

# Slurm's REST API

Nathan Rini

Slurm User Group 2023



# Questions?

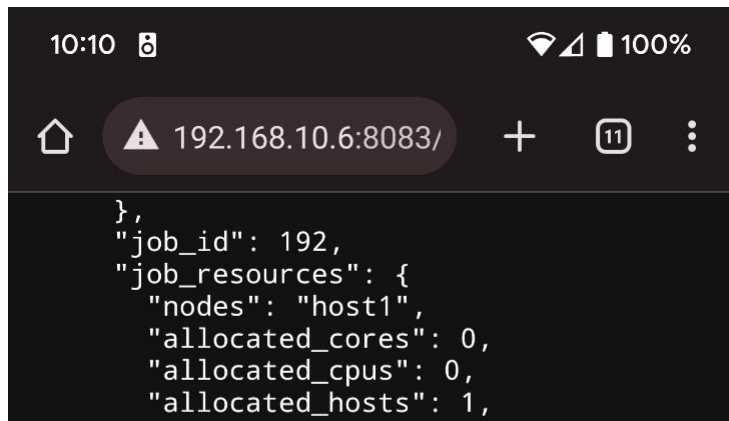
- Please feel free to interrupt at anytime with questions.
  - I will ask you wait if the question is answered later in the presentation.
- Comments and constructive complaints are always welcome
  - I may ask to defer discussion to finish the presentation on time or if question is not applicable to other sites.
- If you don't get your questions answered or you have more follow up questions, please open a ticket or find me after.
- Your feedback on slurmrestd matters to us and helps us with the future roadmap.

# Intro to slurmrestd

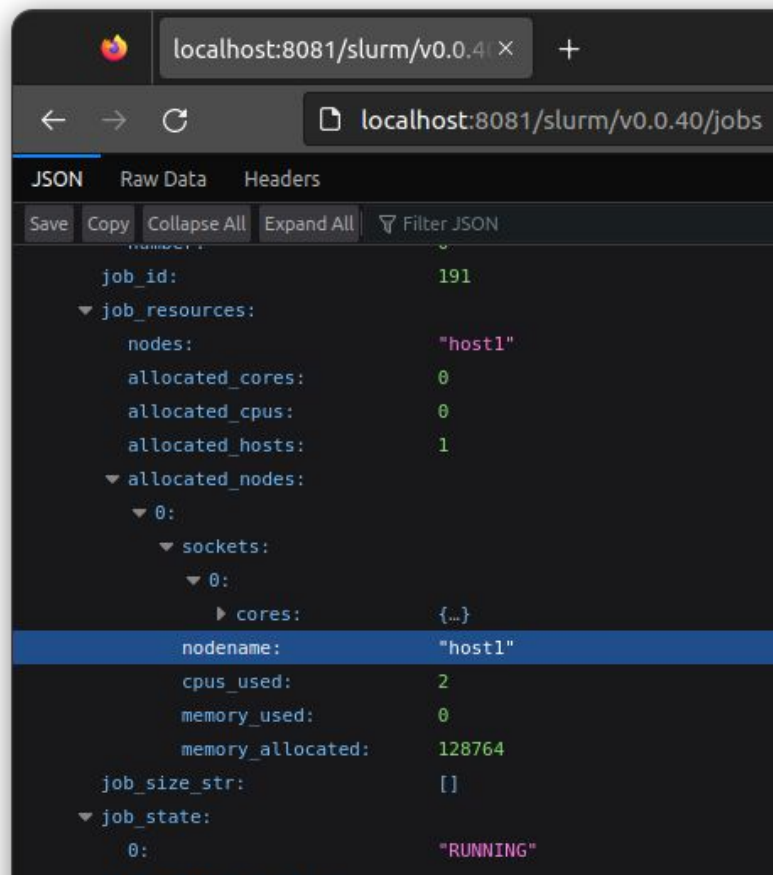
# What is the Slurm REST API?

Short answer:

**Slurm without command line**



```
10:10 100%
192.168.10.6:8083/
},
"job_id": 192,
"job_resources": {
  "nodes": "host1",
  "allocated_cores": 0,
  "allocated_cpus": 0,
  "allocated_hosts": 1,
```



```
localhost:8081/slurm/v0.0.40/jobs
localhost:8081/slurm/v0.0.40/jobs
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
job_id: 191
job_resources:
  nodes: "host1"
  allocated_cores: 0
  allocated_cpus: 0
  allocated_hosts: 1
  allocated_nodes:
    0:
      sockets:
        0:
          cores: {...}
          nodename: "host1"
          cpus_used: 2
          memory_used: 0
          memory_allocated: 128764
  job_size_str: []
  job_state:
    0: "RUNNING"
```

# What is the Slurm REST API?

- Slightly longer answer:
  - Allows users to query Slurm via HTTP requests (AKA [restful API](#))
    - Supports data formatted as [JSON](#) or [YAML](#)
  - [OpenAPI v3](#) (aka [Swagger v3](#)) compliant to allow sites to [easily generate clients](#)
  - [JSON Web Token \(JWT\)](#) authentication for clients outside of [MUNGE security perimeter](#)
    - Allowing for partially trusted clients or using site authentication service
- Exhaustive answers with live demos can be covered during Slurm onsite trainings:
  - Please email [sales@schedmd.com](mailto:sales@schedmd.com) to setup a training session.

