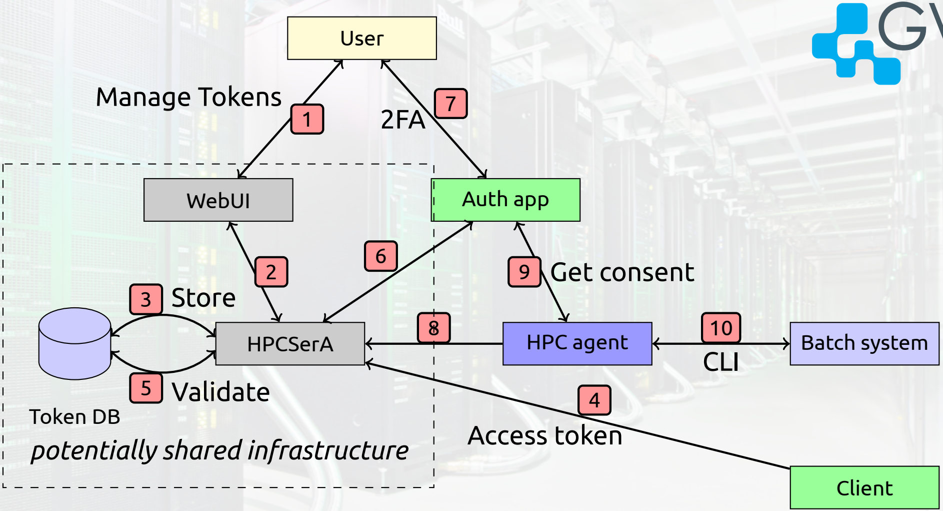
Validate

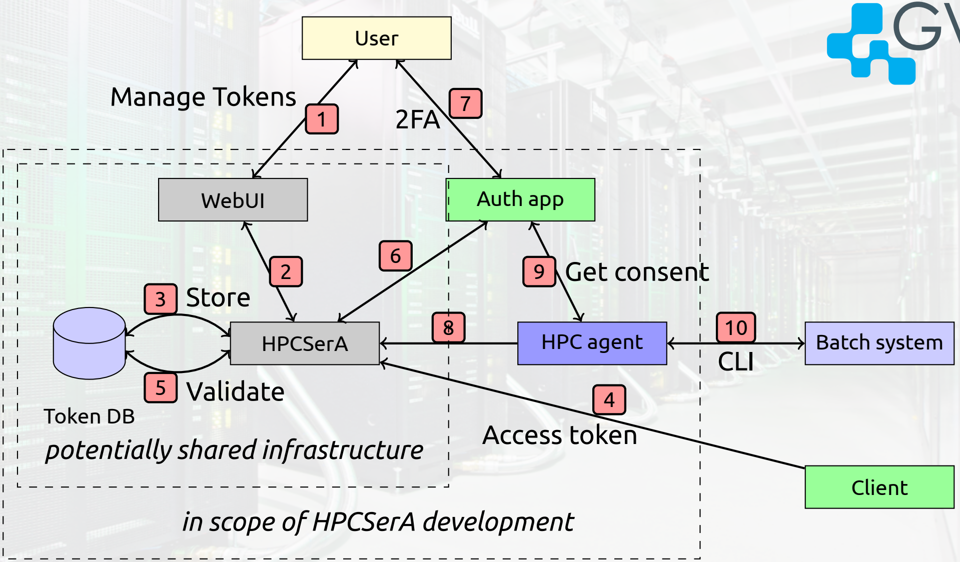
Token DB

4

Access token

Client


User
Manage Tokens
1
2FA
7
WebUI
Auth app
2
6
9
Get consent
3
Store
HPCSerA
8
HPC agent
10
Batch system
CLI
5
Validate
Token DB
4
Access token
potentially shared infrastructure
Client


User
Manage Tokens
1
2FA
7
WebUI
Auth app
2
6
9
Get consent
3
Store
HPCSerA
8
HPC agent
10
Batch system
CLI
5
Validate
Token DB
4
Access token
Client
potentially shared infrastructure
in scope of HPCSerA development

