New User Experiences with K3s and MetalLB on Managed Nodes

Alan Mutschelknaus
Hewlett Packard Enterprise
Bloomington, MN
alanm@hpe.com

***Abstract*—Traditional managed nodes on HPE Cray EX systems dedicated to user compilations and job launch, have not provided support for user interaction beyond the standard SSH shell environment. This model works for many use cases, but it does not provide the flexibility of container orchestration. While User Access Instances (UAIs) are available as containerized login environments, they currently run in the Cray System Management (CSM) Kubernetes (K8s) cluster and would be better suited to run alongside other user activities.**

This paper will show how a cluster of managed nodes running K3s and MetalLB can be used to host a suite of new experiences for users. K3s is a lightweight, API compatible distribution of Kubernetes. Using rootless Podman for container execution and HAProxy to route SSH connections, users can have a fully customizable UAI experience. The HAProxy load balancer for SSH can run as a DaemonSet across the managed nodes for resiliency. A type of Broker UAI on the managed nodes can then forward users into a customizable, rootless Podman container. To illustrate how K3s and MetalLB opens the door to other user interactions, the paper will also show how JupyterHub can be deployed to the K3s cluster using Helm.

***Index Terms*—Kubernetes, Podman, HAProxy, MetalLB**

I. Introduction

There has been a strong response from the HPC community demonstrating an eagerness for new methods of interacting with HPC systems. In particular, the paper ”UAIs Come of Age: Hosting Multiple Custom Interactive Login Experiences Without Dedicated Hardware” won the CUG 2022 Best Paper award [1]. At the same time, a common refrain was that while UAIs are desirable as an alternative to traditional login environments, they may not be used while they coexist on the same nodes as management services like CSM. This points to the need to evolve the concept of UAIs to exist on managed nodes where other user activities take place. This transformation is an opportunity to rethink the purpose of containerized, temporary user environments and lay the foundation for supporting new modes of user interactions in HPC.

Features are also desired of UAIs that are not yet possible or would conflict with the architecture of UAIs on CSM management nodes. The first of which is support for GPUs. The most immediate issue is that GPUs are typically supported on managed nodes and not management nodes. Support for swap is another feature that would impose requirements on management nodes that may be unacceptable or out of scope. This includes support for the latest versions of K8s and performance or scheduling implications of user containers on management nodes utilizing swap.

Another advantage, is that while managed nodes could join CSM management nodes, there are significant benefits in flexibility, velocity, and security in operating an independent K3s cluster. This decision will benefit both managed and management nodes. Specifically, UAIs on management nodes imposed some of the current requirements on management nodes like Data Virtualization Service (DVS) projection of the HPE Cray Programming Environment (CPE) and various networking implementations of Macvlan or IPvlan [2] and K8s *network-attachment-definitions*.

Two approaches to solving the requirement of UAIs on non-management nodes are clear. A managed node could join the CSM K8s cluster as a new worker node meant to host UAIs, or managed nodes could instantiate their own independent K8s compatible cluster and break away from requiring the CSM K8s cluster. For reasons outlined in this paper, the latter was chosen.

II. Design

A. Summary

To support new forms of interactive HPC use on HPE Cray EX, we chose to use K3s from SuSE Rancher, a Cloud native Compute Foundation (CNCF) sandbox project. K3s introduces an alternative way to support container orchestration with less of the management complexity of a full K8s deployment. At the same time, K3s supports plugins for enhanced capabilities where required. This foundation can be used on HPE Cray EX managed nodes, or other bare metal servers, with the aim of enabling new user interactions to compile and run jobs, visualize data, share environments, and more.

Existing solutions for containerized user environments like User Access Instances (UAIs), relied on a K8s cluster that also ran management services. A K3s cluster across a separate set of managed nodes will strengthen the security model around a new concept for UAIs by keeping user processes isolated from management processes. Additionally, the K3s cluster can remain operational during upgrades or maintenance cycles of the management nodes, while still supporting upgrades itself with standard *cordon* and *uncordon* Kubernetes operations.

As a fully encapsulated binary, K3s is optimized for lightness and speed. Installation and deployment on one or more managed nodes can take minutes and requires a minimal amount of configuration for basic operation. The K3s cluster of managed nodes on HPE Cray EX systems can be configured with standard image management and configurations tools.