# Example use cases via CLI

## Preparation: Setup shell

- Use a bash function to make calling curl easier
  - Example to query slurmrestsd via TCP socket:

```
function rest()
{
    path=$1
    shift
    unset SLURM_JWT
    export $(scontrol token lifespan=10)
    curl -s -H "X-SLURM-USER-TOKEN:${SLURM_JWT}" "http://localhost:8090/${path}"
"$@";
}
```

- User name is only required to act as a proxy, otherwise user encoded in token is used:
  - `-H "X-SLURM-USER-NAME:$(whoami)"`

# Query jobs information

- Get job\_id of all jobs known to slurmctld:

```
$ rest slurm/v0.0.40/jobs | jq -r '.jobs[].job_id'  
193  
194
```

- Get state of first Array Job task with state of all jobs known to slurmctld:

```
$ rest slurm/v0.0.40/job/194_1 | jq -r '.jobs[].job_state[]'  
PENDING
```

- Get total number of tasks of all running jobs:

```
$ rest slurm/v0.0.40/jobs | jq -r '.jobs[] | select(.job_state[] == "RUNNING") | .tasks.number'  
| awk '{ sum += $1; } END { print sum; }'  
10
```



## Example Job description

- job.json:

```
{
  "script": "#!/bin/bash\nsrun uptime",
  "job": {
    "environment": [ "PATH=/bin/./usr/bin/./sbin/" ],
    "account": "test",
    "name": "test slurmrestd job",
    "memory_per_node": { "set": true, "number": 100 },
    "tasks": 5,
    "nodes": "2-10"
  }
}
```

## Example Array Job description

- array\_job.json:

```
{
  "script": "#!/bin/bash\nsrun uptime",
  "job": {
    "environment": [ "PATH=/bin/./usr/bin/./sbin/" ],
    "account": "test",
    "array": "100",
    "name": "test slurmrestd array job",
    "memory_per_node": { "set": true, "number": 100 },
    "tasks": 5,
    "nodes": "2-10"
  }
}
```

## Example HetJob description

- het\_job.json:

```
{
  "script": "#!/bin/bash\nsrun uptime",
  "jobs": [
    {
      "environment": [ "PATH=/bin/./usr/bin/./sbin/" ],
      "account": "test",
      "name": "test slurmrestd job",
      "memory_per_node": { "set": true, "number": 100 },
      "tasks": 5,
      "nodes": "2-10"
    },
    { "memory_per_node": { "set": true, "number": 15 }, "tasks": 1, "nodes": 1,
      "environment": [ "PATH=/bin/./usr/bin/./sbin/" ] },
    { "nodes": 1, "environment": [ "PATH=/bin/./usr/bin/./sbin/" ] }
  ]
}
```

## Submit example jobs

```
$ rest slurm/v0.0.40/job/submit -H 'Content-Type:application/json' --data-binary @job.json  
| jq -r '.result.job_id'  
231
```

```
$ rest slurm/v0.0.40/job/submit -H 'Content-Type:application/json' --data-binary  
@array_job.json | jq -r '.result.job_id'  
232
```

```
$ rest slurm/v0.0.40/job/submit -H 'Content-Type:application/json' --data-binary  
@het_job.json | jq -r '.result.job_id'  
233
```

# Control Jobs

- Cancel a job:

```
$ rest slurm/v0.0.40/job/236 -X DELETE
```

- Signal a job with SIGINT:

```
$ rest slurm/v0.0.40/job/236?signal=SIGINT -X DELETE
```

- Change number tasks in a job:

```
$ rest slurm/v0.0.40/job/239 -H 'Content-Type:application/json' --data-binary '{"tasks": "10"}
```

Make sure to always call `--data-binary` and not `--data` when using curl to avoid the payload being corrupted.

# Example use cases via Python Client

# Preparation: Compile and install generated OpenAPI client

- [Install openapi-generator-cli following procedure first](#)
- Compile and install library for client

```
$ slurmrestd unix:slurmrestd.socket -s slurmctld,slurmdbd -d v0.0.40 &  
$ curl --unix-socket slurmrestd.socket 'http://host/openapi/v3' > openapi.json  
$ kill %1  
$ openapi-generator-cli generate -i openapi.json -g python -o py_api_client  
$ virtualenv test  
$ source test/local/bin/activate  
$ cd py_api_client/  
$ pip install -r requirements.txt .
```

## Preparation: start and configure python interactive

- Run python3 in interactive mode and setup environment for all examples:

```
$ python3
from pprint import pprint
import openapi_client
import subprocess
import os
import re
from openapi_client.apis.tags.slurm_api import SlurmApi
from openapi_client.apis.tags.slurmdb_api import SlurmdbApi
from openapi_client import ApiClient as Client
from openapi_client import Configuration as Config
c = Config()
c.host = "http://localhost:8080/"
c.access_token = subprocess.run(['scontrol', 'token', 'lifespan=9999'], check=True,
capture_output=True, text=True).stdout.replace('SLURM_JWT=', '').replace('\n', '')
slurm = SlurmApi(Client(c))
slurmdb = SlurmdbApi(Client(c))
```



# Inspection of generated OpenAPI client

- HTTP methods are implemented as functions in slurm and slurmdb objects:

```
print([a for a in dir(slurmdb) if re.match(r'slurmdb', a)])
['slurmdb_v0040_delete_account', 'slurmdb_v0040_delete_association',
'slurmdb_v0040_delete_associations', 'slurmdb_v0040_delete_cluster',
'slurmdb_v0040_delete_single_qos', 'slurmdb_v0040_delete_user',
'slurmdb_v0040_delete_wckey', 'slurmdb_v0040_get_account',
'slurmdb_v0040_get_accounts', 'slurmdb_v0040_get_association',
'slurmdb_v0040_get_associations', 'slurmdb_v0040_get_cluster',
...(truncated)...

print([a for a in dir(slurm) if re.match(r'slurm', a)])
['slurm_v0040_delete_job', 'slurm_v0040_delete_node', 'slurm_v0040_get_diag',
'slurm_v0040_get_job', 'slurm_v0040_get_jobs', 'slurm_v0040_get_licenses',
'slurm_v0040_get_node', 'slurm_v0040_get_nodes', 'slurm_v0040_get_partition',
'slurm_v0040_get_partitions', 'slurm_v0040_get_ping', 'slurm_v0040_get_reservation',
'slurm_v0040_get_reservations', 'slurm_v0040_get_shares', 'slurm_v0040_post_job',
'slurm_v0040_post_job_submit', 'slurm_v0040_post_node']
```

# Query jobs information

- Get job\_id of all jobs known to slurmctld:

```
jobs = slurm.slurm_v0040_get_jobs()
for job in jobs.body['jobs']:
    print(job['job_id'])
```

- Get state of first Array Job task with state of all jobs known to slurmctld:

```
jobs = slurm.slurm_v0040_get_jobs()
for job in jobs.body['jobs']:
    for state in job['job_state']:
        print(state)
```

# Query jobs information

- Get total number of tasks of all running jobs:

```
jobs = slurm.slurm_v0040_get_jobs()
c=0
for job in jobs.body['jobs']:
    if 'RUNNING' in job['job_state']:
        c += job['tasks']['number']
print(c)
```