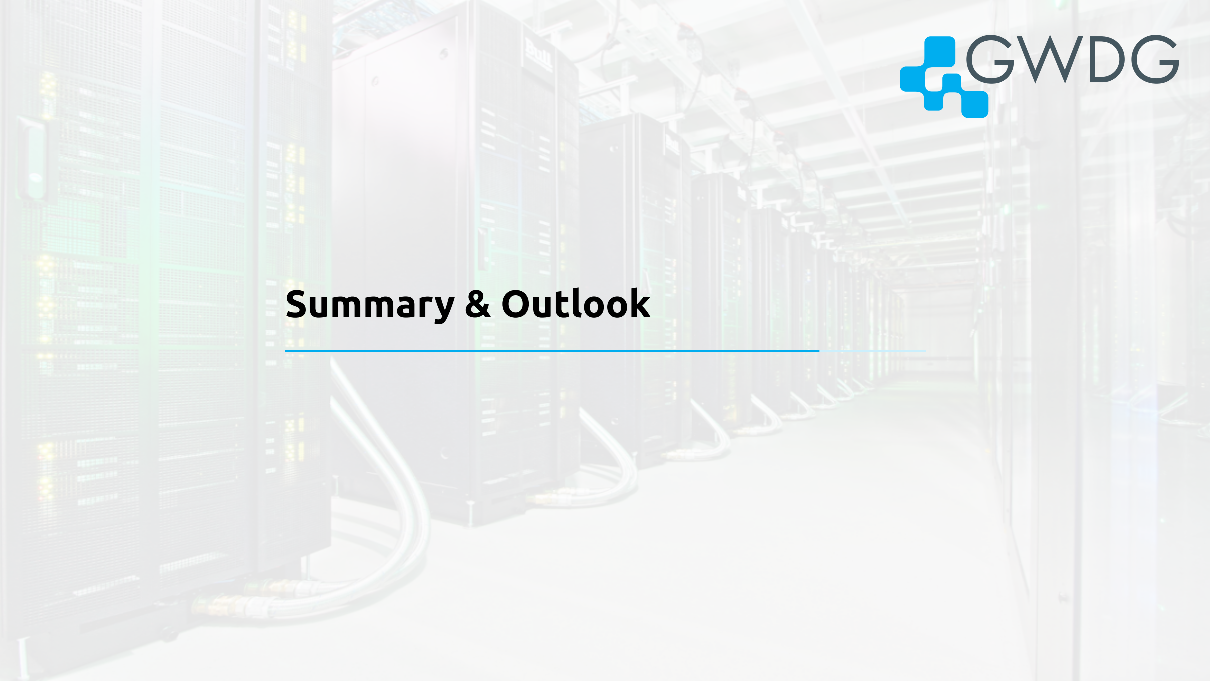
Trust and Access Control

- Decoupled OAuth
  - OAuth allows on-demand token creation
  - Clients can be headless → Use Decoupled Flow
  - Authorization is done out-of-band (web browser/mobile device)
- Multi-site setup
  - Individual HPCSerA deployments
    - Needed when not trusting other sites
    - Users access different endpoints (similar to S3)
  - Shared setup
    - Trusting central API instance, global API endpoint
    - **Clients** specify which **HPC agent** should run a job