MetalLB [3] is one type of extension that can be added to the K3s cluster. By providing a range of IP addresses, MetalLB in the K3s cluster will allow site routable *LoadBalancer* IP addresses to be assigned to services running across the cluster. On an HPE Cray EX system, this can be IP addresses on the Customer High-Speed Network (CHN) or the Customer Access Network (CAN).

Two additional components can be configured to replicate the experience of UAIs: Podman [4] and HAProxy [5].

B. Use of K3S

K3s will serve as the orchestrator of services necessary to replicate the capabilities of UAIs on managed node hardware. This includes HAProxy, MetalLB, and eventually DNS services like ExternalDNS and PowerDNS [6]. Notably, this design does not orchestrate instances of SSHD and Podman containers through K3s (see Figure 1). As a comparison, K3s and the initial set of services are operating in a similar way to Broker UAIs in CSM to handle the SSH ingress and redirection of users into their interactive environment.

C. Use of Podman

Directly porting the current concept of UAIs to managed nodes in K3s would imply a wholesale migration of many of the components needed to manage the configuration and lifecycle of UAIs, namely the User Access Service (UAS) and its dependencies:

- UAS Manager
- API Gateway for handling requests to UAS
- AuthN and AuthZ service (Keycloak) for the API Gateway
- etcd storage for configuration
- Image registry for UAI container images
- RBAC controls for UAS use of K8s API
- Macvlan and IPvlan configuration for net-attachment-definitions to support WLM job launch
- Broker UAI Image
- Switchboard
- Vault for SSH key management

Rather than begin the process of re-implementing these dependencies to operate independently of CSM across managed nodes, Podman, a lightweight alternative, will be introduced. Podman provides many of the same benefits that UAIs offered: process isolation, container image flexibility, and Open Container Initiative (OCI) compatibility [7] while also introducing an additional benefit: rootless containers.

Traditional UAIs required some level of privilege in CSM for access to host volume mounts, networking, and startup activities. Despite any attempts to mitigate these requirements, UAIs fundamentally increase the attack surface on management nodes when in use. The same attack surface would be true of porting the current UAI implementation on managed nodes (not counting the level of isolation from management services that could be achieved). Podman offers an attractive solution for an interactive environment in which to place users, they can be rootless containers that do not rely on privilege escalation.

One disadvantage of relying on Podman for an interactive user environment is the lack of scheduling managed by K3s implicit with this design. There are options for reintroducing some level of scheduling and orchestration of Podman containers that will be discussed later in the paper. The high-level view of the interaction between the services running in K3s are as follows:

1) A user uses SSH to initiate a connection the HAProxy load balancer.
2) HAProxy, using the configured load balancing algorithms, will forward the SSH connection to an instance of SSHD.
3) SSHD, running on a managed node via systemd, will initiate a rootless Podman container as the user using the ForceCommand configuration.
4) The user is placed in a Podman container for an interactive session, or their *SSH_ORIGINAL_COMMAND* is run in the container.
5) When the user disconnects, the Podman process exits, and the container is removed.

There are alternate configurations of Podman that would allow for different workflows, for example, the main PID of the container could be long running, to facilitate easier reentry to the container on subsequent logins. This may even prove necessary for container reuse, where *podman run* starts the long running container if none exist, and *podman exec* is used to place the user in the container.

D. Configuration with Ansible

The design should support, as generically as possible, deployment and configuration of the necessary components. This implies that dependencies on CSM should be avoided so the solution could be adaptable to other cluster managers. Ansible is compatible with the Configuration Framework Service (CFS) of CSM, and it is also portable enough,

