Advanced Security Services for Computer Simulation Research in Medicine

Jan Meizner[1], Marian Bubak[1], Daniel Harężlak[1], Tomasz Bartyński[1], Tomasz Gubala[1], Marek Kasztelnik[1], Maciej Malawski[1], Piotr Nowakowski[1]

[1]ACC Cyfronet AGH, Krakow, Poland
http://dice.cyfronet.pl/

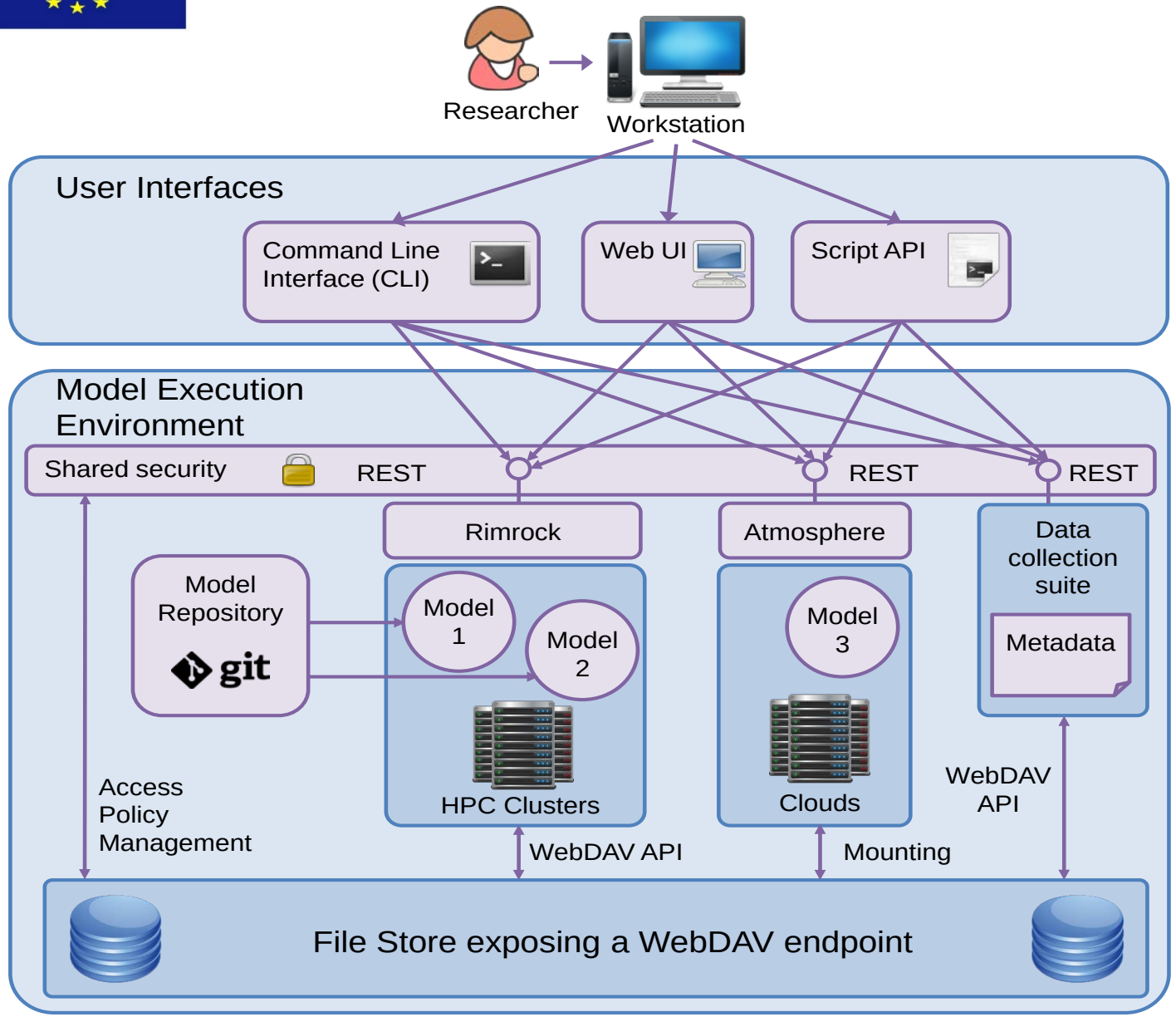
Motivation and objectives

Motivation:
providing proper security mechanisms for advanced medical simulations

