Kubernetes with Open OnDemand using Kyverno

Outline

- Overview of technologies
- Security challenges faced supporting Kubernetes with OnDemand
- Solutions to support user based “jobs” inside Kubernetes

Overview of Technologies

- Kubernetes
  - Open-source container orchestration
- Open OnDemand
  - Web interface to make HPC access easier
  - Provides a way for sites to make things like interactive jobs easy to deploy and use
  - Web processes run as logged in HPC user
  - Supports multiple resource manages: SLURM, Torque, Kubernetes, many more
- Kyverno
  - Kubernetes policy engine
  - Deploy policies using Kubernetes resources, ie standard Kubernetes YAML resources

Challenges

- Kubernetes pods can run as root
  - This can be very dangerous on systems with shared filesystems like GPFS.
- How to ensure a user running a pod is doing so using their UID/GIDs?
  - Want to ensure operations like filesystem access are taking place as that user
- How to charge users for their usage of Kubernetes similar to job charging in traditional HPC batch environment

Design patterns

- All user pods run in user specific namespace of pattern user-$USER which is bootstrapped by OnDemand at login
- RBAC for user-$USER namespaces limits user operations to just the things needed to run OnDemand jobs
- Kubernetes authenticates with Keycloak OIDC IDP and the OIDC tokens for OnDemand are allowed to be used for Kubernetes via OAuth2 audience
- Deploy job-pod-reaper tool to cleanup pods after “walltime” is reached
  - Use annotation to set what walltime should be
- Deploy k8-namespace-reaper to cleanup unused namespaces

Solutions to security using Kyverno

- Deploy policies that enforce user’s pods run as that user
  - Ensure pod user UID and GID match the requesting user based on LDAP
  - Ensure pod supplemental groups match those of user based on LDAP
  - Ensure user pods cannot escalate privileges or access host filesystems outside of filesystems needed to run OnDemand jobs
- LDAP user mapping is performed by k8-ldap-configmap tool that generates ConfigMap resources from LDAP data that Kyverno can use in policies

Solutions for accounting using Kyverno

- Deploy policies that ensure accounting is possible
  - Require pods to have account label
  - Ensure the account label is valid when compared to LDAP data
- Deploy policies that ensure controlled usage of Kubernetes
  - Ensure CPU and Memory requests and limits exist
  - Ensure pod lifetime annotation is present and set max lifetime
  - Ensure pods are pulling images from trusted image registries

Example policies - runAsUser

```
validate:
  message: >-
    Invalid user UID specified in fields
    spec.securityContext.runAsUser or spec.containers[*].securityContext.runAsUser or
    spec.initContainers[*].securityContext.runAsUser
  anyPattern:
  - spec:
      securityContext:
        runAsUser: "{{ uidMap.data.\"{{ request.object.metadata.namespace }}\" }}"
      =(initContainers):
        - =(securityContext):
            =(runAsUser): "{{ uidMap.data.\"{{ request.object.metadata.namespace }}\" }}"
      containers:
        - =(securityContext):
            =(runAsUser): "{{ uidMap.data.\"{{ request.object.metadata.namespace }}\" }}"
  - spec:
      =(initContainers):
        - securityContext:
            runAsUser: "{{ uidMap.data.\"{{ request.object.metadata.namespace }}\" }}"
      containers:
        - securityContext:
            runAsUser: "{{ uidMap.data.\"{{ request.object.metadata.namespace }}\" }}"
```

Example policies – account and supplement groups

```
  validate:
    message: "{{ request.object.metadata.namespace }} not authorized to charge against account {{
request.object.metadata.labels.account }}"
    deny:
      conditions:
      - key: "{{ request.object.metadata.labels.account }}"
        operator: NotIn
        value: "{{ userGroupMap.data.\"{{ request.object.metadata.namespace }}\" }}"


  validate:
    message: "{{ request.object.metadata.namespace }} not authorized to use those supplemental groups"
    deny:
      conditions:
      - key: "{{ request.object.spec.securityContext.supplementalGroups[*].to_string(@) }}"
        operator: NotIn
        value: "{{ userGIDMap.data.\"{{ request.object.metadata.namespace }}\" }}"
```

Example LDAP config maps

LDAP user UID map:

```
    user-tdockendorf: "20821"
```

LDAP user GID map:

```
    user-tdockendorf: "5509"
```

LDAP user GIDs map:

```
    user-tdockendorf:
'["1021","2399","3241","3285","3309","4391","4496","4547","4548","5087","5301","5325","5353","
5356","5358","5509","5527","5607","6393","6557","6558","6951","6952","6957","7175"]'
```

LDAP user groups map:

```
    user-tdockendorf: '["PZS0708","PZS0703","PAS1936","PDE0001"]'
```

Kyverno Policies

Upstream:

- https://github.com/kyverno/policies/
- https://github.com/kyverno/kyverno/tree/main/charts/kyverno-policies

OSC Policies

- https://github.com/OSC/osc-helm-charts/tree/main/charts/kyverno-policies