Access to services via SSH with OpenID Connect.

Aksieniia Shtimmerman
CNAF INFN
12-14 MAY 2025,

Motivation

- Using SSH access with OIDC tokens allows for federated access.
- Streamlines access management for users across various organizations and systems.
- Enhances security by leveraging dynamic, time-limited tokens.
- Facilitates centralized identity and access management (IAM).

Access to Services via SSH with OpenID Connect

OIDC-Agent supports agent forwarding over SSH, allowing the remote user to obtain tokens from the local agent. This is done by forwarding the Unix domain used to communicate with the agent. OpenID Connect (OIDC) offers a modern solution for managing federated identities, enhancing security and access management.

Objectives

- **Security**: Use OIDC access tokens to authenticate users.
- **Simplicity**: Avoid significant changes to existing SSH clients and servers.
- **Scalability**: Support integration with local user management systems.

Main Components

- **motley_cue:** Service for mapping OIDC identities to local identities.
- **pam-ssh-oidc:** Pluggable Authentication Module (PAM) that accepts OIDC access tokens.
- **mccli: Wrapper for the SSH client to connect to an OIDC-enabled SSH server.**

Server:
motley cue,: a service for mapping OIDC identities to local identities.
pam-ssh-oidc: a Pluggable Authentication Module (PAM) that accepts OIDC Access Tokens for authenticating users.

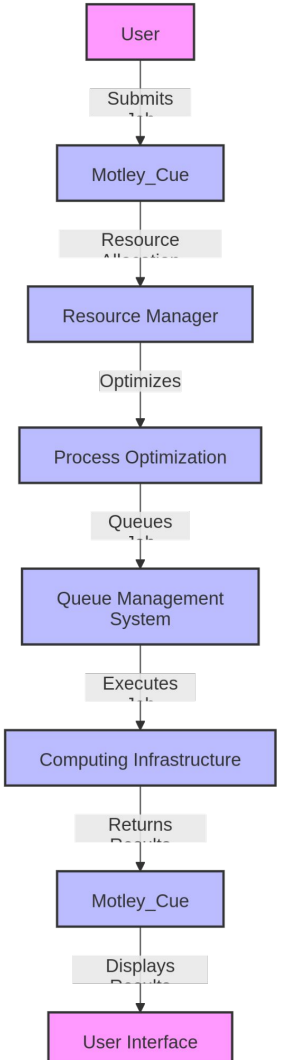
Client-Side
oidc-agent: Obtains OIDC access tokens.
mccli: Manages access tokens and communicates with motley_cue.

Motley_Cue
Introduction and Installation

Motley_Cue is a tool designed to simplify the management of job queues in high-performance computing environments.

It provides an intuitive interface for resource management and process optimization.

Supports various queue management systems, making it versatile and adaptable to different infrastructures.



This diagram show Motley_Cue operates.

Configuration

Install Motley cue - a service for mapping OIDC identities to local identities.

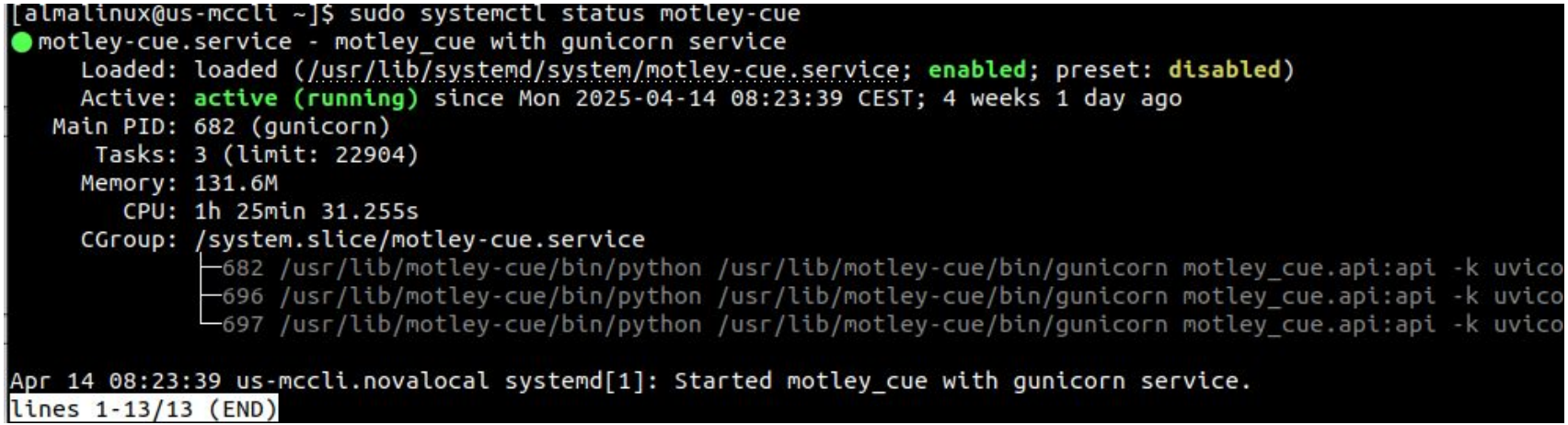
motley_cue works with Python 3 (>= 3.7), and only on Linux:

