
ParslFest 2023
Multi-User Endpoints
Kevin Hunter Kesling – kevin@globus.org
October 20, 2023
THE UNIVERSITY OF CHICAGO
Argonne NATIONAL LABORATORY
globus
1

What is a Multi-User Compute Endpoint?

- Reminder of “normal endpoints”
  - Configure
  - Start (which we interpret at the web-services as “register”)
  - Use the printed uuid identifier in your SDK scripts
  - “Oh wait, I need to change that configuration because [reason]”
  - Log in, change the configuration, restart
  - “Oh shoot, the cluster has been restarted”
  - Log in, start the endpoint afresh
  - “Hmm …I need more than one configuration for my various workloads”
  - Log in, and manage them ad-hoc
- … And so forth; for many folks, it’s not uncommon to have multiple configurations, that each need to be managed

What is a Multi-User Compute Endpoint?

- Aside … hereafter:
  - MEP → Multi-User Endpoint
  - UEP → User Endpoint (“normal endpoint”)

- Reminder of “normal endpoints”
  - Configure
  - Start (which we interpret at the web-services as “register”)
  - Use the printed uuid identifier in your SDK scripts
  - “Oh wait, I need to change that configuration because [reason]”
  - Log in, change the configuration, restart
  - “Oh shoot, the cluster has been restarted”
  - Log in, start the endpoint afresh
  - “Hmm …I need more than one configuration for my various workloads”
  - Log in, and manage them ad-hoc
- … And so forth; for many folks, it’s not uncommon to have multiple configurations, that each need to be managed

What is a Multi-User Compute Endpoint?

- Aside … hereafter:
  - MEP → Multi-User Endpoint
  - UEP → User Endpoint (“normal endpoint”)
- In contrast to a “normal” compute endpoint, an MEP **does not run tasks**.

- Reminder of “normal endpoints”
  - Configure
  - Start (which we interpret at the web-services as “register”)
  - Use the printed uuid identifier in your SDK scripts
  - “Oh wait, I need to change that configuration because [reason]”
  - Log in, change the configuration, restart
  - “Oh shoot, the cluster has been restarted”
  - Log in, start the endpoint afresh
  - “Hmm …I need more than one configuration for my various workloads”
  - Log in, and manage them ad-hoc
- … And so forth; for many folks, it’s not uncommon to have multiple configurations, that each need to be managed

What is a Multi-User Compute Endpoint?

- Aside … hereafter:
  - MEP → Multi-User Endpoint
  - UEP → User Endpoint (“normal endpoint”)
- In contrast to a “normal” compute endpoint, an MEP **does not run tasks**.
- Instead, an MEP
  - starts UEPs
  - (Slightly more precisely, fork, drop privileges, exec)
  - Manages their lifecycle (okay, `os.fork()` and `os.waitpid()`)

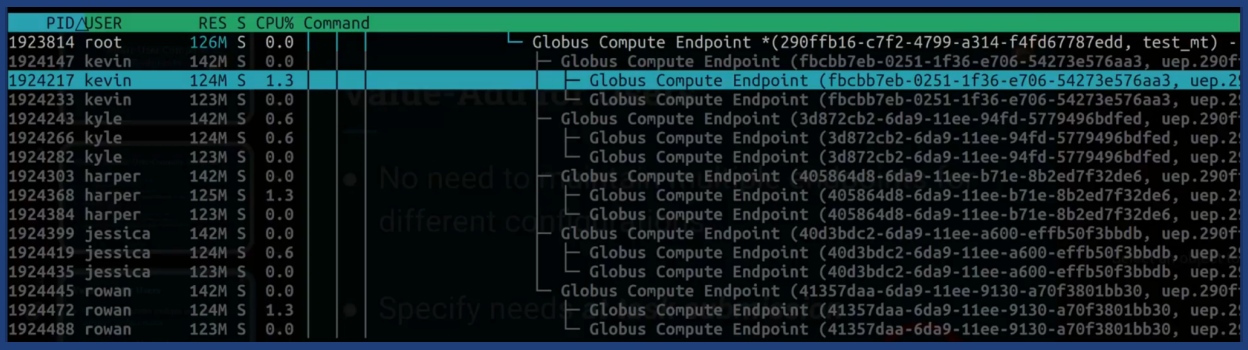
- Reminder of “normal endpoints”
  - Configure
  - Start (which we interpret at the web-services as “register”)
  - Use the printed uuid identifier in your SDK scripts
  - “Oh wait, I need to change that configuration because [reason]”
  - Log in, change the configuration, restart
  - “Oh shoot, the cluster has been restarted”
  - Log in, start the endpoint afresh
  - “Hmm …I need more than one configuration for my various workloads”
  - Log in, and manage them ad-hoc
- … And so forth; for many folks, it’s not uncommon to have multiple configurations, that each need to be managed

