# UAIs Come of Age: Hosting Multiple Custom Interactive Login Experiences Without Dedicated Hardware

Eric Lund
*Hewlett Packard Enterprise*
Bloomington, MN
eric.lund@hpe.com

*Abstract*— **On HPE Cray EX systems, User Access Instances (UAIs) provide convenient lightweight temporary single-user interactive login environments. Access to a set of End-User UAIs is mediated by a Broker UAI which presents an SSH login to users. On successful login, the Broker UAI locates or creates an End-User UAI as needed for the user, then redirects the user to the End-User UAI. On reaching the End-User UAI the user sees a login environment that supports interactive work and launching of batch workloads.**

**Using UAI Classes, Custom UAI container images, and Kubernetes Volumes registered with the User Access Service (UAS), a site can configure a rich variety of End-User UAIs tailored to specific workflows and define multiple Broker UAIs, each providing access to a distinct set of specifically tailored End-User UAIs. Access to each set of End-User UAIs can be individually controlled by configuring the associated Broker UAI.**

**UAIs are a powerful tool for isolating multiple missions and workflows without the need for dedicated hardware investment. This paper describes the Broker UAI / End-User UAI structure and relationship, examines common configurations, and explores common uses for UAIs.**

***Keywords—User Access, User Access Instance, User Environment, User access Node, Custom User Access, UAI, UAN, HPE Cray EX***

## I. Introduction

Both User Access Nodes (UANs) and User Access Instances (UAIs) provide interactive login access to HPE Cray EX systems. UANs are dedicated hardware nodes configured for general purpose multi-user access, while UAIs are disposable containerized single-user login platforms that can be configured for specific uses.

UANs are well suited for stable long-running tasks or for handling periodic background activities that require a stable always-on platform to host them. UANs also typically have access to disk, memory and swap resources that make them well suited to resource intensive tasks. UANs can be customized based on Compute Node installations, including access to physical resources like GPUs, to permit building software in a programming environment and hardware environment that matches the runtime environment in which the software will execute.

UAIs on the other hand appear on-demand and can run shorter- or longer-term tasks without requiring a permanent presence on the system. UAIs connect to external storage by mounting the storage within their containers but contain no persistent state of their own. As a result, UAIs are disposable, and easily come and go without loss of user data. Like UANs, UAIs can be customized based on compute nodes from a software perspective. This allows the use of a programming environment that matches compute nodes. Unlike UANs, however, UAIs currently lack the ability to access native host features like GPUs and swap. For builds that require GPU access or that cannot run in available physical memory, UAIs are not well suited. Despite this limitation, UAIs are easily customized to a wide range of uses and can be assigned targeted resource requirements and limits to tailor their resource consumption to their intended purpose. This, combined with Kubernetes scheduling based on resource availability, allows UAIs to present a highly elastic interactive user login experience.

UANs are best suited to long-running swap or other native resource intensive tasks while UAIs are more suited to more flexible, cloud-like interactions. Between the extremes of this spectrum, lies a great deal of overlap between what can reasonably be done with UAIs and what can be done with UANs. The elastic on demand nature of UAIs gives sites flexibility in supporting a wide range of tailor-made environments without dedicating hardware to any given environment.

UAN usage and management is more traditional, and, therefore, more familiar than UAIs. This paper focuses nearly entirely on UAIs, their implementation, administration and uses.

## II. UAI Implementation

The HPE Cray EX system provides the User Access Service (UAS) which manages UAIs. HPE Cray EX systems use two broad categories of UAIs to implement the on-demand UAI functionality:

- Broker UAIs
- End-User UAIs.

This section examines the relationships between the UAS and UAIs and the structure and implementation of Broker and End-User UAIs.

### A. *UAI / UAS Components and Relationships*

Fig. 1 shows the UAS / UAI components in relation to each other. The UAS, Broker UAIs, End-User UAIs and SSH Key Management all run within a Kubernetes cluster on top of Non-Compute Nodes (NCNs), specifically Kubernetes worker nodes. Users reach Broker UAIs through SSH on an external IP ingress address allocated for the Broker UAI by the HPE Cray EX system. A Broker UAI authenticates a user's session using the authentication / authorization rules configured for the broker by an administrator.