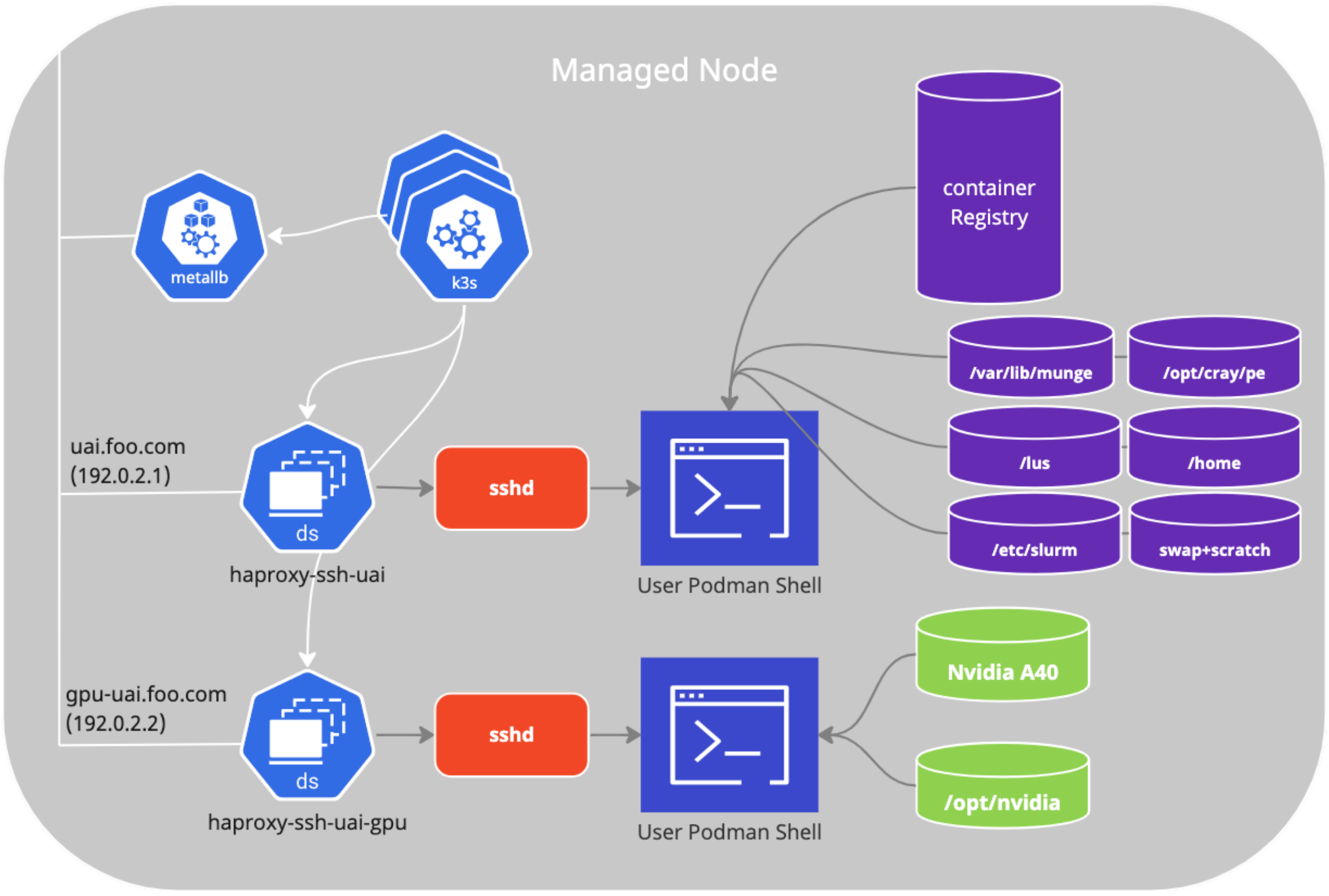

Fig. 1. Managed Node K3s Diagram

without introducing an explicit dependency on CSM, as the framework to deploy the solution to managed nodes. The new deployment and configuration Ansible could be readily adaptable to other cluster management tools.

A benefit of CFS is that configuration changes to actively running CFS configurations will automatically reconfigure managed nodes. For example, if an administrator wants to add another instance of HAProxy, or reconfigure Podman, pushing updates to CFS will automatically update the managed nodes without requiring downtime. This satisfies a number of use cases, and gives administrators a consistent experience with existing configuration tooling.