To address the following critical security-related aspects:

- Authentication, authorization and accounting (AAA)
- Data security during processing and storage
- Mechanisms to ensure data cannot be recovered given reasonable time and resources, after being deleted

EurValve platform



API – Application Programming Interface

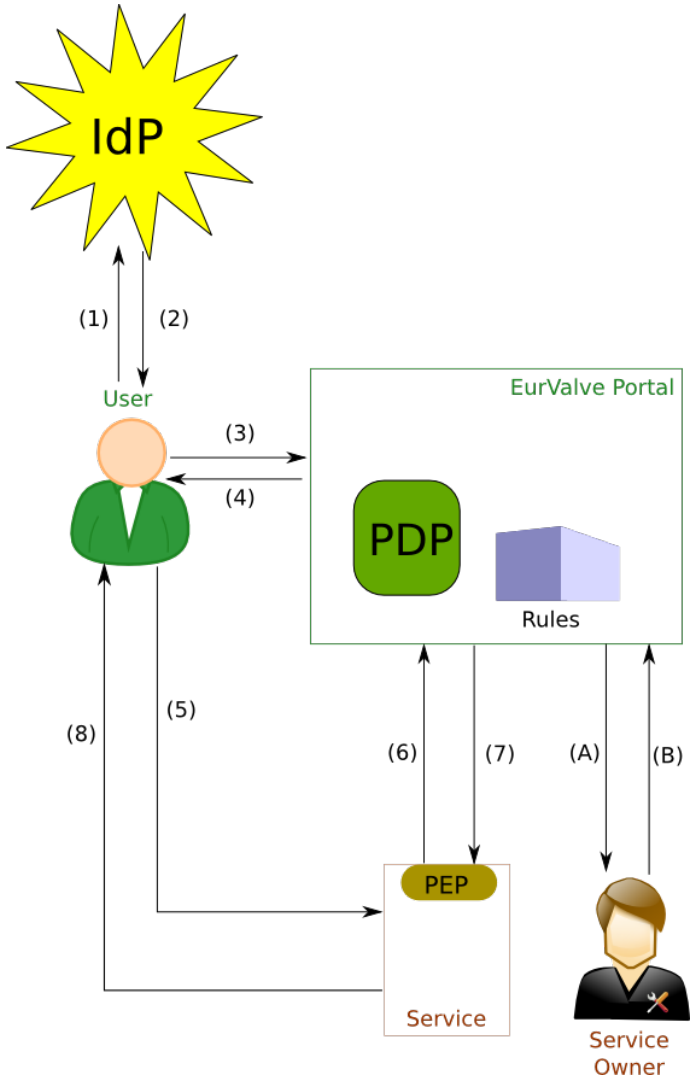
REST – Representational state transfer

Rimrock – service used to submit jobs to HPC cluster

Atmosphere – provides access to cloud resources

git – a distributed revision control system

AAA security use case



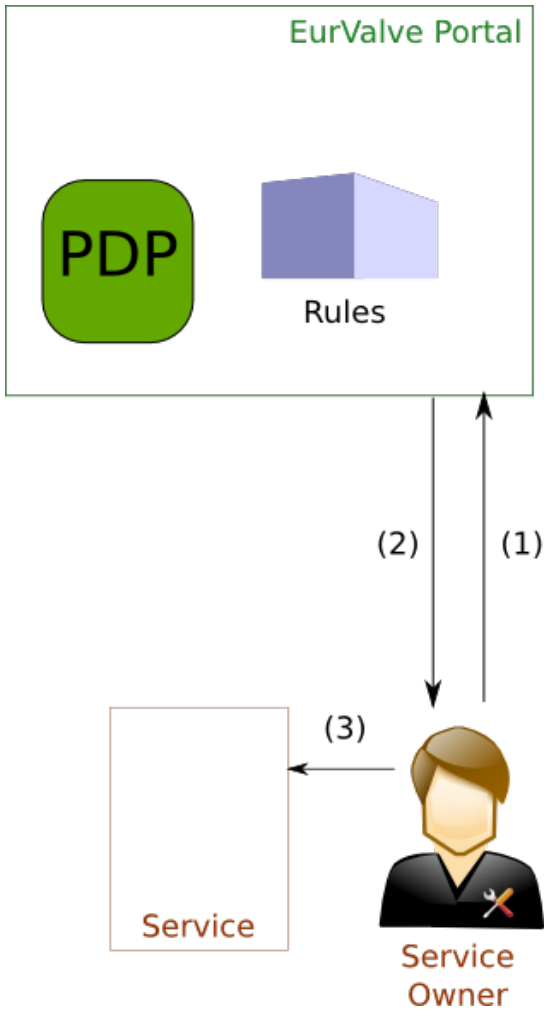
- **Step 1-2 (optional)**: Users authenticate themselves with the selected identity provider (hosted by the project or an external trusted IdP) and obtain a secure token which can then be used to authenticate requests to the MEE
- **Step 3-4**: User requests JWT token from the Portal, based on IdP or local authentication
- **Step 5** – User sends a request to a service (token attached)