- Debian (testing + stable + oldstable)
- Ubuntu (22.04 + 20.04 + 18.04)
- Centos (7 + 8 + Stream)
- Rockylinux (8.5)
- OpenSuse (15.4, 15.5)

Packages are available at https://repo.data.kit.edu This guide explains how to install the server Motley cue on Almakinux9 distributions

Install motley_cue:

```
sudo dnf update
sudo dnf upgrade
dnf install python3-pip
cd /etc/yum.repos.d
wget https://repo.data.kit.edu//data-kit-edu-almalinux9.repo
install motley-cue
```



Configuration file /etc/motley_cue/motley_cue.conf:

Here we configure several different aspects:

Which OPs do we trust

Authorisation:

Which Virtual Organisations do we support
Which individual users do we support
The privacy statement to display
Auhorised security staff
The default self-documenting configuration file is shipped with the motley-cue installation.

```
[almalinux@us-mccli ~]$ ls /etc/motley_cue/
feudal_adapter.conf  motley_cue.env       motley_cue_old.conf  templates
motley_cue.conf      motley_cue_copy.conf  privacystatement.md
[almalinux@us-mccli ~]$ sudo vim /etc/motley_cue/motley_cue.conf
```

Configuration file /etc/motley_cue/feudal_adapter.conf:

Feudal adapter is the plugin-based tool that implements user provisioning. Aspects that are configured here include:

Minimally required levels of assurances
How to map remote users to local unix accounts
How to map VO-memberships to local unix groups
Which backend to use

```
### supported_groups
### method -- default: classic
method = classic

# deploy_user_ssh_keys -- default: yes. Allows using ssh keys, when they are found in the deployment request
deploy_user_ssh_keys = no

# The base URL of the bwidm API
url = https://iam-t1-computing.cloud.cnaf.infn.it/

### org_id - The ID for bwidm. This is used for prefixing user- and group names.
org_id = fdl

log_outgoing_http_requests = True

[backend.ldap]
# Configuration for the ldap backend
mode = read_only
host = ldap_server
group_base = ou=groups,dc=example
```

Configuration file /etc/nginx/conf.d/nginx.motley_cue.conf:

```
# You can uncomment the server block for use port 80 instead to 443 and use port 8443 instead of 443.
```

- The main server block listens on port 443 with SSL enabled.
- SSL certificates are specified for secure connections.
- Requests to the root (`/`) and `/oidc` paths are proxied to a local service running on port 8080.
- Various headers are set to ensure proper forwarding of the original request information.
- Error and access logs are specified for troubleshooting and monitoring.

Debugging and Troubleshooting Motley-cue

Identify the Problem:

motley_cue logs for any errors.
sudo journalctl -u motley_cue

sudo tail -f /var/log/auth.log

/var/log/nginx/motley_cue_error

/var/log/nginx/motley_cue_access.log.

/etc/nginx/conf.d/nginx.motley_cue.conf

https://mccli.readthedocs.io/en/latest/index.html

Ensure that environment variables set by `eval $(oidc-agent-service use)` are correctly configured and active

Install PAM module

This library uses authentication code flow and follows rfc7636 and rfc6749. User is prompted with a verification code and then receives authentication code which allows to call keycloak token endpoint and authenticate the user.

```
dnf install pam-ssh-oidc-autoconfig
```

Module PAM must be configured. The setup is different, depending on the linux distribution you use file sshd config `/etc/ssh/sshd_config one of the following enable:`

```
UsePam yes
# one of the following, depending on your version of OpenSSH:
ChallengeResponseAuthentication yes
KbdInteractiveAuthentication yes

PasswordAuthentication no
```