## Example Job description

- Submit example job:

```
from openapi_client.model.v0040_job_submit_req import V0040JobSubmitReq
from openapi_client.model.v0040_job_desc_msg import V0040JobDescMsg

job = V0040JobSubmitReq(script='#!/bin/bash\nsruntime',
job=V0040JobDescMsg(partition='debug',name='test
job',environment=['PATH=/bin/./sbin/./usr/bin/./usr/sbin/'],current_working_directory='/tmp/'))

print(slurm.slurm_v0040_post_job_submit(body=job).body['result']['job_id'])
```

## Example Array Job description

- Submit example job:

```
from openapi_client.model.v0040_job_submit_req import V0040JobSubmitReq
from openapi_client.model.v0040_job_desc_msg import V0040JobDescMsg

job = V0040JobSubmitReq(script='#!/bin/bash\nsrun uptime',
job=V0040JobDescMsg(array='100',partition='debug',name='test
job',environment=['PATH=/bin/./sbin/./usr/bin/./usr/sbin/'],current_working_directory='/tmp/'))

print(slurm.slurm_v0040_post_job_submit(body=job).body['result']['job_id'])
```

# Example HetJob description

- Submit example job:

```
from openapi_client.model.v0040_job_submit_req import V0040JobSubmitReq
from openapi_client.model.v0040_job_desc_msg import V0040JobDescMsg
from openapi_client.model.v0040_uint32_no_val import V0040Uint32NoVal

job = V0040JobSubmitReq(script='#!/bin/bash\nsrun uptime',
jobs=[V0040JobDescMsg(partition='debug',name='test
job',environment=['PATH=/bin/./sbin/./usr/bin/./usr/sbin/'],memory_per_node=V0040Uint32
NoVal(set=True,number=100),tasks=5,nodes='2-10'),
V0040JobDescMsg(memory_per_node=V0040Uint32NoVal(set=True,number=100),tasks=
1,nodes='1',environment=['PATH=/bin/./sbin/./usr/bin/./usr/sbin/']),V0040JobDescMsg(node
s='1',environment=['PATH=/bin/./sbin/./usr/bin/./usr/sbin/'])])

print(slurm.slurm_v0040_post_job_submit(body=job).body['result']['job_id'])
```

# Control Jobs

- Cancel a job:

```
print(slurm.slurm_v0040_delete_job(path_params={'job_id': '3694'}).response.reason)
```

- Signal a job with SIGINT:

```
print(slurm.slurm_v0040_delete_job(path_params={'job_id': '3694'},query_params={'signal': 'SIGINT'}).response.reason)
```

# Control Jobs

- Change number tasks in a job:

```
from openapi_client.model.v0040_job_submit_req import V0040JobSubmitReq
from openapi_client.model.v0040_job_desc_msg import V0040JobDescMsg
from openapi_client.model.v0040_uint32_no_val import V0040Uint32NoVal

job = V0040JobDescMsg(name='updated test
job',environment=['PATH=/bin/:/sbin/:/usr/bin/:/usr/sbin/'],priority=V0040Uint32NoVal(set=True,number=0))

print(slurm.slurm_v0040_post_job(path_params={'job_id':'3697'},body=job).response.reason)
```



# Slurm CLI: YAML & JSON

# JSON and YAML for the command line

- Functionality from slurmrestd has been added to existing CLI commands to provide JSON and YAML output.
- Following commands (at least partially) support new output formats:

sshare -json	sshare -yaml
sacct -json	sacct -yaml
sacctmgr -json	sacctmgr -yaml
scontrol -json	scontrol -yaml
sdiag -json	sdiag -yaml
sinfo -json	sinfo -yaml
sprio -json	sprio -yaml
squeue -json	squeue -yaml

# Query jobs information

- Get job\_id of all jobs known to slurmctld:

```
$ squeue -json | jq -r '.jobs[].job_id'  
193  
194
```

- Get state of first Array Job task with state of all jobs known to slurmctld:

```
$ scontrol show job -json 194_1 | jq -r '.jobs[].job_state[]'  
PENDING
```

- Get total number of tasks of all running jobs:

```
$ scontrol show jobs -json | jq -r '.jobs[] | select(.job_state[] == "RUNNING") |  
.tasks.number' | awk '{ sum += $1; } END { print sum; }'  
10
```

## Selecting data\_parser plugin version (23.11+)

- CLI commands
  - `-yaml/-json` without an argument defaults to latest version (v0.0.40 on 23.11)

```
sinfo --json=v0.0.39
sinfo --json=v0.0.40
sinfo --json
```

```
sinfo --yaml=v0.0.39
sinfo --yaml=v0.0.40
sinfo --yaml
```

- `slurmrestd`
  - `slurmrestd` supports loading multiple `data_parser` plugins at once
    - 23.02:

```
slurmrestd -d v0.0.39 -s v0.0.39,dbv0.0.39
```

- 23.11:

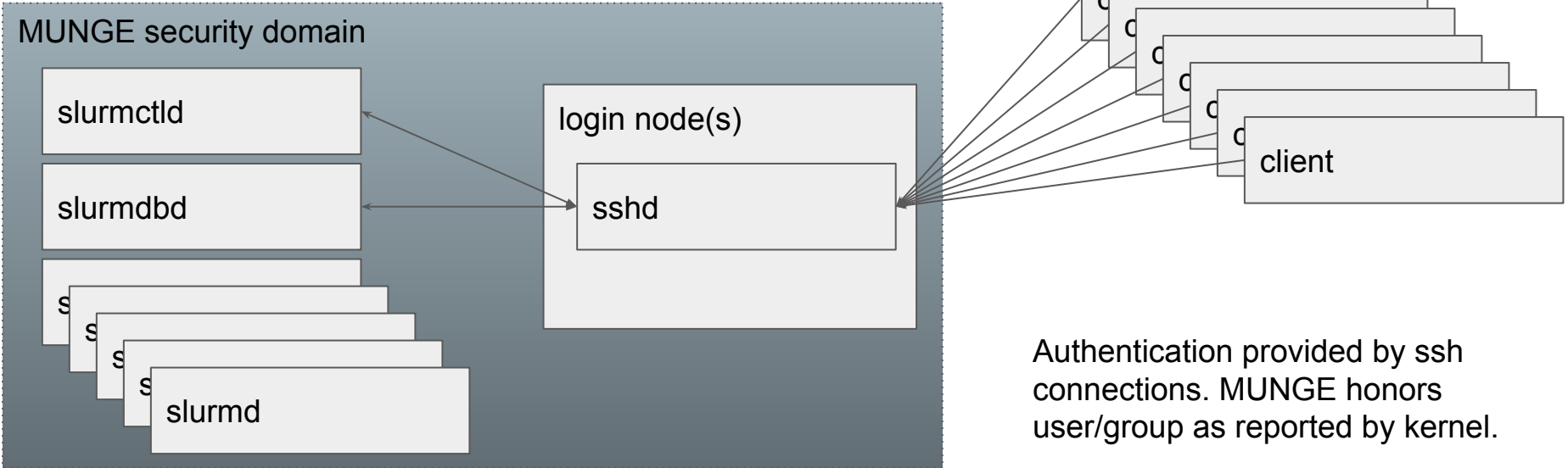
```
slurmrestd -d v0.0.39,v0.0.40 -s v0.0.39,dbv0.0.39,slurmdbd,slurmctld
```