- **Step 6-7** – Service PEP validates token and permissions against the PDP (authorization).
- **Step 8** – service replies with data or error (access denied)

Optional interaction by the service owner:

- **Step A-B** – Service Owner may modify policies for the PDP via:
  - the Portal GUI: **global** and **local**
  - API (e.g. from the Service): **local only**

IdP – Identity Provider
PDP – Policy Decision Point
JWT – JSON Web Token
PEP – Policy Enforcement Point

Registration of a new service



To secure a service its owner first needs to register it in the Portal/PDP.

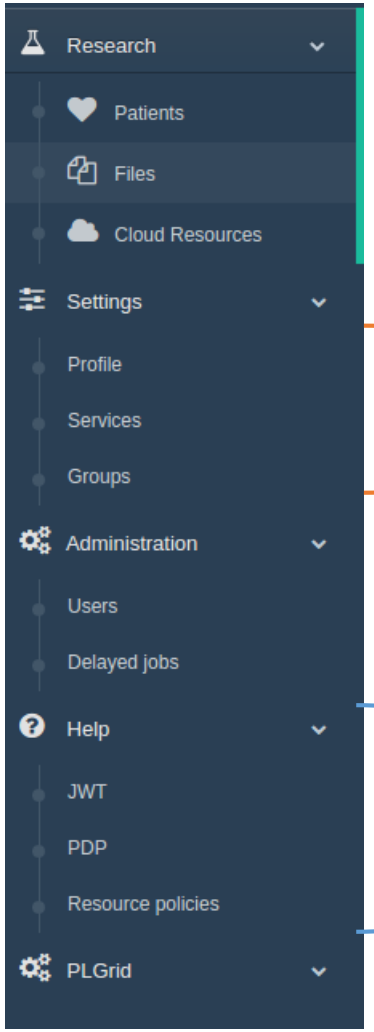
- **Step 1-2**: Service Owner logs into the Portal, creates the service and a set of Global Policies, and obtains a Service Token
- **Step 3**: Service Owner configures the service PEP to interact with the PDP (incl. setting the service token).

A standard PEP for Web-based services is provided by the DICE team.
Custom PEPs may be developed using the provided API.