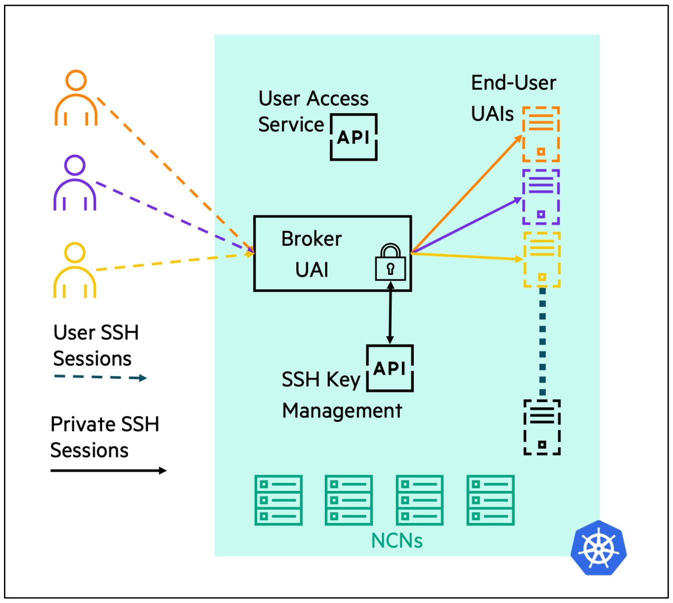

Fig. 1. UAS/UAI Components and Relationships

Upon successful authentication, the SSH session executes the *switchboard* command to locate or create an End-User UAI belonging to the user. If there is no available End-User UAI, first *switchboard* creates an SSH key pair known only to the Broker UAI and places the keys in SSH Key Management. After that, *switchboard* invokes the UAS to create a new End-User UAI, passing the newly created public SSH key to the End-User UAI as the authorized key for the user. Finally, *switchboard* establishes a private internal SSH connection to the End-User UAI and forwards all SSH traffic over that connection. The End-User UAI uses the authorized public key described above to authenticate the connection and starts a login session for the user.

At the end of the login session, the user's private SSH connection drops causing *switchboard* to exit and drop the user's SSH connection to the Broker UAI. Unlike historical *switchboard* use on UANs, *switchboard* on the Broker UAI runs non-interactively to completion. At no time does the user have uncontrolled interactive access to anything on the Broker UAI.

### B. *Structure of UAIs*

Fig. 2 illustrates the structure of Broker and End-User UAIs. All UAIs consist of three main Kubernetes resources:

- A Kubernetes service
- A Kubernetes job
- One or more Kubernetes (replica) pods

The Kubernetes service provides the UAI with its network ingress (IP address) and, in the case of UAIs that have external IP addresses, an External DNS host name composed by the HPE Cray EX system. The Kubernetes service also forwards load-balanced network connections to the Kubernetes pod(s) in the UAI.