# Command Line - OpenAPI specification

- Requesting OpenAPI schema for output (23.11+, v0.0.40+)
  - `sinfo -json=v0.0.40+spec_only`
- Produces output similar to an OpenAPI schema to allow caller to know the format of the expected result.
  - *sinfo* has no equivalent request in *slurmrestd*.
- OpenAPI standard only applies to URL paths:
  - Only the schema for output is returned instead of a full OpenAPI specification

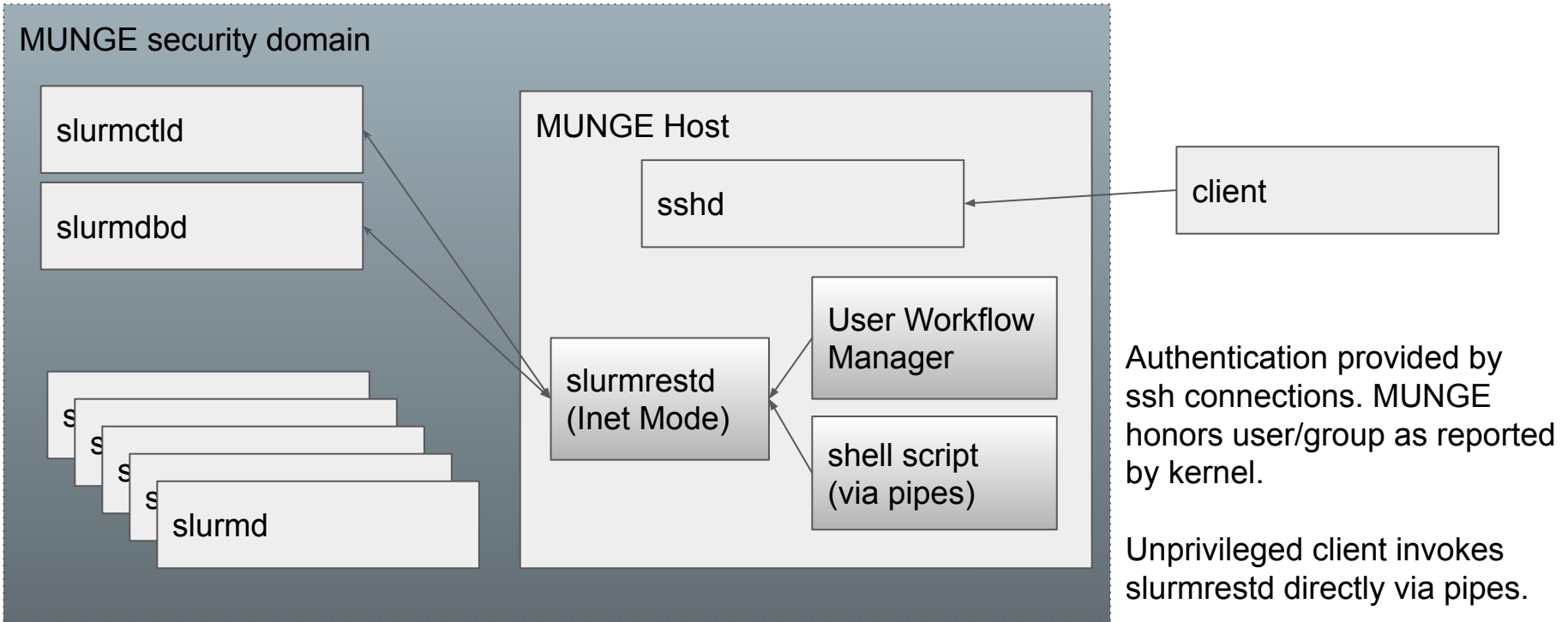
# Ways to deploy Slurm's REST API

# MUNGE and SSH based Slurm (aka Slurm without REST API)



Authentication provided by ssh connections. MUNGE honors user/group as reported by kernel.