The Service may use its token to:

- query the PDP for user access
- modify Local Policies for fine-grained access to the service

Security features in the MEE Portal



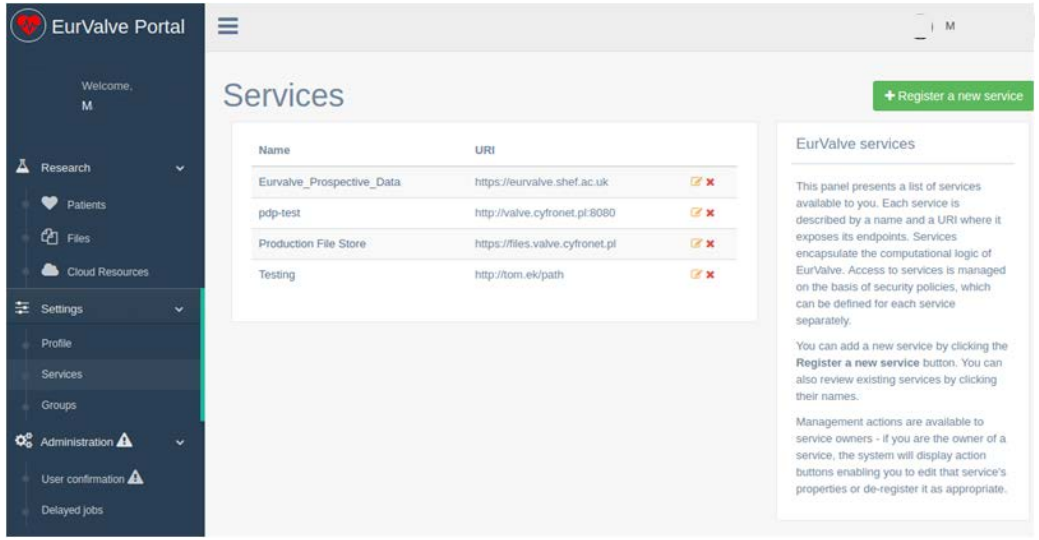
Security configuration

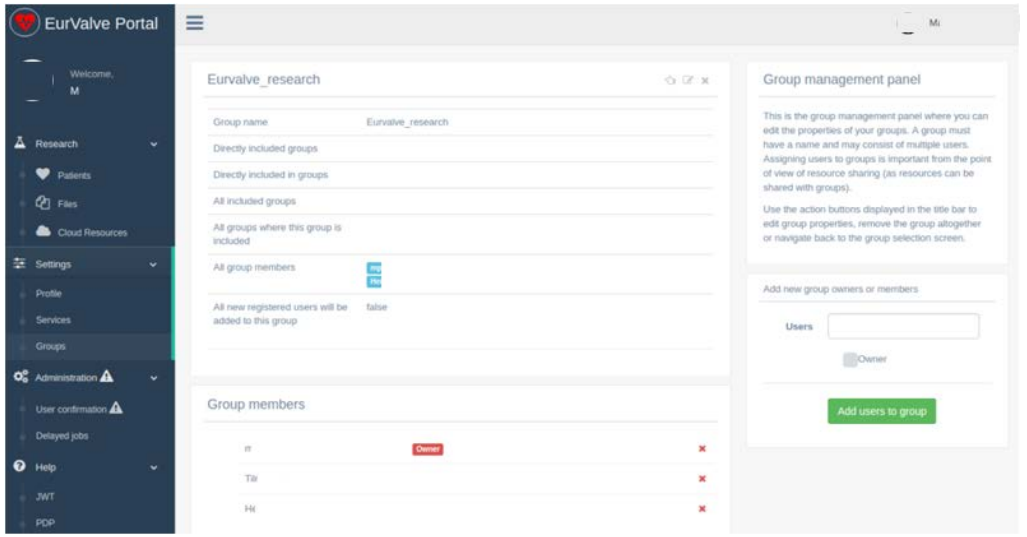
- Service management – for every service dedicated set of policy rules can be defined
- User Groups – can be used to define security constraints