Restart sshd service:

```
systemctl restart sshd
```

To SSH into a server that supports OIDC authentication, you'll need install the client

The recommended way to install mccli is with pip:

```
pip install mccli
```

For a full description of the options, use the help option — also on each subcommand, as they might have additional options available:

```
mccli --help
mccli ssh --help
mccli info --help
```

Configure account in oidc-agent <your-mccli-host>:

Youo need to start the `oidc-agent` service and set the necessary environment variables for managing OpenID Connect (OIDC) tokens. Here's a breakdown of what it does: `$ eval $(oidc-agent-service use)`

add a new OIDC account configuration:
`oidc-add your-mccli-host`

```
shtimmerman at shtimmerman [master] [~/modules/user-support/mccli]
 $ eval $(oidc-agent-service use)
977441
shtimmerman at shtimmerman [master] [~/modules/user-support/mccli]
 $ oidc-gen -l
The following account configurations are usable:
argocd
ashtimmerman_token
mccli
mccli-t1-computing
mccli_test
pilota
wlcg
```

initializes the `oidc-agent` service:

```
oidc-agent-service use
```

For add OIDC account <your-mccli-host>:

For add *OIDC* account <mccli-t1-computing>:

```
% export OIDC_AGENT_ACCOUNT=you-account
% export ACCESS_TOKEN=$(oidc-token you-mccli-host
```

If your want unset token use command:

Connecting to an OIDC-capable SSH server:

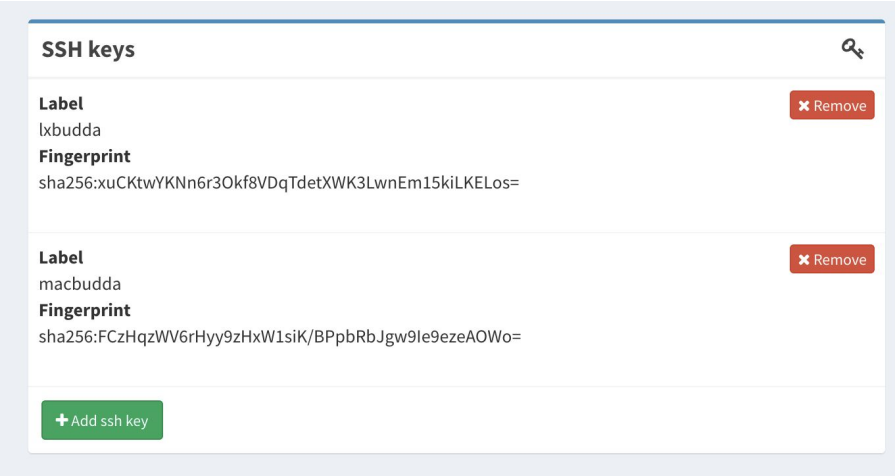
```
shtimmerman@shtimmerman:~$ mccli ssh 131.154.162.55
Last login:Mon May 12:12:34 2025 from 172.16.10.169
[user-support001@us-mccli ~]$
```

mccli info --oidc <oidc-agent account name>

Bonus

SSH with IAM

Two features meet…



- INDIGO-IAM allows any **user** to upload **ssh public keys** into its own account
  - public keys can be programmatically read via **IAM API**
- AuthorizedKeysCommand SSHD configuration keyword
  - Starting from OpenSSH 9.4/9.4p1 (2023-08-10)
  - **AlmaLinux 9** supports it
  - https://man.openbsd.org/sshd_config#AuthorizedKeysCommand
- Let's try to combine them:
  - IAM as the only source of AuthN/Z
  - UNIX account provisioning handled separately

$ man 5 sshd_config

AuthorizedKeysCommand

Specifies a program to be used to look up the user's public keys. The program must be owned by root, not writable by group or others and specified by an absolute path. Arguments to **AuthorizedKeysCommand** accept the tokens described in the TOKENS section. If no arguments are specified then the username of the target user is used.

The program should produce on standard output zero or more lines of authorized_keys output (see **AUTHORIZED_KEYS** in sshd(8)). **AuthorizedKeysCommand** is tried after the usual **AuthorizedKeysFile** files and will not be executed if a matching key is found there. By default, no **AuthorizedKeysCommand** is run.