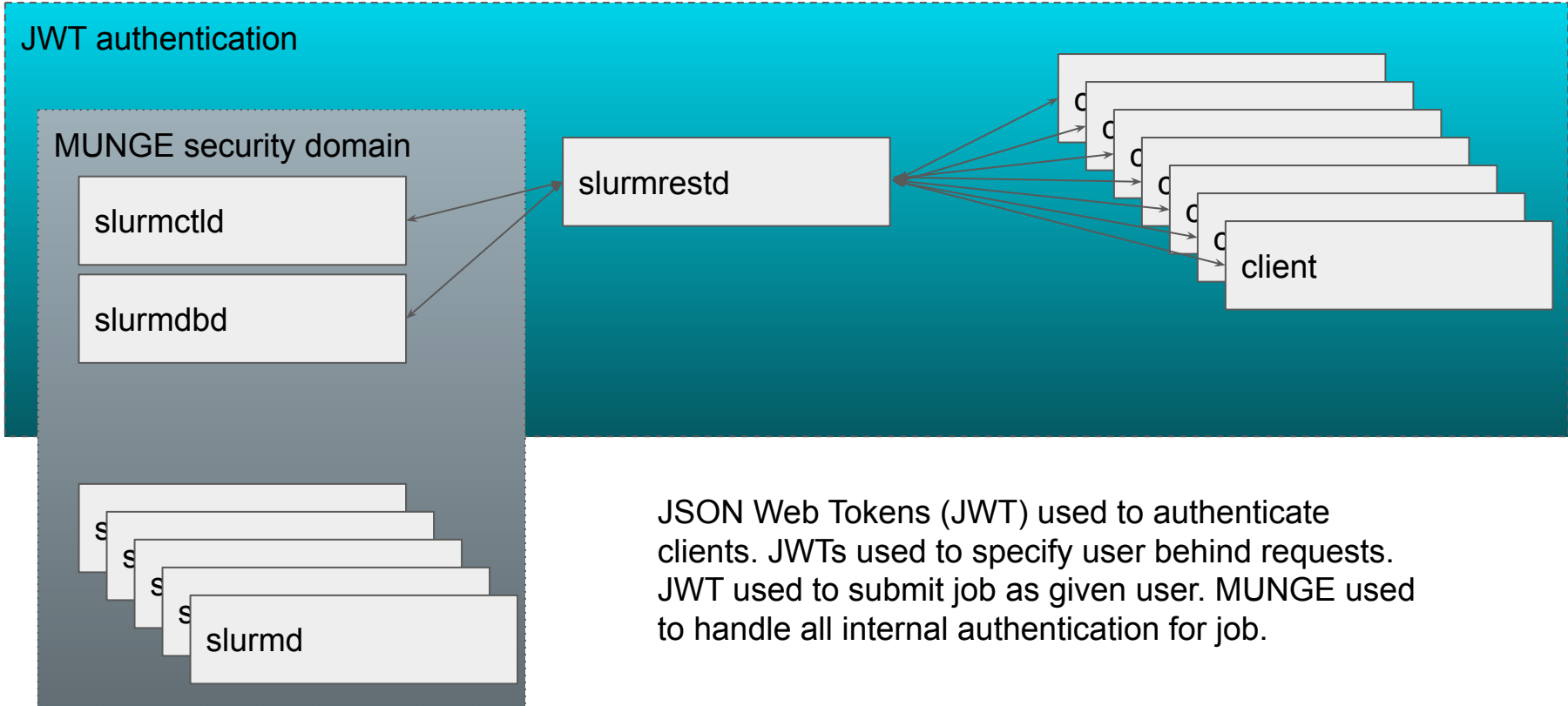
# Slurm REST API using only MUNGE and command line





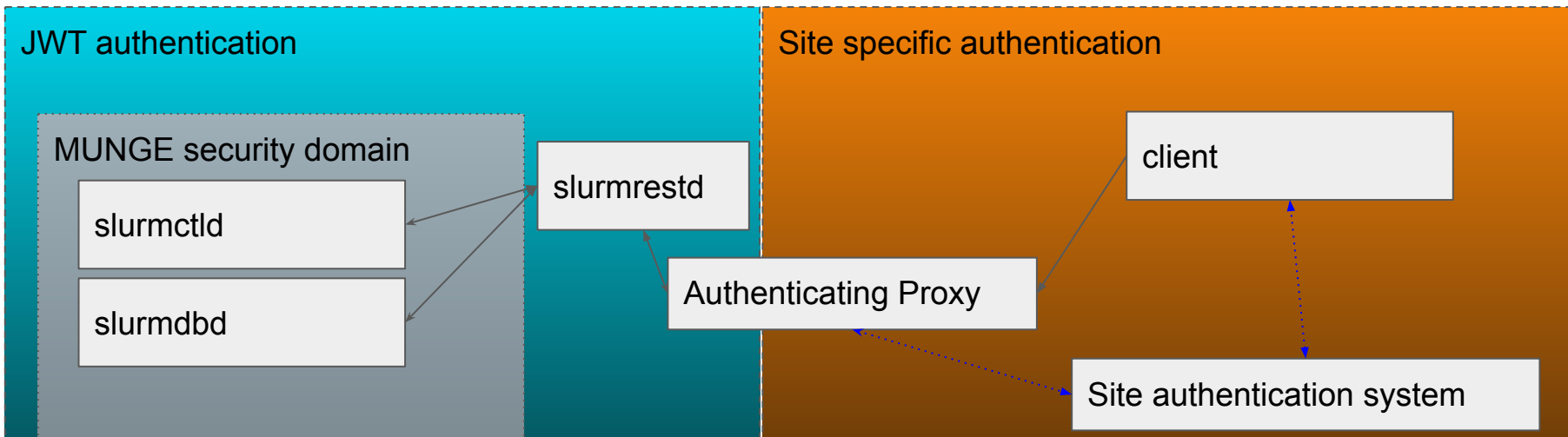
# Slurm REST API using JSON web tokens in an existing cluster

## JWT authentication



JSON Web Tokens (JWT) used to authenticate clients. JWTs used to specify user behind requests. JWT used to submit job as given user. MUNGE used to handle all internal authentication for job.