In order to integrate with the existing configuration framework of CFS, Ansible plays perform the operational tasks of installation and configuration. The ansible playbook steps could be summarized as:

1) Download assets for K3s, HAProxy, and MetalLB to the managed node image (*uan_k3s_stage*, *uan_helm*)
2) K3s initialization (*uan_k3s_install*)
3) Helm initialization
   a) Configure paths
   b) Download charts
4) Podman configuration
   a) Initialize subuid and subgid
5) MetalLB Configuration
   a) Deploy Helm Chart
   b) Deploy Custom Resource Definition (CRD) for *IPAddressPool*
6) HAProxy Configuration
   a) Process Helm values.yml for configmaps
   b) Install Helm chart
7) SSHD
   a) Create SSHD config files and systemd unit files
   b) Enable SSHD

E. Installation of K3s

The offline installation of K3s is performed through new Ansible roles *uan_k3s_stage* and *uan_k3s_install*. K3s artifacts are installed as part of CFS Image Customization, while the actual startup takes place after the node boots during CFS Node Personalization. The *uan_k3s_stage* may optionally be performed during CFS Node Personalization to facilitate rapid prototyping and experimentation. As the *uan_k3s_stage* is idempotent, there is no functional impact other than scaling implications of the managed nodes pulling the necessary artifacts.

F. Configuration of MetalLB

To provide routable IP addresses for the HAProxy load balancers, two Ansible parameters must be configured. The IP address range must be site-routable and previously unallocated. The new Ansible role *uan_metallb* will consume these parameters:

```
metallb_ipaddresspool_range_start: "x.x.x.x"
metallb_ipaddresspool_range_end: "y.y.y.y"
```

Listing 1. MetalLB IPAddressPool Configuration

This will populate the MetalLB Custom Resource Definition *IPAddressPool*:

```
uan01:~ # kubectl get -n metallb-system
    IPAddressPool/ipaddresspool -o json | jq .spec.
    addresses
[
  "x.x.x.x-y.y.y.y"
]
```

Listing 2. Checking MetalLB IPAddressPool Resource

G. UAI Classes

A more recent evolution of UAIs running under K8s in CSM added the construct of a ”UAI Class”. A UAI class is a collection of configurations that specifies how a UAI should be constructed and accessed. While not all the available options will apply to the new Podman approach, many of the concepts could be carried over. In the current iteration of this design, configuration of SSHD, HAProxy, and Podman will be handled separately and automated across managed node hardware via CFS. All of this configuration information could ultimately be consolidated into a new form of UAI Classes for Podman. This would make it easier for administrators to coordinate the configuration of HAProxy, SSHD, and Podman in one location.

H. Configuration of HAProxy

A list of HAProxy helm charts may be defined to represent a particular SSH ingress into the K3s cluster:

```
uan_haproxy:
  - name: "haproxy-gpu"
    namespace: "haproxy-gpu
    chart: "{{ haproxy_chart }}"
    chart_path: "{{ helm_install_path }}/charts/{{
    haproxy_chart }}.tgz"
    args: "--set service.type=LoadBalancer"

  - name: "haproxy-uai"
    namespace: "haproxy-uai"
    chart: "{{ haproxy_chart }}"
    chart_path: "{{ helm_install_path }}/charts/{{
    haproxy_chart }}.tgz"
    args: "--set service.type=LoadBalancer"
    config: |
      global
        log stdout format raw local0
        maxconn 1024
      defaults
        log     global
        mode    tcp
        timeout connect 10s
        timeout client 36h
        timeout server 36h
        option  dontlognull
      listen ssh
        bind *:22
        balance leastconn
        mode tcp
        option tcp-check
        tcp-check expect rstring SSH-2.0-OpenSSH.*
        server host1 uan01:9001 check inter 10s fall
     2 rise 1
```

Listing 3. HAProxy Configuration

The Ansible role *uan_haproxy* deploys the instances of HAProxy as configured to K3s, including a configurable jinja2 template for Helm values.yml. In the *haproxy-uai* example (See Listing. 3), the HAProxy configmap is shown with a single, generic ”uan01:9001” as an example. This example implies there is an instance of SSHD running on the node uan01 listening on port 9001.

As shown in the next section, it is up the configuration of SSHD to initiate the Podman container in which to place the user.

There are numerous ways to configure HAProxy. For more advanced use cases, HAProxy supports a Lua-based plugin infrastructure that could be leveraged in the future [8].

There are also numerous load balancing algorithms beyond *leastconn* for fine grain control to route SSH connections [9].