Access Roles

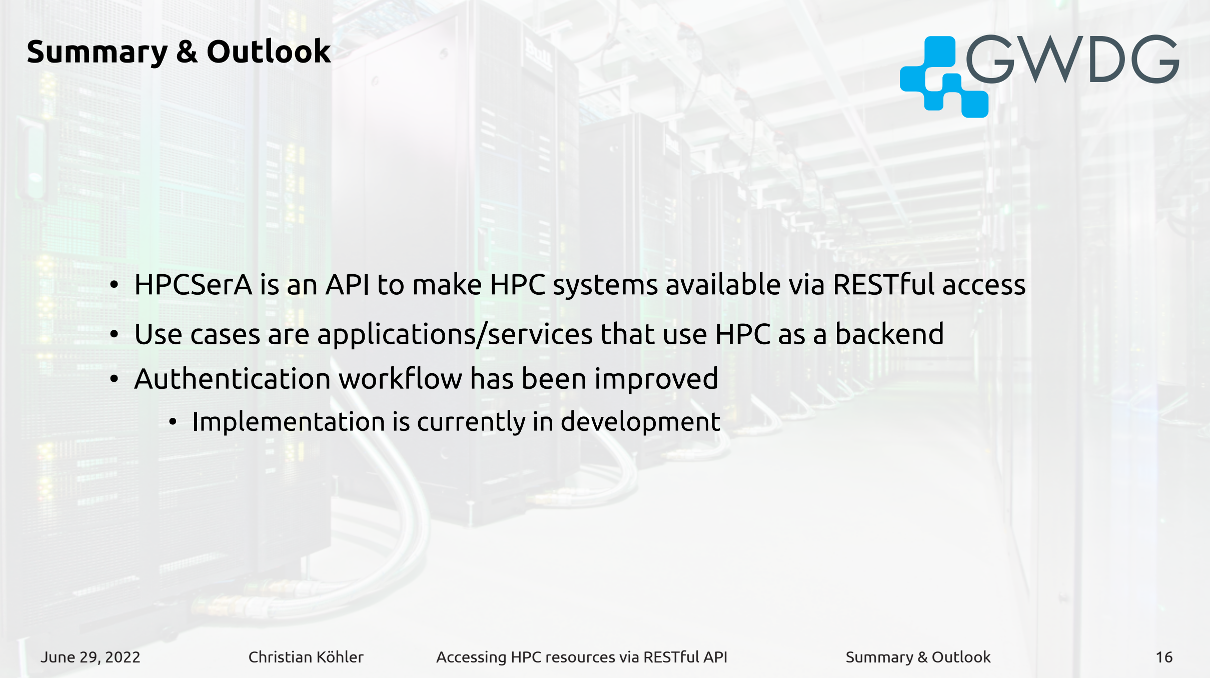
- Granted for each token and `user_id` via WebUI
- Also Specific to some `project_id` → segmentation of workflows
- Orthogonal roles that can be arbitrarily combined:

Role Number	Role	Description
1	GET_JobStatus	Client can retrieve information about a submitted job
2	UPDATE_JobStatus	Used by client/agent to update the job status
3	GET_Job	Endpoint used by the agent to retrieve job information
4	POST_Code	Client to ingest new code to the HPC sytsem
5	GET_Code	Agent pulls new code. Might be necessary to run new job
6	POST_Job	Client triggers parameterized job
7	UPDATE_Job	Client updates already triggered job
8	DELETE_Job	Client deletes already triggered job



Summary & Outlook

Summary & Outlook



- HPCSerA is an API to make HPC systems available via RESTful access
- Use cases are applications/services that use HPC as a backend
- Authentication workflow has been improved
    - Implementation is currently in development

Acknowledgements

- Based on joint work withWaqar Alamgir, **Sven Bingert**, **Mohammad Hossein Biniaz**, **Julian Kunkel** and **Hendrik Nolte**
- We gratefully acknowledge funding by the *Niedersächsisches Vorab* funding line of the Volkswagen Foundation and *Nationales Hochleistungsrechnen*.

→ **We'd like to hear about your potential use case!**

References

- S. Bingert, C. Köhler, H. Nolte, and W. Alamgir, “An API to Include HPC Resources in Workflow Systems,” in *INFOCOMP 2021, The Eleventh International Conference on Advanced Communications and Computation*, C.-P. Rückemann, Ed., 2021, pp. 15–20.

  `https://www.thinkmind.org/index.php?view=article&articleid=infocomp_2021_1_30_60014`

- M. H. Biniaz, S. Bingert, C. Köhler, H. Nolte, and J. Kunkel, “Secure Authorization for RESTful HPC Access,” submitted, accepted for *INFOCOMP 2022*.

- H. Nolte and P. Wieder, “Realising Data-Centric Scientific Workflows with Provenance-Capturing on Data Lakes,” Data Intelligence, pp. 1–13,03 2022.

  `https://doi.org/10.1162/dint_a_00141`