In other words: set the command line that upon execution writes the user's public SSH keys on stdout

System setup – `sshd_config`

```
1 PubkeyAuthentication yes
2 AuthorizedKeysFile none
3 AuthorizedKeysCommand /etc/security/ssh-keys-command.sh %u
4 AuthorizedKeysCommandUser sshkey
5 Match User almalinux
6         AuthorizedKeysFile      .ssh/authorized_keys
```

- **AuthorizedKeysFile** is set to **none**, to avoid users set their keys in an alternate location
  - we want **IAM** as the **only source of AuthN/Z** for better security
- **AuthorizedKeysCommand** is set to the path of the script, followed by the %u placeholder that is replaced at execution time by the username that requests access
- **AuthorizedKeysCommandUser** is set to **sshkey** system user, to avoid running as **root**
- Override rule allows an admin user to access the server also via authorized keys file
  - we don’t want to lock out the admins if IAM goes down

System setup - user and group

“sshkey” is a unprivileged system **account** and **group**

```
$ getent group sshkey

sshkey:x:1005:

$ getent passwd sshkey

sshkey:x:1004:1005::/home/sshkey:/sbin/nologin
```

System setup - permissions and SELinux

- **SELinux** is **Enforcing**!
- after installing the script, “restorecon -v /etc/security/ssh-keys-command.sh”
- the **sshkey** group can read and execute the script
- the owner of the script is **root**

```
$ getenforce

Enforcing

$ ls -lhZ /etc/security/ssh-keys-command.sh

-rwxr-x---. 1 root sshkey unconfined_u:object_r:etc_t:s0 724 Apr  4 15:19 /etc/security/ssh-keys-command.sh
```

$ cat /etc/security/ssh-keys-command.sh

```
#!/bin/bash

ISSUER=https://iam-dev.cloud.cnaf.infn.it
CLIENT_ID=REDACTED
CLIENT_SECRET=REDACTED

USERNAME=$1

TOKEN=$(curl -s -L -X POST \
 -d "grant_type=client_credentials" \
 -d "client_id=$CLIENT_ID" \
 -d "client_secret=$CLIENT_SECRET" \
 -d "audience=$ISSUER" \
 -d "scopes=iam:admin.read" \
 $ISSUER/token | jq -r .access_token
)

curl -s -L -H "Authorization: Bearer ${TOKEN}"
$ISSUER/iam/account/find/byusername?username=$USERNAME | jq -r
'select(([.Resources[0].groups[].display] | index("users") != null) and .Resources[0].active) |
.Resources[0]."urn:indigo-dc:scim:schemas:IndigoUser".sshKeys[].value '
```

$ cat /etc/security/ssh-keys-command.sh

```
#!/bin/bash

ISSUER=https://iam-dev.cloud.cnaf.infn.it
CLIENT_ID=REDACTED
CLIENT_SECRET=REDACTED

USERNAME=$1

TOKEN=$(curl -s -L -X POST \
 -d "grant_type=client_credentials" \
 -d "client_id=$CLIENT_ID" \
 -d "client_secret=$CLIENT_SECRET" \
 -d "audience=$ISSUER" \
 -d "scopes=iam:admin.read" \
 $ISSUER/token | jq -r .access_token
)

curl -s -L -H "Authorization: Bearer ${TOKEN}"
$ISSUER/iam/account/find/byusername?username=$USERNAME | jq -r
'select(([.Resources[0].groups[].display] | index("users") != null) and .Resources[0].active) |
.Resources[0]."urn:indigo-dc:scim:schemas:IndigoUser".sshKeys[].value '
```

- Environment variables set the issuer, client ID and secret
- An access token is requested to IAM via the token endpoint
  - the grant type is client_credentials
  - scopes are set to “iam:admin.read”
- The list of SSH public keys of the specific user are downloaded if both conditions are met:
  - the account is active (not deactivated/banned)
  - the user is in the “users” IAM group

It works

```
 $ ssh cpellegr@ui

Last login: Tue May 13 23:23:20 2025 from XXX.YYY.ZZZ.WWW

[cpellegr@ui ~]$ ls .ssh/authorized_keys*

ls: cannot access '.ssh/authorized_keys*': No such file or
directory
```

Access latency

Each time the user tries to SSH into the server:

- an access token is requested
- the SSH keys are downloaded from IAM

```
 $ time ssh cpellegr@ui true
real    0m3.091s
user    0m0.238s
sys     0m0.587s
```

- latency issues can be addressed by caching the access token and/or the ssh keys
- few seconds when starting interactive work could anyway be acceptable