I. Configuration of SSHD

The Ansible role *uan_sshd* will create the SSHD config file and systemd unit file needed to provide the backend to instances of HAProxy. An example configuration will look similar to:

```
uan_sshd_configs:
  - name: "uai"
    port: "9001"
    state: "started"
    config: |
      Match User *
        AcceptEnv DISPLAY
        X11Forwarding yes
        AllowTcpForwarding yes
        PermitTTY yes
        ForceCommand podman run -it --root /scratch/
    containers --userns=keep-id --network=host -v /
    lus:/lus -v /home/users:/home/users -e DISPLAY=
    $DISPLAY registry.local/cray/uai-podman:1.0
```

Listing 4. SSHD Configuration for Podman

This will create the files */etc/ssh/uan/sshd_uai_config* and */usr/lib/systemd/system/sshd_uai.service* and set the systemd state based on the state value. As shown in the *ForceCommand* field, any user that connects to this SSHD instance will be

placed into a Podman container.
The arguments *–userns=keep-id* and *–network=host* in this example are necessary to support Slurm job launch. Slurm uses the HSN and expects that the UID of the user matches its awareness of user IDs.

J. Configuration of Podman

Some aspects of Podman will need to be configured in order to use rootless containers as an interactive environment for users. This includes configuring the image storage location and preparing subuid and subgid files. Additionally, some limitations of rootless Podman containers are outlined below.

1) Container Images: Currently, Podman does not support container image storage on network filesystems by default. This means that users are not able to store images on NFS or Lustre. For managed nodes, disk-based filesystems will need to be used as a default location for storing Podman images. At the time the Podman containers are started, the following will be used to restrict the container root to a disk-based filesystem (while still being configurable if this behavior needs to be overwritten):

```
podman --root {{ path to disk }}/containers/
```

Listing 5. Podman Configuration for Disk Use

2) Subuid and Subgid: In order to emulate privileges in a container context using user namespaces [10], Podman requires that the user have appropriate entries set in */etc/subuid* and */etc/subgid* [11]. The format of these files is:

```
[uid|name]:[subordinate uid|gid ID]:[subordinate
    range]
```

Listing 6. Subuid and Subgid format

If a user is not listed in the subuid and subgid files, they will not be able to start rootless containers.

The entries for each user should remain static as migrating ranges is not easily supported for existing container images. As there is no standard support for subuid or subgid support in System Security Services Daemon (sssd) to dynamically populate subuid and subgid, administrators will be required to generate appropriate subuid and subgid files and maintain them over time.

A simple python tool was written to generate subuid and subgid files for all users in a Lightweight Directory Access Protocol (LDAP) server, as an example of how administrators might programmatically generate these files [12]. A new S3 bucket will be created for CSM systems to host these files, and an Ansible role would be able to download these files to support rootless Podman containers.

3) Podman Lifecycle Management: In the initial implementation, it will be up to the configuration of Podman and the SSHD *ForceCommand* option to handle Podman timeouts and reentering existing Podman containers on subsequent SSH connections.

A forward-looking design will use Open Container Initiative (OCI) hooks for registration and termination of rootless Podman containers started via HAProxy. This will allow for a new service to report currently running Podman containers to fill the gap presented by the lack of a ”UAS Manager” service. While not part of this design, the new service could potentially operate as a lifecycle manager of Podman pods. These OCI hooks will be an important mechanism to introduce the concept of UAI Classes into rootless Podman containers. Each type of rootless Podman container would be registered with following information at a minimum:

- UUID representing the ”class”
- Podman friendly name or container ID
- Managed node running the container
- User ID/Name

Registration of rootless Podman containers into a centrally managed service will also be integral into reusing already running Podman containers. While it’s easy enough to use *podman ps* and *podman exec* to reenter a locally running Podman container, traversing to another managed node to *podman exec* will require information on running containers that spans the managed nodes.

4) Podman Limitations: There are some aspects of rootless Podman containers that will pose challenges. However, initial testing with the Reframe [13] HPC test suite has shown generally positive results.

a) nobody files: For example, when a Lustre filesystem is mounted into a Podman container (*-v /lus/snx110102:/lus/snx110102*), the user namespace for the container shifts UIDs and GIDs outside the standard range that Lustre understands. The result is that files will show as being owned by ”*nobody*”. The POSIX file permissions are still correct, and users can read and write to appropriate files, but it could be an impediment to user collaboration.

b) scp: As the ingress for Podman is presented as SSH, users will expect that SCP works. During testing, there has been some limitations getting this to work appropriately. *SSH_ORIGINAL_COMMAND* is being passed into Podman which allows for commands such as ”*ssh uai.hermod.can.dev.cray.com srun hostname*” to work. It has only partially worked for SCP which uses the same mechanism. What has been observed, is that the file is created in the Podman container, but no data is written to it.