REST API

- Creating a new user session – as a result, new JWT (JSON Web Token) tokens are generated for credential delegation
- PDP – Policy Decision Point: check if user has access to concrete resource
- Resource policies – add/remove/edit service security policies

Security management via the UI





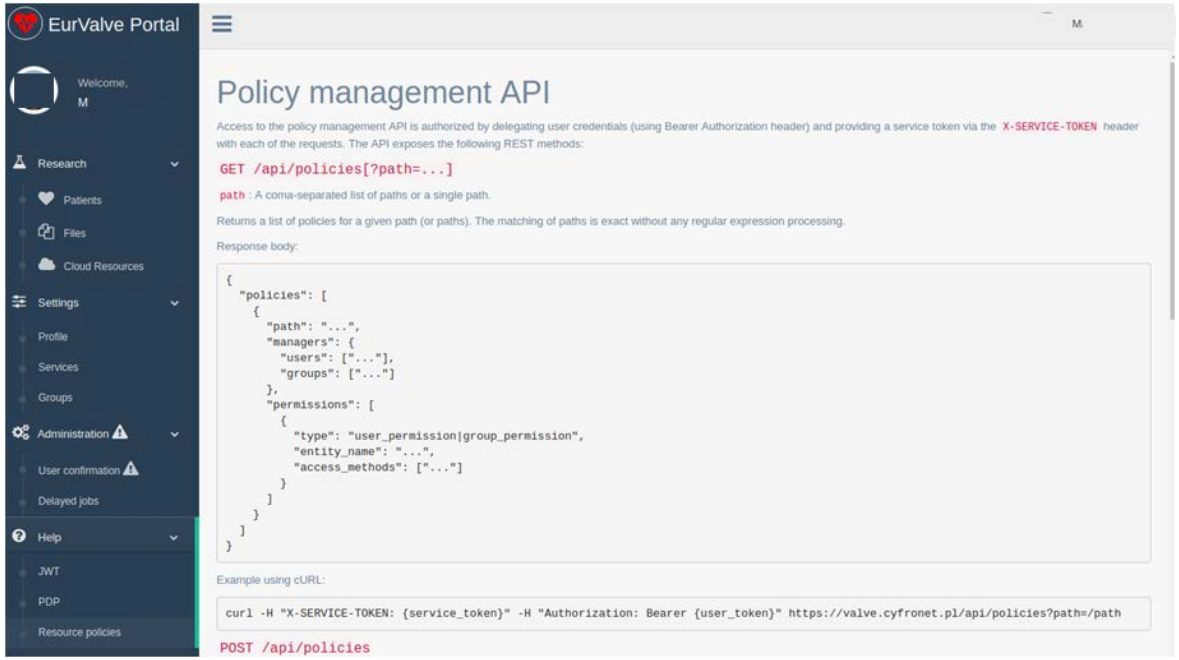
Services

- Basic security unit where dedicated security constraints can be defined
- Two types of security policies:
  - Global – can be defined only by service owner
  - Local – can be created by the service on the user's behalf

Groups

- Group users
- Dedicated portal groups:
  - Admin
  - Supervisor – users who can approve other users in the portal
- Generic groups:
  - Everyone can create a group
  - Groups can be used to define security constraints

Security management via REST API



Generate user JWT Token

- User (or other service) can generate new JWT token by passing username and password
- JWT token can be used for user credential delegations by external EurValve services