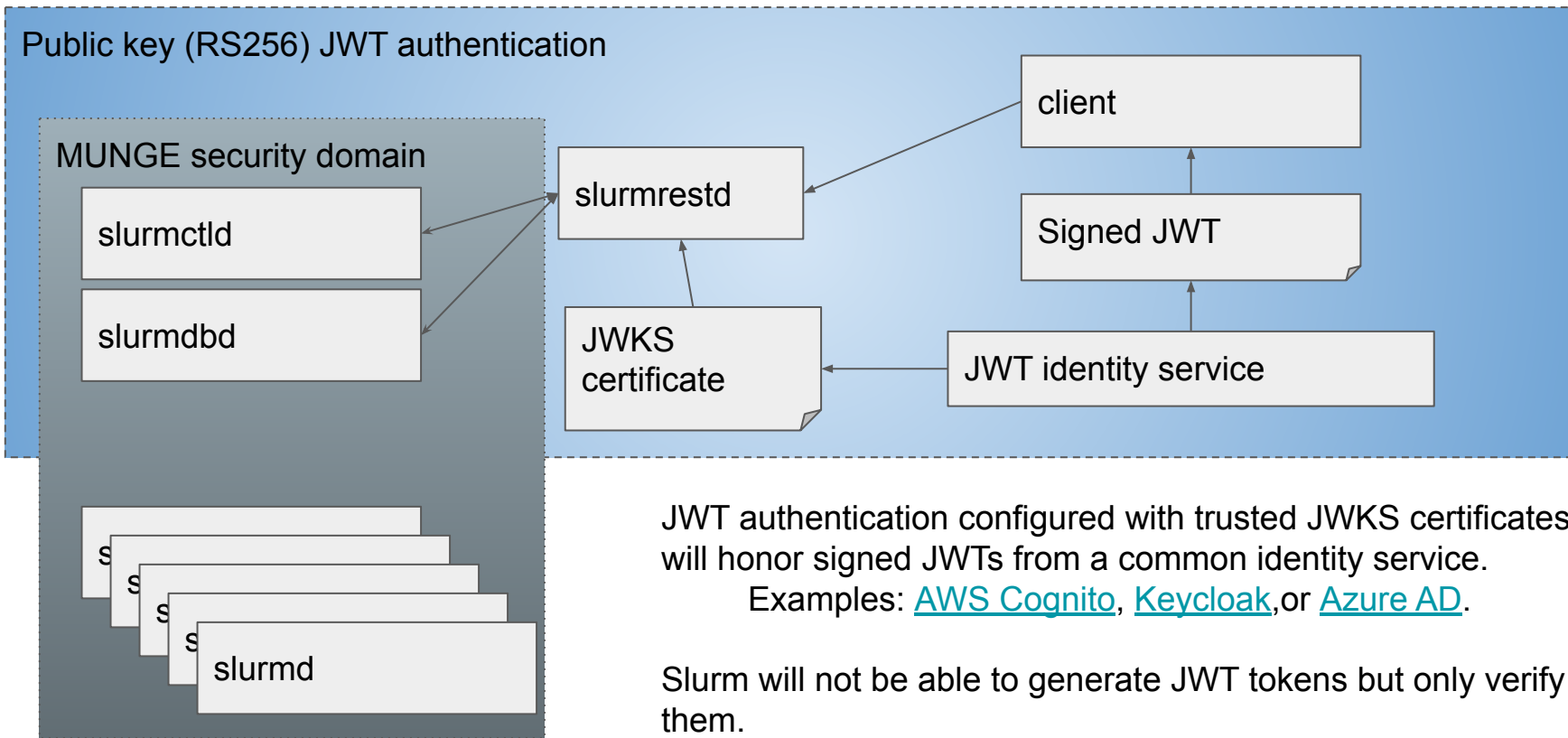
# Slurm REST API for the whole site or even the Internet



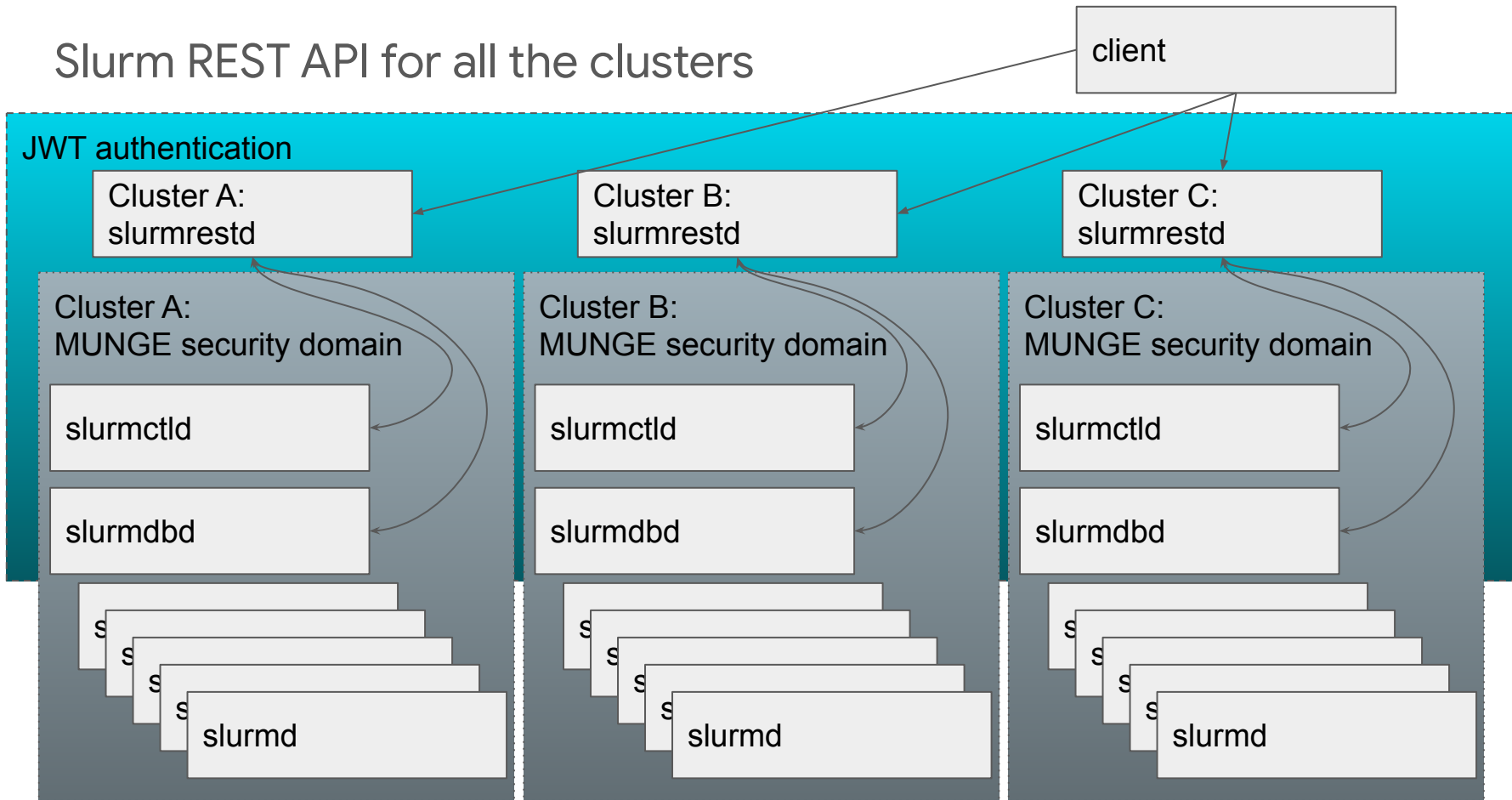
Authenticating Proxy can use token for SlurmUser to proxy requests for any user on the cluster following site designated security rules. This allows use of external authentication system such as Radius or Active Directory.

