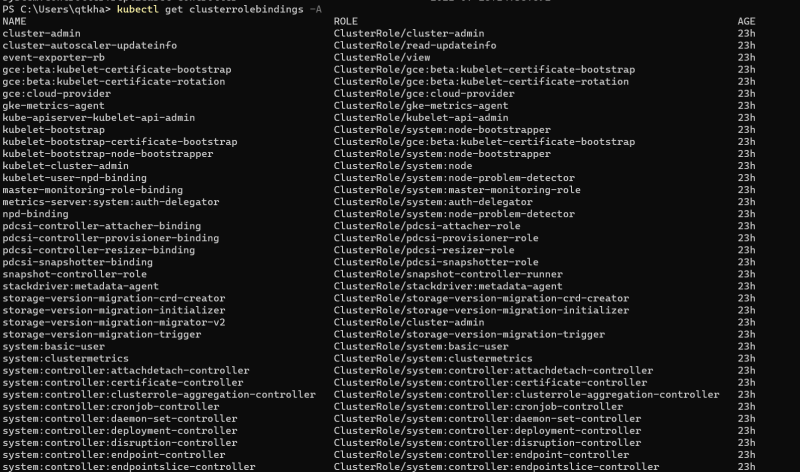
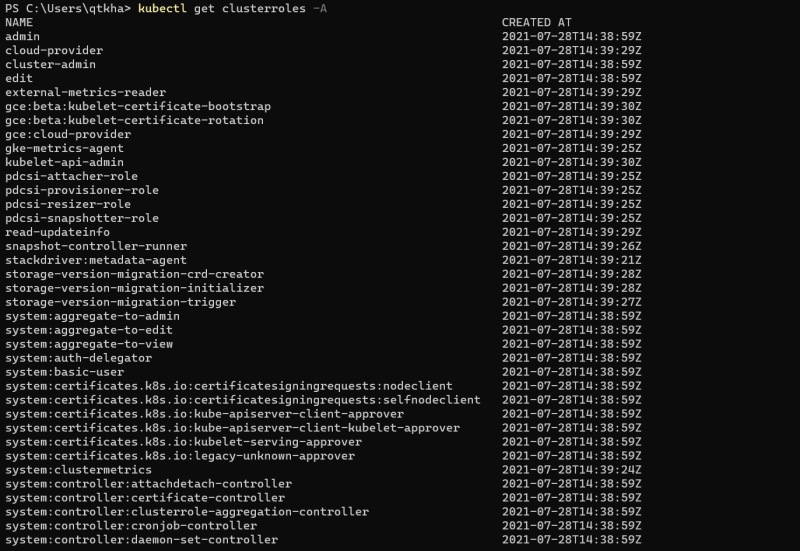
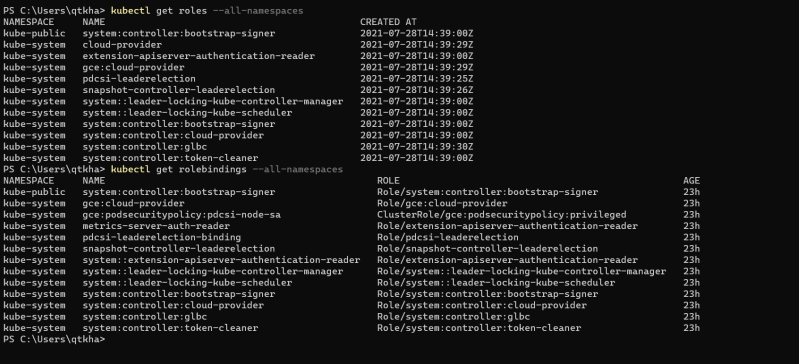
**Role Based Access Control**

**Important term: Authentication, authorization, name space and cluster.**

* To properly manage access in k8s it’s critical to understand how identity, roles and role bindings interact to control who can do what with resources.
* Identity in Kubernetes:
  + Every request that comes to k8s is associated with some identity, Event a request with no identity is associated with system:unauthenticated group.
  + k8s makes a distinction b/w user identities and service account identities
  + Service accounts are created and managed by k8s itself and are generally associated with components running inside cluster
  + User accounts are all other accounts with actual users of cluster, and often include automation like continuous delivery as a service that runs outside of cluster
  + K8s uses a generic interface for authentication providers
  + k8s supports a number of different authentication providers including
    - HTTP Basic Authentication (Largely deprecated)
    - x509 Client Certificates
    - Static token files on the host
    - Cloud Authentication providers like Azure Active Directory and AWS IAM
    - Authentication Web Hooks
  + While most managed k8s installation configure authentication for you, if you are deploying your own authentication yo will need to configure flags on the k8s API Server appropriately
* Understanding Roles and Role Bindings:
  + Identity is just beginning of authorization in K8s. Once the system knows the identity, it needs to determine if the request is authorized for that user. To achieve this, k8s used the general concept of role and role bindings
  + A role is set of abstract capabilities. For example ability to create Pods, Services etc
  + A role binding is an assignment of a role to one or more identities
* Roles and Role Bindings in K8s
  + In k8s there are two pairs of related resources that represent role and role binding.
  + One pair applies to just a namespace (Role and RoleBinding) and then other applies across the cluster (ClusterRole and ClusterRoleBinding





| **Verb** | **HTTP Method** | **Description** |
| --- | --- | --- |
| create | POST | Create a new resources |
| delete | DELETE | Delete an existing resource |
| get | GET | Get a resource |
| list | GET | List a collection of resources |
| patch | PATCH | Modify an existing resource via a partial change |
| update | PUT | Modify an existing resource via a complete object |
| watch | GET | Watch for streaming updates to a resource |
| proxy | GET | Connect to resource via streaming WebSocket Proxy |

* Using built-in roles: K8s has a large number of built-in cluster roles kubectl get clusterroles
  + While most of the built-in-roles are for system utilities, four are designed for generic end users
  + The cluster-admin role provides the complete access to the entire cluster
  + The admin role proved complete access to a complete namespace
  + The edit role allows an end user to modify things in a namespace
  + The view role allows for read-only access to a namespace
* Testing Authorization with can-i:
  + This tool is very useful for testing if a particular user can do particular action
* Note: Fix for inventory Service issue

FROM python:3-alpine3.13

LABEL author="khaja"

LABEL organization="qualitythought"

ARG HOME\_DIR='/inventory-service'

ADD . ${HOME\_DIR}

ENV MYSQL\_USERNAME='qtdevops'

ENV MYSQL\_PASSWORD='qtdevops'

ENV MYSQL\_SERVER='localhost'

ENV MYSQL\_DATABASE='qtinvsrv'

EXPOSE 8080

WORKDIR ${HOME\_DIR}

RUN apk add build-base

RUN apk add --update py-pip

RUN apk add py-cryptography

RUN apk add gcc musl-dev python3-dev libffi-dev libressl-dev cargo

RUN pip install cryptography

RUN pip install -r requirements.txt

ENTRYPOINT [ "python", "app.py" ]