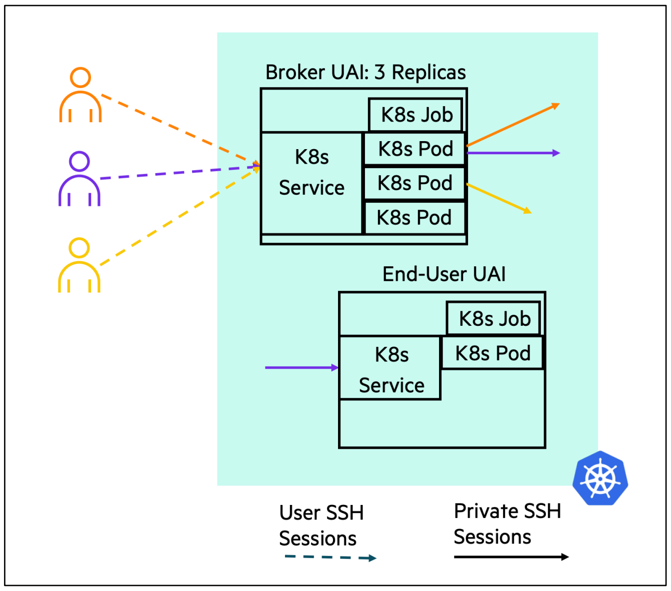

Fig. 2. Structure of UAIs

The Kubernetes job orchestrates the lifecycle of the pods, restarting them as needed until one or more pods reach a successful completion. When one or more pods reach successful completion, the job completes signaling end-of-life for the UAI. When a UAI reaches its end-of-life, the UAS detects the completion of the UAI job and removes all components of that UAI.

Kubernetes schedules the pods in the UAI. The pods in turn execute the containerized UAI environment. What takes place in the UAI is dictated by the UAI container image (UAI image) and the container entry point specified in the pod. Composition of the UAI image is the first layer of UAI specialization. The pod also specifies a set of Kubernetes volume mounts for the container that provide a second, run-time, layer of specialization.

On a Broker UAI, the entry point does two things:

- Starts the SSSD/LDAP client for user credentials
- Starts SSHD to begin accepting user logins

Once running, the Broker UAI entry point loops (sleeping) indefinitely without ever completing. Broker UAIs run until they are administratively deleted through the UAS.

Prior to starting a Broker UAI, administrators customize the SSSD/LDAP and SSHD configuration of the Broker UAI to use the appropriate directory server and to authorize users based on appropriate rules for the kind of End-User UAI the Broker UAI provides. All this customization happens using Kubernetes volume mounts, allowing use of a common Broker UAI image for most purposes.

The entry point for an End-User UAI also starts SSHD but does not start SSSD/LDAP. The only user the End-User UAI knows is the user for whom it was created, and that user is pre-authorized to log into the End-User UAI at creation. When the correct user initiates a private SSH connection, therefore, the End-User UAI authenticates and authorizes the user and creates a new login session. The entry point for the End-User UAI also loops (sleeping), but during that loop it keeps track of timeout conditions. When an appropriate set of timeout conditions is met, the End-User UAI entry point exits with success (completes), causing the job to complete which triggers UAI cleanup in the UAS.

Beyond functional differences between Broker and End-User UAIs, there are structural differences. While an End-User UAI always has a single Kubernetes pod in its set of pods, a Broker UAI may have any number of replica pods. The service component of a Broker UAI load-balances incoming connections across the replica pods in the Broker UAI. This provides both resilience to pod failure or pod migration and better distribution of network traffic and SSH session processing load within the Broker UAI, allowing the Broker UAI to handle many simultaneous SSH connections efficiently and reliably.

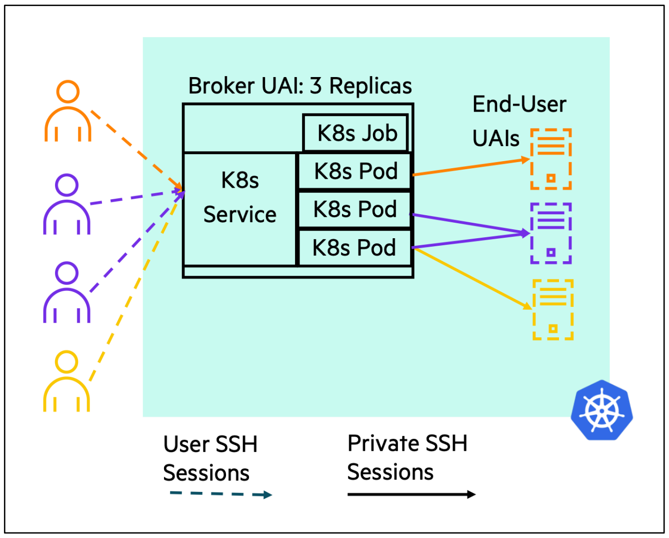

Fig. 3. Broker UAI Connection Load Balancing

Fig. 3 presents a Broker UAI with 3 replica pods and four incoming connections. Notice that two of the incoming connections come from the same user: Purple. Two things to notice about this are

- Two different pods in the Broker UAI handle Purple's two SSH sessions
- Both Broker UAI pods connect Purple to the same End-User UAI.