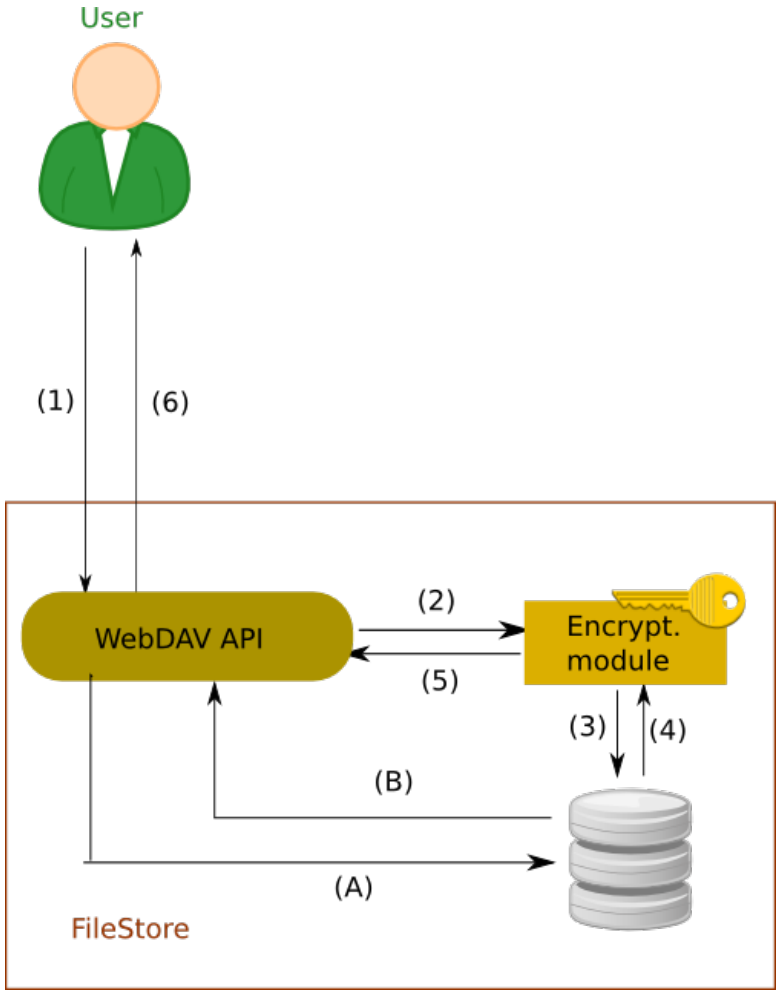
PDP API

- Check if user has right to access a specific resource

Resource policy management

- Create/edit/delete local policies by external EurValve service on user behalf
- Currently integrated with File Store
- Initial ArQ integration tests underway