What is a Multi-User Compute Endpoint?

- Aside … hereafter:
  - MEP → Multi-User Endpoint
  - UEP → User Endpoint (“normal endpoint”)
- In contrast to a “normal” compute endpoint, an MEP **does not run tasks**.
- Instead, an MEP
  - starts UEPs
  - (Slightly more precisely, fork, drop privileges, exec)
  - Manages their lifecycle (okay, `os.fork()` and `os.waitpid()`)
- Receives start UEP commands from the web-service

- Reminder of “normal endpoints”
  - Configure
  - Start (which we interpret at the web-services as “register”)
  - Use the printed uuid identifier in your SDK scripts
  - “Oh wait, I need to change that configuration because [reason]”
  - Log in, change the configuration, restart
  - “Oh shoot, the cluster has been restarted”
  - Log in, start the endpoint afresh
  - “Hmm …I need more than one configuration for my various workloads”
  - Log in, and manage them ad-hoc
- … And so forth; for many folks, it’s not uncommon to have multiple configurations, that each need to be managed

htop screen recording



PDF NOTE: Original presentation had a live screen recording, showing the values updating in real time as “presentation-proof” that the software exists (if not yet released). See speaker notes.

Video of original presentation linked via the ParslFest 2023 list of presentations.
(`https://parsl-project.org/parslfest/parslfest2023.html`)

- Showing it in action on my laptop; a screen recording of htop so as “to prove” that it exists, “really,” even though still in development. (“Nearly there!!!”)
- Key point is the main process has children – forked, and not double-forked – and the children are not owned by root but by actual users on the system
- Tree is *enforced* – respect the admin, always.

How do we *do* it?

- Admin writes the main configuration
- Configuration will be run through the Jinja template engine
- Admin may export variables via the usual Jinja syntax (`{{ variable_name|filter1|filter2|... }}`)
- `User need only specify the variables at submission time.`

Admin Writes/Controls

```yaml
engine:
 type: GlobusComputeEngine

 provider:
   type: SlurmProvider
   partition: cpu
   account: {{ ACCOUNT_ID }}

   launcher:
     type: SrunLauncher

   walltime: {{ walltime|default("00:30:00") }}
```

user_config_template.yaml

- Admin writes the main configuration
- Configuration will be run through the Jinja template engine
- Admin may export variables via the usual Jinja syntax (`{{ variable_name|filter1|filter2|... }}`)
- `User need only specify the variables *at submission time*.`

Admin Writes/Controls

```
engine:
 type: GlobusComputeEngine

 provider:
   type: SlurmProvider
   partition: cpu
   account: {{ ACCOUNT_ID }}

   launcher:
     type: SrunLauncher

   walltime: {{ walltime|default("00:30:00") }}
```

user_config_template.yaml

User Script

```
import globus_compute_sdk as GC

uep_conf = {
    "ACCOUNT_ID": "314159265",
    "walltime": "00:02:00"
}

with GC.Executor(
  endpoint_id=mep_id,
  user_endpoint_config=uep_conf
) as gce:
    fut = gce.submit(some_func)
    res = fut.result()
```