Whenever a user logs into a Broker UAI the Broker UAI tries to find and re-use an existing End-User UAI before creating a new one. This means there will only ever be a single End-User UAI for each user under the control of a given Broker UAI.

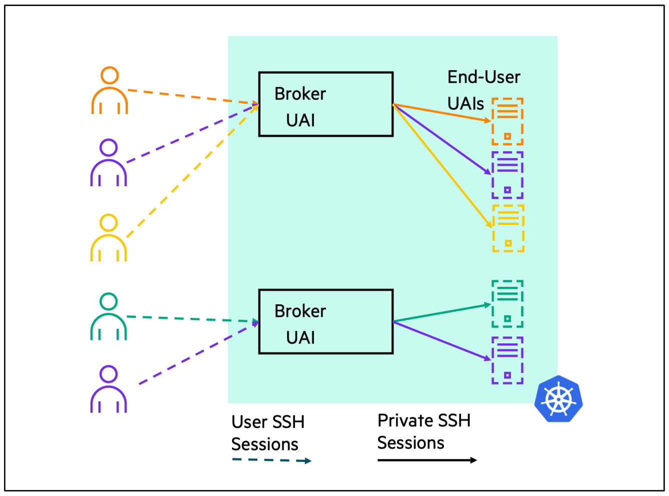

Fig. 4. Multiple Different Broker UAIs

Fig. 4 depicts an HPE Cray EX system running two different Broker UAIs. Each broker UAI has its own set of End-User UAIs and may have its own authentication/authorization rules or domain, and other different properties. Notice that user Purple is logged into both the top Broker UAI and the bottom Broker UAI. Each Broker UAI has a distinct End-User UAI for Purple to use. An HPE Cray EX system can run as many different Broker UAIs as are needed to support different workflows or audiences hosted by the system. Each Broker UAI has its own configuration and creates its own class of End-User UAI. All of this is managed through the configuration of UAIs in the UAS.

## III. UAI Administration

UAI administration consists of configuring the UAS to create different kinds of UAIs and managing the life cycle of Broker UAIs.

### *A. Configuration*

Configuration focuses on four configuration object types in the UAS:

- UAI image registrations
- UAI volumes
- UAI resource specifications
- UAI classes

### 1) *UAI Image Registrations*

UAIs run in containers, and containers execute container images. The UAS keeps a configured list of registered container images that can be used to create UAIs (UAI image registrations). The UAS assigns an image ID in the form of a UUID to each UAI image registration as it is created. This image ID is specified when using that image to configure a UAI Class.

### 2) *UAI Volumes*

UAI volumes offer the most flexible way to customize a UAI and the only way to connect a UAI to external persistent storage. A UAI volume describes a named source of data and a mount-point within the UAI container's filesystem where that source of data should be mounted as a file or directory. A UAI volume can describe anything that can be used as a Kubernetes volume. For more information on Kubernetes volumes, see "Kubernetes Volumes" under the "Resources" section below.

Uses of UAI volumes include customization of SSSD/LDAP, SSHD and so forth in Broker UAIs, connecting End-User UAIs with the HPE programming environment and analytics software packages, and connecting End-User UAIs with Kubernetes secrets and configmaps used to interact with workload managers like Slurm or PBS. Finally, UAI volumes can connect End-User UAIs to home directories or Lustre filesystems for shared user storage.

```
{
  "mount_path": "/etc/localtime",
  "volume_description": {
    "host_path": {
      "path": "/etc/localtime",
      "type": "FileOrCreate"
    }
  },
  "volume_id": "7f115bb6-2150-4829-
80d7-1d7bd81f748a",
  "volumename": "timezone"
}
```

Fig. 5. Example UAI Volume

Fig. 5 shows a UAI Volume to mount the file */etc/localtime* from the host node at */etc/localtime* on the UAI using the volume name *timezone* inside the pod specification for the UAI. The Volume ID gives UAI classes a way to refer to the UAI volume so that UAI Volumes can be used to construct UAI classes.