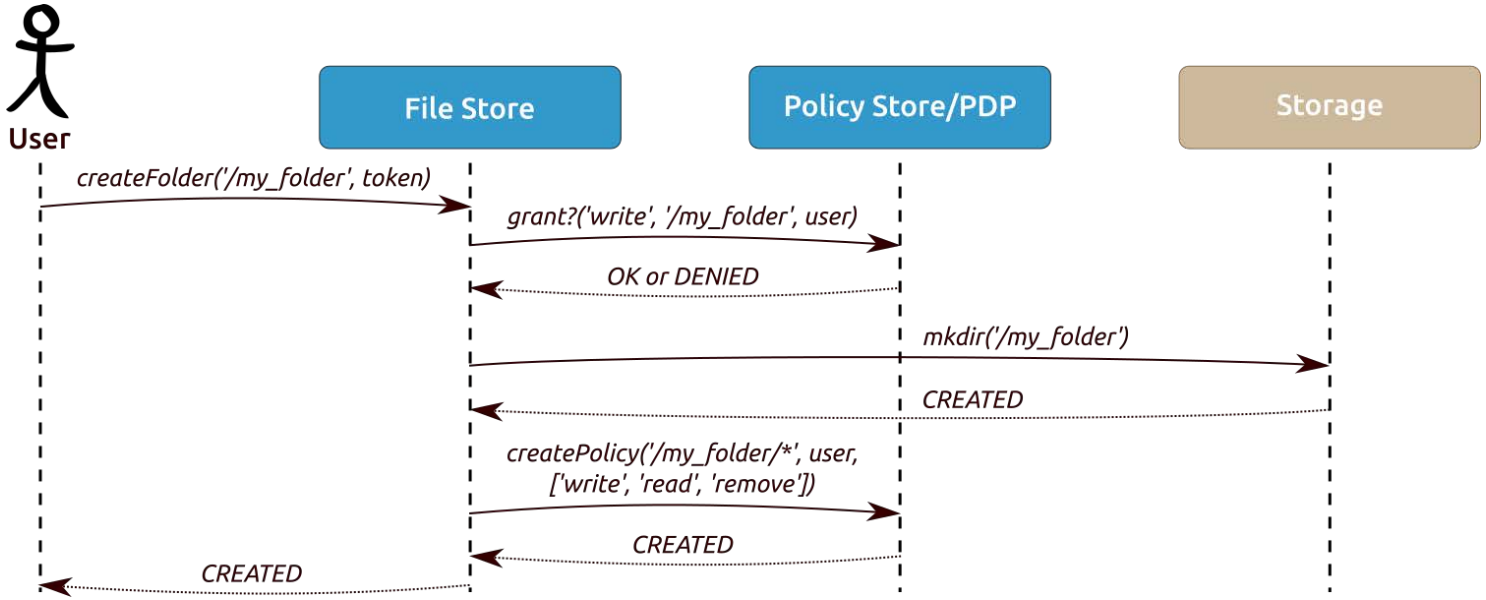
FileStore encryption use case



- BLOB Data handling:
  - Step 1 – data is sent via encrypted channel to the service
  - Step 2-3 – data encrypted and stored on disk
  - Step 4-5 – data decrypted and retrieved
  - Step A-B (optional) – data stored directly to disk
  - Step 6 (all) – data sent back to the user
- Currently all data is encrypted (steps 1-6)
  - It may be skipped if needed

Communication between the File Store and the Policy Store/PDP

- By default File Store can create top-level private folders
- Each File Store request is evaluated by a PDP on the basis of the requested action, resource path and user identifier
- Storage operations are performed only as allowed by the PDP
- When creating top-level folders a new policy is created, which grants write, read and remove permissions only to the user invoking the operation



Encryption performance (1/2)

- The benchmark evaluates the overhead of AES (Advanced Encryption Standard) encryption for the File Store based on various settings
- Results were used to find a compromise between speed and security for a given settings
- Benchmark scenario
  - Generate multiple input files with different sizes
  - Use customized prototype module to encrypt files and measure the overhead (no encryption, AES with 128, 192 and 256 bits keys)
  - Use the same module for decryption – also measure overhead
  - Compare decrypted data vs. input (validate the process)

Encryption performance (2/2)

- Benchmark environment:
  - **CPU:** Intel Core i7 2.3 GHz (4 cores)
  - **RAM:** 16GB DDR3
  - **OS:** Mac OS X 10.9
  - **Java:** 1.8.0_121
  - **Input:** 10 blocks of data 100 MB each (in memory, to avoid network overhead)