- Admin writes the main configuration
- Configuration will be run through the Jinja template engine
- Admin may export variables via the usual Jinja syntax (`{{ variable_name|filter1|filter2|... }}`)
- `User need only specify the variables *at submission time*.`

Admin Writes/Controls

```
engine:
 type: GlobusComputeEngine

 provider:
   type: SlurmProvider
   partition: cpu
   account: {{ ACCOUNT_ID }}

   launcher:
     type: SrunLauncher

   walltime: {{ walltime|default("00:30:00") }}
```

user_config_template.yaml

User Script

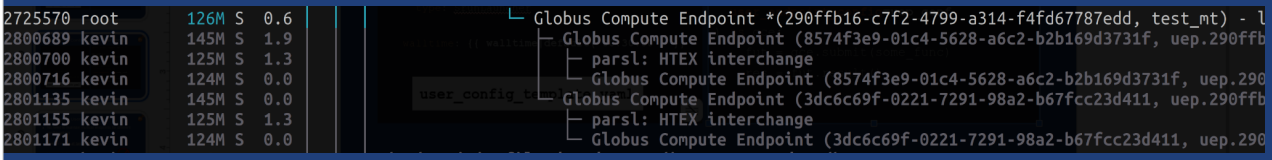
```
import globus_compute_sdk as GC

uep_conf = {
    "ACCOUNT_ID": "543126688"
}


with GC.Executor(
  endpoint_id=mep_id,
  user_endpoint_config=uep_conf
) as gce:
    fut = gce.submit(some_func)
    res = fut.result()
```


- The user still needs to be aware of the configuration pieces of interest
- "The abstraction is still leaky!"
  - But less leaky.
  - The user needs to know about less (SlurmProvider can be ignored by user; only `account_id` matters)
- Key point: configuration of interest is *closer to the SDK codes that user them*
  - Not attached to an opaque uuid identifier
- Side note: observe that the admin *can* specify defaults, meaning the user need not specify ALL variables. Just the required one.
  - N.B. if the user does not supply account_id, then the UEP would still start up (valid YAML to have an empty account_id) but the submission would fail

Two different configurations; same user!



Value-Add for Users

Value-Add for Users

- No need to maintain multiple endpoints for different configurations

Value-Add for Users

- No need to maintain multiple endpoints for different configurations
- Specify needs **at task submission**

Value-Add for Users

- No need to maintain multiple endpoints for different configurations
- Specify needs **at task submission**
- No need to log in to the terminal

Value-Add for Site Administrators

Value-Add for Site Administrators

- Templatable User Endpoint Configurations (Jinja)
  - e.g., pre-choose SlurmProvider, PBSProvider; enforce limits

Value-Add for Site Administrators

- Templatable User Endpoint Configurations (Jinja)
  - e.g., pre-choose SlurmProvider, PBSProvider; enforce limits
- No orphaned user compute endpoints
  - Enforced process tree
  - Idle-endpoints are shutdown (per template configuration)

Value-Add for Site Administrators

- Templatable User Endpoint Configurations (Jinja)
  - e.g., pre-choose SlurmProvider, PBSProvider; enforce limits
- No orphaned user compute endpoints
  - Enforced process tree
  - Idle-endpoints are shutdown (per template configuration)
- Standard Globus Identity Mapping

Value-Add for Site Administrators

- Templatable User Endpoint Configurations (Jinja)
  - e.g., pre-choose SlurmProvider, PBSProvider; enforce limits
- No orphaned user compute endpoints
  - Enforced process tree
  - Idle-endpoints are shutdown (per template configuration)
- Standard Globus Identity Mapping
- **Lower barrier for users**

Current status

- We’re buttoning up a few details
- Have not yet written any documentation
- Looking for brave volunteers to give it go

Thank You!

- **Questions?**
- **Comments?**
- **Synergistic thoughts?**