### 3) *UAI Resource Specifications*

Kubernetes places and schedules pods based on resource availability. The UAS keeps a list of configured UAI resource specifications that can be used to request specific resources for a given class of UAIs. While a UAI resource specification can contain any resource limit or request recognized by Kubernetes, the most common resources are CPU and memory. For more information on Kubernetes resource limits and requests see "Kubernetes Resource Limits and Requests" under the "Resources" section below.

```
limit = {
    "cpu": "300m",
    "memory": "250Mi"
}
request = {
    "cpu": "300m",
    "memory": "250Mi"
}
resource_id = "a47847a8-50f9-4c3d-8513-
94e18ef78d49"
```

Fig. 6. Example UAI Resource Specification

Fig. 6 shows an example UAI resource specification. Notice that this example contains a *request* part and a *limit* part. A UAI resource specification can have either or both. The example shows a *request* for 300 millicpus (0.3 CPUs) and 250 Mibibytes of memory. It shows a *limit* of the same amounts of each. The example also shows the resource ID that identifies this UAI resource specification for use in constructing UAI classes.

While it is legal to set up a request that is smaller than the limit for a UAI resource specification, doing so may cause any UAI to which the specification is applied to grow into its limits and oversubscribe the host node where the UAI is deployed. When that happens, Kubernetes may seek to place that UAI or other UAIs on a less loaded node. This results in temporary termination and re-scheduling of a UAI which appears as instability to the user of the UAI. To avoid such instability, keep the requested resources the same as the resource limits in UAI resource specifications.

### 4) *UAI Classes*

UAI classes define the construction details for making a UAI. UAI classes combine a UAI image, an optional UAI resource specification, and a list of UAI volumes with other configuration details to describe the desired behavior of a class of UAIs. In addition to the items mentioned already, some of the items that can be configured are:

- An End-User UAI creation class ID for Broker UAIs
- A replica count
- Timeouts for End-User UAIs
- An internal or public ingress IP indication
- Kubernetes host node tolerations

A UAI class contains several other configuration items not described here. For more information on UAI class content see the link under "UAS / UAI Documentation" in the "Resources" section below.

The End-User UAI creation class ID in a Broker UAI class tells the Broker UAI what kind of End-User UAI to create when creating an End-User UAI for a newly logged in user. It refers to the UAI class that describes the configuration of the End-User UAIs reached through that broker. This is how different Broker UAIs can each serve different End-User UAI functionality.

The replica count for a Broker UAI controls the number of replicas the Broker UAI will request from Kubernetes on creation. It should reflect the desired amount of load balancing or redundancy for the given class of Broker UAIs. Kubernetes places each replica on a host node separate from all other replicas in the Kubernetes cluster. If a node fails or loses network connectivity with the cluster, the UAI connections through that Broker UAI replica drop, but users can reconnect immediately using a different replica and continue operation. Another advantage of replicas is that they spread the work and network traffic associated with hosting UAI connections across a larger set of pods to increase overall capacity and throughput.

Using replica pods with End-User UAIs does nothing to improve the End-User UAI stability or capacity, and it wastes resources. HPE recommends keeping the replica count for End-User UAIs set to the default value of 1.

UAI Classes for End-User UAIs can use two kinds of timeouts:

- Hard timeouts
- Soft timeouts

A hard timeout, if present, specifies the overall length of time the End-User UAI is allowed to run before terminating and being cleaned up. When an End-User UAI reaches its hard timeout, the entry point to the End-User UAI Pod exits immediately with success causing the UAI job to complete and the UAS to clean up the UAI. An administrator can also configure a warning interval which will cause a message to be sent to all logged in sessions some number of seconds before a hard timeout indicating impending termination.

A soft timeout, if present, specifies the length of time an End-User UAI can run before terminating because it either is idle or becomes idle. When an End-User UAI reaches its soft timeout, the entry point starts looking periodically for opportunities to exit with success causing the UAI Job to complete and the UAS to clean up the UAI. If the entry point finds a non-zero number of login sessions in the End-User UAI, there is no opportunity to exit. If the number of logged in sessions is or becomes zero after the soft timeout expiration, an opportunity to exit occurs and the UAI will complete.