User/groups must have 1:1 mapping between security realms. All client connections must be TLS wrapped by proxy.

# Slurm REST API from the Cloud



# Slurm REST API for all the clusters



**Setup**

# Slurm's JWT Authentication

- Setup procedure: <https://slurm.schedmd.com/jwt.html>
- RFC7519 compliant implementation
- Supported algorithms:
  - HS256 (shared secret)
  - RS256 (public key)
- Users and groups on cluster must match token's user exactly
- Implementation is limited and can not be used to communicate with slurmd daemons.
- How to generate token in Slurm (HS256 algorithm only):

```
$ scontrol token  
SLURM_JWT=eyJhbGciOiJIUz.....
```

- Sites should consider generating JWTs outside of Slurm for automatic configuration of clients
  - Example: <https://slurm.schedmd.com/jwt.html#compatibility>
  - The SLURM\_JWT environment variable should not be set in user environments.

# slurmrestd - Slurm's REST API implementation

- How to compile
  - [Follow normal configuration procedure first](#)
  - slurmrestd will be automatically compiled if all prerequisites are present
    - checking whether to compile slurmrestd... yes
    - checking for slurmrestd default port... 6820
  - Possible to explicitly request slurmrestd
    - `./configure --enable-slurmrestd`
- slurmrestd is just another unprivileged binary callable by any user
  - Installed at `EPREFIX/sbin/slurmrestd`
    - Possible to change install path when calling configure:
      - `./configure --prefix=$NEW_INSTALL_PATH`
      - `./configure --sbindir=$NEW_INSTALL_PATH`
- slurmrestd must be able to communicate with slurmctld and slurmdbd via TCP connections.

# slurmrestd - Invoked directly

- Call slurmrestd directly from a shell script or from a in-cluster workflow manager
  - Avoids requiring any new authentication for the cluster
- Example (truncated):

```
$ echo -e 'GET /slurm/v0.0.40/jobs HTTP/1.1\r\n' | slurmrestd
HTTP/1.1 200 OK
Content-Length: 8758
Content-Type: application/json

{
  "jobs": [
    {
      "job_id": 192,
      "job_state": [
        "RUNNING"
      ],

```



# slurmrestd - Proxying

- slurmrestd is HTTP standards compliant
  - Any tool that can work with a web server should work with slurmrestd
- Sites are suggested to setup a proxy between clients and slurmrestd
  - Use [Nginx](#), [Apache](#), or proxy du-jour
- Add caching
  - Configure caching in proxy as desired
  - Reduce strain on slurmctld and slurmdbd for repeated requests
- Always wrap communications with TLS
  - Never directly expose slurmrestd (or any of Slurm) to the Internet.
- Use authentication proxy functionality in the proxy to use existing site authentication instead of MUNGE or Slurm's JWT implementation.
  - Example: <https://github.com/naterini/docker-scale-out/tree/master/proxy>
  - Avoid users having to grab a new JWT by calling `scontrol token`

## slurmrestd - Running as a listening daemon

- Start daemon listening on IPv4 localhost TCP port 8080, IPv6 localhost TCP port 8080, IPv6 and IPv4 on all interfaces TCP port 8181, streaming Unix socket at /path/to/unix.socket with Slurm-23.11 content plugins only using JWT authentication for a Slurm-23.11 install.

```
$ env SLURM_JWT=daemon slurmrestd unix:/path/to/unix.socket 127.0.0.1:8080  
ip6-localhost:8080 :8181 -a jwt -s slurmctld,slurmdbd -d v0.0.40
```

- Start daemon listening on IPv4 localhost TCP port 8080, IPv6 localhost TCP port 8080, IPv6 and IPv4 on all interfaces TCP port 8181, streaming Unix socket at /path/to/unix.socket with Slurm-23.02 content plugins only using JWT authentication for a Slurm-23.02 install.

```
$ env SLURM_JWT=daemon slurmrestd unix:/path/to/unix.socket 127.0.0.1:8080  
ip6-localhost:8080 :8181 -a jwt -s v0.0.39,dbv0.0.39
```

# slurmrestd - Running as a listening daemon via systemd

- Sites are suggested to use slurmrestd.service for systemd
  - Compiled and placed in build directory as etc/slurmrestd.service
  - Drop-in unit should be used to change values instead of modifying template.
    - Make sure to always update the systemd unit during an upgrade
- Example setup procedure:

```
cp $BUILD_PATH/etc/slurmrestd.service /usr/lib/systemd/system/slurmrestd.service
mkdir -p /usr/lib/systemd/system/slurmrestd.service.d
cp local.conf /usr/lib/systemd/system/slurmrestd.service.d
systemctl daemon-reload
systemctl start slurmrestd
```

- Drop in example (local.conf): (restricts slurmrestd to only startup on host “rest”)

```
[Service]
ExecCondition=bash -c 'test $(hostname -s) = "rest"'
```

**Optimization**

## slurmrestd: Fast mode Parser (v0.0.40+,23.11+)

- Attempt to process queries as fast as possible by sacrificing warning checks and readability for humans.
- **This flag should only be used for production systems with production clients that have already been fully tested** without the fast flag active.
- Other enhancements may be added in future releases to improve processing speed.
- Example:

```
slurmrestd -d v0.0.40+fast -s slurmdbd,slurmctld $LISTEN_PORTS
```

# slurmrestd: Compact JSON/YAML (v0.0.40+,23.11+)

- Generated JSON/YAML outputs are by default done with extra characters to improve readability.
  - For some sites with large number of requests to slurmrestd, skipping unnecessary whitespace characters can have considerable performance benefit in processing time and reduced network usage.