Another strategy that could be considered is checking if the program passed via *SSH_ORIGINAL_COMMAND* is ”scp” and running scp as if it were targeting the host node. This would only work for target paths that are mounted into the Podman container and it could be unintuitive if files are sent, but not present in the container. If that were the case, an error would need to be returned rather than silently sending files to a path not present in the container.

Yet another solution may involve sending files to a temporary path on the host as an intermediate step, and then using *podman cp* to transfer files to the destination. Ideally scp should work seamlessly through *SSH_ORIGINAL_COMMAND*, but there are options to pursue if the standard method does not work.

K. DNS Support

Domain Name System (DNS) support for the services running in K3s will be rolled out in a phased approach. Initially, DNS records for the services would need to be added through the System Layout Service (SLS) in CSM. The CSM deployment *cray-dns-unbound-manager* will see these new entries and create the necessary DNS records. This has two drawbacks:

1) A dependency on CSM is introduced.
2) The next time the services is redeployed, a new IP may be handed out by MetalLB and the DNS record will be wrong.

Each of these issues can be addressed by introducing two DNS services to the K3s cluster to mirror a subset of the capabilities of CSM. The first of which, is ExternalDNS. Through service annotations, a DNS record can be associated to a particular service running in K3s, and ExternalDNS running in K3s may forward to PowerDNS running in CSM. This will solve the second issue of proper service discovery, but it will still be relying on services running in CSM. To connect the K3s ExternalDNS to the CSM PowerDNS, a K8s secret representing the PowerDNS API token from CSM will need to be imported to K3s. This will be handled during CFS Node Personalization.

After ExternalDNS is working for service discovery in K3s, PowerDNS will be added to the K3s cluster and would then need to be configured to the site DNS resolver. This would provide standalone DNS support in the K3s cluster for the instances of HAProxy and other services that may follow (JupyterHub, logging, etc).

III. Usage Examples

This example shows a simple configuration where users are directed through HAProxy, into a rootless Podman container that can interact with Slurm and run jobs:

```
$ ssh uai.can.hermod.dev.cray.com

alanm@uai:/> module list -t
Currently Loaded Modulefiles:
craype-x86-rome
libfabric/1.12.1.2.2.0.0
craype-network-ofi
perftools-base/23.03.0
xpmem/2.6.2-2.5_2.14__gd067c3f.shasta
cce/15.0.1
craype/2.7.20
cray-dsmml/0.2.2
cray-mpich/8.1.25
cray-libsci/23.02.1.1
PrgEnv-cray/8.3.3

alanm@uai:~> cd /lus/snx11010/alanm/
alanm@uai:/lus/snx11010/alanm> ls
bin  mpi_hello  mpi_hello.c

alanm@uai:/> sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
workq*       up   infinite     18   idle nid
    [000001-000018]

alanm@uai:/> srun -N4 mpi_hello
Hello world!  I am rank 2 of 4
Hello world!  I am rank 1 of 4
Hello world!  I am rank 3 of 4
Hello world!  I am rank 0 of 4

alanm@uai:/> exit
Connection to uai.can.hermod.dev.cray.com closed
```

Listing 7. Example Connecting to a Podman Container

Other options include access to Podman containers that support GPUs using of Podman's *–device* parameter:

```
$ ssh uai-gpu.can.hermod.dev.cray.com

alanm@uai-gpu:/> nvidia-smi -L
GPU 0: NVIDIA A40 (UUID: GPU-2deb83b0-a7a0-d113-475f
    -2f29b1bfd300)
alanm@uai-gpu:/> exit
Connection to uai-gpu.can.hermod.dev.cray.com closed
```

Listing 8. Example Connecting to a GPU Podman Container

In each of these examples, the SSH connections was received by HAProxy initially and forwarded to an SSHD instance that used Podman to start a container. Using Podman command line arguments, the hostname of the container was changed to match the type of environment being emulated by Podman (*uai* and *uai-gpu*).