Hard timeouts are intended to ensure that UAIs left with logged in sessions for extended periods of time are removed to free resources for other users. Soft timeouts are intended to ensure that idle End-User UAIs get cleaned up in a timely fashion, whatever that may mean for the tasks the UAIs were created to support.

UAIs can present an SSH ingress either publicly, through the HPE Cray EX system Customer Access Network (CAN) or the HPE Cray EX Customer High Speed Network (CHN) [1], or internally across the Kubernetes cluster's internal network. Broker UAIs usually present SSH ingress publicly to let the Broker UAI act as a gateway to End-User UAIs. End-User UAIs usually present SSH ingress privately to prevent unmediated external access by users. When a UAI presents SSH publicly the HPE Cray EX system assigns it an IP address on the CAN or CHN and composes an External DNS hostname to allow the UAI to integrate into site DNS. The internal or public IP indication in the UAI Class controls whether a UAI presents a public or private SSH ingress.

Kubernetes places pods on host nodes based on pod *affinity* and *anti-affinity* and, also, based on host node *taints* and pod *tolerations*. A pod may be assigned explicit affinity for a given set of nodes based on flags or annotations maintained in those nodes by Kubernetes. Kubernetes schedules a pod with such an affinity on those nodes. A pod may also have an anti-affinity for certain nodes based on the same criteria. Kubernetes does not schedule pods with such an anti-affinity on those nodes. All UAIs have an anti-affinity for any node with the label:

*uas=False*

Assigning this label to HPE Cray EX Kubernetes worker nodes that run HPE Cray EX CSM management services prevents any UAI from being scheduled on those worker nodes.

In addition to explicit affinity or anti-affinity, Kubernetes also permits a node to be *tainted*. A tainted node carries a *taint* which prevents any Pod that does not carry a corresponding *toleration* from being scheduled on that node. By default, all UAIs have the toleration

*uai_only*

This means that nodes tainted with *uai_only* only host UAIs and certain Kubernetes internal pods that are not affected by taints. Most importantly, nodes with this taint, will *not* host CSM management services. By applying the *uas=False* label to Management Kubernetes worker nodes and the *uai_only* taint to UAI Host Kubernetes worker nodes, a site can isolate all UAIs to worker nodes that do not carry CSM loads.

Beyond these defaults, though, UAI Classes may contain optional site specified tolerations to allow fine grained control of UAI pod placement. By tainting nodes and assigning tolerations to UAI classes a site may separate UAIs based on their intended use, their intended audience, or any other criteria a site chooses.

For more information on Kubernetes affinity/taints see "Kubernetes Node-Affinity" in the "Resources" section below.

### *B. Managing Broker UAIs*

Broker UAI management uses the Cray CLI (*cray* command), which may be run from any node or host that has access to the CSM management API. The examples in this section do not specify a context for running the *cray* command, but simply provide example command lines. The focus here is on command line content, not execution context.

#### *1) Creating a Broker UAI*

Administrators create Broker UAIs using a command of the form shown in Fig. 7:

```
cray uas admin uais create \
  --class-id \
  a9d61724-976a-4a07-85f8-00e422bff3ce \
  --uai-name workload-monitoring-uai \
  --owner workload-monitoring-broker
```

Fig. 7. Example Broker UAI Creation Command

The only required parameter is the Broker UAI class ID used to create the Broker UAI, specified here as

*–class-id a9d61724-976a-4a07-85f8-00e422bff3ce*

The command shown also assigns a UAI name:

*--uai-name workload-monitoring-uai*

used to construct a custom External DNS name for the Broker UAI so that the Broker UAI has a name as soon as it is created and there is no need for the site to track the automatically assigned external IP address for the Broker UAI.