- Average speed for AES128
  - Encryption: 98.11 MB/s
  - Decryption: 91.02 MB/s

- Average speed for AES192
  - Encryption: 89.57 MB/s
  - Decryption: 84.25 MB/s

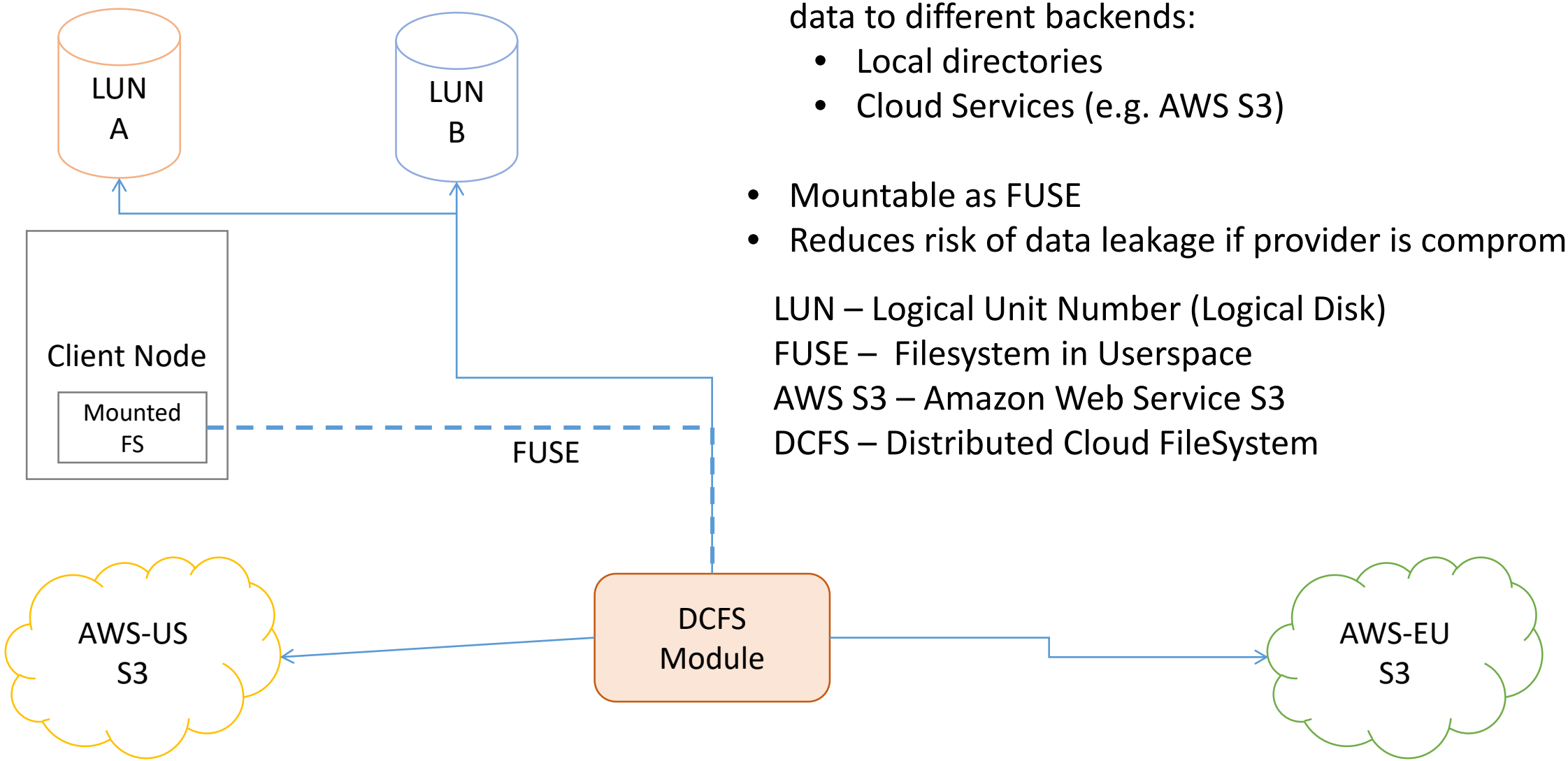
- **Average speed for AES256 (our choice for production)**
  - **Encryption: 87.94 MB/s**
  - **Decryption: 78.56 MB/s**

Data dispersal proof of concept

- Proof of Concept of the solution distributing chunks of data to different backends:
  - Local directories
  - Cloud Services (e.g. AWS S3)

- Mountable as FUSE
- Reduces risk of data leakage if provider is compromised

LUN – Logical Unit Number (Logical Disk)
FUSE – Filesystem in Userspace
AWS S3 – Amazon Web Service S3
DCFS – Distributed Cloud FileSystem



Summary and future work

- Security services has been integrated with the EurValve Model Execution Environment

- We have provided solutions for 2 main use cases:
  - Securing access to the MEE
  - Securing data stored in the FileStore

- Solution has been successfully validated and deployed in production

- We plan to:
  - Add advanced accounting mechanism
  - Consider extending data dispersal POC to make it production ready

EurValve H2020 Project 689617

http://www.eurvalve.eu

http://dice.cyfronet.pl