IV. Future Considerations

A non-exhaustive list of other uses for this design, are explored below. While there are no firm plans to pursue any one of these, it is included to show how K3s on managed node hardware may be leveraged in the future, and that there are significant advantages to this approach.

A. JupyterHub

Ultimately, rootless Podman containers are one type of interactive environment for user activities. Adding support for K3s and MetalLB to managed nodes enables other runtimes like JupyterHub to be added easily. JupyterHub is included in this discussion to show that K3s and MetalLB introduces the flexibility to add new services to managed nodes quickly.

JupyterHub can be installed with the default configuration, using Helm [14]:

```
helm repo add jupyterhub https://jupyterhub.github.
    io/helm-chart/
```

```
helm upgrade --install jupyterhub jupyterhub/
    jupyterhub \
    --namespace jupyter \
    --create-namespace \
    --values config.yaml
```

Listing 9. Installing JupyterHub with Helm

Unlike Podman containers running outside the control of K3s, JupyterHub deployed in this manner would allow users to interact with the system using a browser UI, with corresponding K3s pods being run in the namespace *jupyter*. More configuration and testing would be required to make this a useful environment, but the ease in which K3s allows it to be installed makes this a interesting area to pursue.

B. Cross System Scheduling

HAProxy allows for many different load balancing configurations. While it would require coordination of matching configurations between various systems, it could be possible to have other managed nodes on different systems allow for connectivity from a single ingress. A theoretical use case might involve a failover managed node on another system in the event that all of the other managed nodes are not available. This example HAProxy configuration could illustrate how that might look:

```
listen ssh
    bind *:22
    balance leastconn
    mode tcp

    server hermod_uan01 uan01.chn.hermod.dev.cray.
    com:9001 check inter 10s fall 2 rise 1
    server hermod_uan02 uan02.chn.hermod.dev.cray.
    com:9001 check inter 10s fall 2 rise 1
    server loki_uan03   uan03.chn.loki.dev.cray.com
    :9001   check inter 10s fall 2 rise 1
```

Listing 10. HAProxy using Multiple Systems

If the nodes uan01 and uan02 on hermod are down, uan03 on loki might still be available. This approach might also work well to pool specific hardware requirements like processors or GPUs into a single SSH ingress across more than one system.

C. Alternatives to Podman

Instead of Podman containers, virtual machines (VMs) or K3s pods could be used as the interactive environment in which users could operate.

V. Summary

This paper has shown that new software components like K3s, MetalLB, HAProxy, and Podman can be used to evolve the current design of temporary, interactive container environments such as UAIs into one that does not rely on a cluster of management nodes. Instead, an independent cluster using K3s can be formed to isolate user processes onto managed nodes where user activity already occurs. This cluster can also host a variety of forward-looking designs like JupyterHub, VMs, or represent a single SSH ingress across multiple systems.

References

[1] CUG 2022 Best Paper, *https://cug.org/cug-2022-technical-program/*.
[2] Macvlan vs IPvlan, *https://ipwithease.com/macvlan-vs-ipvlan-understand-the-difference/*.
[3] MetalLB Universe, *https://metallb.universe.tf/*.
[4] Podman, *https://podman.io/*.
[5] HAProxy, *https://www.haproxy.org/*.
[6] PowerDNS, *https://www.powerdns.com/*.
[7] Open Container Initiative, *https://opencontainers.org/*.
[8] HAProxy Lua Support, *https://www.haproxy.com/documentation/hapee/latest/api/lua/scr*
[9] HAProxy Load Balancing Algorithms, *https://docs.haproxy.org/2.7/configuration.html#4.2-balance*.
[10] user_namespaces(7), *https://man7.org/linux/man-pages/man7/user_namespaces.7.html*.
[11] subuid subgid, *https://wiki.gentoo.org/wiki/Subuid_subgid*.
[12] Subordinates repository, *https://github.com/alanm-hpe/subordinates*.
[13] HPC Testing Framework Reframe, *https://reframe-hpc.readthedocs.io/en/stable/*.
[14] Zero to JupyterHub, *https://z2jh.jupyter.org/en/latest/jupyterhub/installation.html*.