Finally, this command assigns an owner:

*--owner workload-monitoring-broker*

On systems with many active UAIs, listing or deleting UAIs can be overwhelming. Assigning a descriptive owner to Broker UAIs allows administrators to filter by owner and list or delete only the desired Broker UAI.

### *2) Examining Broker UAIs*

Administrators examine the status of UAIs using a command of the form shown in Fig. 8:

```
cray uas admin uais list \
  --owner workload-monitoring-broker
```

Fig. 8. Example Command to List UAIs Filtered by Owner

Without the *--owner* option, the above command would list all UAIs on the system. By specifying the *--owner* option along with the owner's name used to create the Broker UAI, the administrator can see only the Broker UAI(s) owned by that owner.

### *3) Deleting a Broker UAI*

Administrators delete a Broker UAI using a command of the form shown in Fig. 9:

```
cray uas admin uais delete \
  --owner workload-monitoring-broker
```

Fig. 9. Example Command to Delete UAIs Filtered by Owner

Here, if the *--owner* option were not specified, the command would require an explicit list of UAIs to delete or some other criterion by which to delete UAIs. By specifying the *--owner* option along with the owner's name used to create the Broker UAI, the administrator can delete all Broker UAIs created with that owner without affecting other UAIs or needing to search for UAIs to delete.

## IV. Example UAI Use Cases

There are two major considerations when customizing Broker and End-User UAIs. The first consideration primarily drives the customization of Broker UAIs: *intended audience*. The second consideration drives the customization of End-User UAIs: *intended function*. Administrators manage intended audience by customizing authentication rules in Broker UAIs. If a user cannot log into a given Broker UAI, that user cannot access an End-User UAI for a given audience. Carefully selecting End-User UAI Volumes to limit access to shared storage and tailoring taints and tolerations to control node placement can further separate End-User UAIs according to intended audience, so there can be End-User UAI considerations regarding audience as well. Administrators tailor End-User UAIs to their intended function by customizing UAI image content and UAI volumes along with UAI Resource Specifications and Timeouts.

The following are several hypothetical UAI use cases and the customizations likely to support those use cases. These examples are here to spark creativity not to prescribe any configuration.

### *A. Workload Development and Testing UAI*

- Intended Audience: developers
- Intended Function: software development

Software development often requires long term uninterrupted login sessions with access to development tools like a programming environment, debuggers, IDEs or editors, version control software and so forth, and access to some set of compute nodes through a workload manager. Software development also usually requires access to sample data and shared storage to facilitate collaboration between developers. Compiling software can be both memory and CPU intensive.

While software development sessions may tend to run long, once they are over, the slight overhead of restarting a new End-User UAI is easily amortized over the length of the session.

Long running large memory and CPU End-User UAIs can be resource hogs, so access to them should be limited to users who truly need those characteristics. Furthermore, such UAIs may need to be isolated away from production host nodes.

Likely customizations for this use case are:

- Broker UAI: authorize only developers
- End-User UAI
  - Resource Limits
    - Memory: large or very large
    - CPU: large
  - Use Taints and Tolerations
    - Direct scheduling to higher capacity host nodes
    - Keep development activity away from production host nodes
  - Timeouts
    - Hard: long or none
    - Soft: short
- Programming environment tools in either the image or mounted volumes
- Development storage access through volumes
- Workload Manager access for launching and monitoring test runs

### *B. Production Workload Launch UAI*

- Intended Audience: production workload managers
- Intended Function: production launch

Workload launch requires access to production data, production software and workload management commands able to launch on production compute nodes. Launching a workload does not take a lot of time and probably does not happen frequently. Workload management commands do not require much CPU or memory. Production activities should not be forced to contend for resources with non-production activities.

Likely customizations for this use case are:

- Broker UAI: authorize only production workload managers
- End-User UAI
    - Resource Limts
        - Memory: small
        - CPU: small
    - Taints and Tolerations direct scheduling to production host nodes
    - Timeouts
        - Hard: modest
        - Soft: short
- Production software either in the image or accessed through mounted volumes
- Production storage accessed through mounted volumes
- Workload Manager access for launching

### C. *Production Workload Monitoring UAI*

- Intended Audience: production workload managers
- Intended Function: production monitoring

Workload monitoring requires access to production workload management commands able to gather job status. Monitoring does not require access to production software or data. Individual monitoring operations do not take a lot of time, but they are likely to be scripted meaning that they will likely run frequently, making End-User UAI start latency an issue, and work best if End-User UAIs do not terminate unexpectedly. Workload management commands do not require much CPU or memory. Production activities should not contend for resources with non-production activities.