- Environment variables to activate compact mode:

- SLURMRESTD\_YAML=compact
- SLURMRESTD\_JSON=compact

- Example:

```
env SLURMRESTD_JSON=compact SLURMRESTD_YAML=compact slurmrestd -d  
v0.0.40 -s slurmdbd,slurmctld $LISTEN_PORTS
```

**Client compatibility**

# slurmrestd - Plugins lifetime matrix

- All paths in slurmrestd requests include the relevant plugin version:
  - Example: `http://$HOST/slurmdb/v0.0.40/jobs`

Added in Slurm release	Content Plugins (-s)	Data_parser Plugin (-d) [23.11+]	Removal in Slurm release
21.08	v0.0.37,dbv0.0.37		23.11
22.05	v0.0.38,dbv0.0.38		24.08
23.02	v0.0.39,dbv0.0.39	v0.0.39	25.05
23.11	slurmctld,slurmdbd	v0.0.40	26.02 (v0.0.40 only)
24.08		v0.0.41	26.11

- Unversioned slurmctld and slurmdbd content plugins added in Slurm-23.11 have no planned removal date.



# Compatibility Testing

- slurmrestd is currently tested using:
  - [openapi-generator-cli](#) generated python client
    - Tests uses static driver code against generated python clients
    - New test units are required for each data\_parser version and the major version of *openapi-generator-cli*.
    - Arguably the most popular client generator for OpenAPI due to heritage from Swagger.
  - *curl*
    - Direct queries of slurmrestd using hand crafted requests
- Breaking changes of existing clients of the same version is considered a bug.
- General goal of reducing changes required for porting to newer versions.
  - Depending on the relevant change(s), requests in prior accepted formats will still be accepted but with warnings sent to client.

# OpenAPI Standard Compliance

- *openapi-generator-cli* created clients can not handle or refuse unexpected data types
  - In most cases, the client will assert but others just result in a segfault.
- OpenAPI standard includes *oneOf()* and *anyOf()* operators to allow for polymorphism
  - Allows return of *null* when a field isn't set.
  - Slurm makes heavy use of polymorphism internally.
    - *slurmrestd* designed to handle polymorphic formats
- *openapi-generator-cli* client is not monolithic
  - Uses a plugin based approach to create generators for many languages
  - Clients for each language have varying level of OpenAPI standard support
- *openapi-generator-cli* generated clients will crash when handed (some) schemas using *oneOf()*

# To Infinity and... Assert!

- Slurm makes heavy use of Infinity or Unlimited, usually as a way to disable a limit.
- ECMA-404 JSON does not support a value of *infinity* (or  $\pm$ *infinity* or  $\pm$ NaN)
  - Most JSON parsers actually support *infinity*
    - Some silently convert to max of the internal type:

```
$ echo infinity | jq  
1.7976931348623157e+308
```

- OpenAPI standard does not support (or explicitly ban) use of *infinity*
    - *openapi-generator-cli* python client will assert upon receiving *infinity*
- slurmrestd supports *infinity* (and NaN which is not used)
  - slurmrestd can automatically convert “*inf*”, “+*inf*”, “-*inf*”, “*infinity*”, “+*infinity*”, “-*infinity*” string values to OpenAPI number format for inputs.
    - Warnings will still be issued about non-compliance with OpenAPI specification for such format conversions for any given field.
    - slurmrestd should **not** output *infinity* or NaN to avoid breaking clients.

# slurmrestd and the non-compliant clients?

- Several sites opened tickets against slurmrestd for broken clients
  - slurmrestd was written against the OpenAPI standard
    - In theory, the non-compliance to the OpenAPI standard of any one client should be fixed by clients.
    - This effectively set the bar for entry too high for most sites who were not writing their own clients.
- slurmrestd's workarounds - (tagged with "NO\_VAL" in parser/schema naming)
  - All use of *oneOf()* and *anyOf()* removed (20.11+)
  - *Infinity*, *null*, and *NaN* will not be dumped as result of a request (20.11+)
  - *Infinity* and *null* must be presented as booleans fields in representative object (23.02+)
    - Example: `{ "set": true, "infinite": true, "number": 0 }`
  - All fields present and populated in dumped responses (23.02+)

# OpenAPI Specification

# OpenAPI Specification

- slurmrestd generates the OpenAPI Schema based on runtime arguments.
  - Sites should always specify the plugins via ``-s``, ``-a`` and ``-d`` (23.11+) they plan to use explicitly via arguments instead of the default of loading all plugins found for production servers.
- Previously, we tried to have a single static specification as a static file (openapi.json).
  - Maintaining the OpenAPI Specification by hand in git ended up causing more problems than it solved as git kept mangling the content and formatting.
  - Schemas are now generated by slurmrestd and the openapi.json is a basic template in the source code (23.02+)
    - Same code that generates the output also generates the schema to keep everything as coherent as possible.
    - The generated OpenAPI schemas at `"http://$HOST/openapi/v3"` should always be used instead of the openapi.json in the source code.

# Improving the OpenAPI Specification

- String Schemas with Enum (23.02+)
  - OpenAPI standard provides the Enum array to allow strings with well defined values to enumerated out.
  - slurmrestd internally tracks most of these well defined strings as flags.
    - Many fields have been converted to flags to make it easier for users to know possible values (23.11)
- Path parameters are now generated (23.11+)
  - All possible parameters should now be in generated OpenAPI specification including enum strings.
- Boolean query parameters in the URI without a value will be considered to be *true*.
  - Example: [http://\\$HOST/slurmdb/v0.0.40/associations?with\\_usage](http://$HOST/slurmdb/v0.0.40/associations?with_usage)
  - *openapi-generator-cli* clients will need to pass “true” or “false” in the query objects as the internal schema checker will reject a value of *None*.

# OpenAPI Specification Versioning

- Format and layout of schemas are consistent between all Slurm releases where the versioned plugin is present in the release:
  - A query to v0.0.40 endpoint in 23.11 should work the same as a query to v0.0.40 endpoint in Slurm 24.08.
  - Schemas changes during patchset releases are only done to correct breaking issues, such as ones causing most *openapi-generator-cli* clients to crash.
  - Schemas between different data\_parser versions are not guaranteed to be compatible and in some cases may be entirely different. Make sure to test clients when porting between versions.
  - OpenAPI Specifications are versioned the same as the data\_parser versions and have same version stability.



**Questions?**

**SCHEDMD**

The Slurm Company