Likely customizations for this use case are:

- Broker UAI: authorize only production workload managers
- End-User UAI
    - Resource Limits
        - Memory: small
        - CPU: small
    - Taints and Tolerations direct scheduling to production host nodes
    - Timeouts
        - Hard: none
        - Soft: long
- Workload Manager access for monitoring

### D. *Batch Mode Workflow UAI*

Batch mode (crontab style) operations executed on End-User UAIs could have many different customizations depending on the activities required, but they likely have certain things in common. The use case looks a lot like Production Workload Monitoring, but, since batch mode operations most likely run less frequently, End-User UAI creation overhead is less of a problem. Memory and CPU sizing should be tailored to the specifics of the batch mode tasks.

Likely customizations for this use case are:

- Broker UAI: authorize only users who run batch mode tasks
- End-User UAI
    - Resource Limits
        - Memory: appropriate to expected background tasks
        - CPU: appropriate to expected background tasks
    - Taints and Tolerations as appropriate
    - Timeouts
        - Hard: none
        - Soft: very short
- Workload Manager access as needed

## V. Resources

All UAS and UAI related code is open source and publicly available as is the documentation for managing and using UAIs. This section provides links to the relevant repositories and documents.

### A. *UAS / UAI Documentation*

Here is a link to the current UAS/UAI documentation:

https://github.com/Cray-HPE/docs-csm/blob/main/operations/UAS_user_and_admin_topics/index.md

### B. *UAS Manager*

The UAS manager is the heart of configuration and administration of UAIs. The source code for this can be found here:

https://github.com/Cray-HPE/uas-mgr

### C. *HPE Provided UAI Images and Default Config Update*

HPE provides a basic End-User UAI Image that can be used for experimentation and basic sanity testing of the UAS/UAI mechanisms, and a Broker UAI Image that sites can use to create Broker UAIs. HPE also provides a tool that installs and updates the UAS default configuration in new releases of the UAS/UAI software. The code for these can be found here:

https://github.com/Cray-HPE/uai-images

### D. *The Switchboard Tool*

As described in "UAI Implementation" the *switchboard* tool is the heart of the Broker UAI's mechanism for delegating user sessions to End-User UAIs. The source code for that can be found here:

https://github.com/Cray-HPE/switchboard

### E. *The End-User UAI Entry Point and Other Utilities*

As mentioned in "UAI Implementation" the End-User UAI entry point script is responsible for starting SSHD and handling timeouts. The code for that and other UAI related utilities can be found here:

https://github.com/Cray-HPE/uai-util

### F. *Kubernetes Concepts*

This paper refers to several Kubernetes concepts that underly the implementation and use of UAIs. The following links provide more detail on those concepts.

#### *1) Kubernetes Pods*

https://kubernetes.io/docs/concepts/workloads/pods/

#### *2) Kubernetes Services*

https://kubernetes.io/docs/concepts/services-networking/service/

#### *3) Kubernetes Jobs*

https://kubernetes.io/docs/concepts/workloads/controllers/job/

#### *4) Kubernetes Volumes*

https://kubernetes.io/docs/concepts/storage/volumes/

#### *5) Kubernetes Resource Limits and Requests*

https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

#### *6) Kubernetes Node-Affinity and Taints*

- https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/
- https://kubernetes.io/docs/concepts/scheduling-

## VI. References

[1] Alex Lovell-Troy, Sean Lynn. 2021. User and Administrative Access for CSM-Based Systems: Network Architecture Evolution and Access Control Mechanics in Shasta v1.4 and Shasta v1.5. In *Proceedings of the 2021 Cray User Group Meeting*.