Let's say we have a total of 2 GPUs and 2 research teams, and we would like to assign 1 GPU to each team.

1. Lets create a namespace for the first team - lets call it team-a

   `kubectl create namespace team-a`

2. We will now create a user (service account), a role, and attach that role to that user. Open a file and call it `sa-role-rolebinding-team-a.yml`

```yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
 name: team-a-user
 namespace: team-a

---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
 name: team-a-user-full-access
 namespace: team-a
rules:
- apiGroups: ["", "extensions", "apps"]
 resources: ["*"]
 verbs: ["*"]
- apiGroups: ["batch"]
 resources:
 - jobs
 - cronjobs
 verbs: ["*"]

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
 name: team-a-user-view
 namespace: team-a
subjects:
- kind: ServiceAccount
 name: team-a-user
```

```
 namespace: team-a
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: Role
 name: team-a-user-full-access
```

As you can see, in the `Role` definition, we add full access to everything in that namespace, including batch types like jobs or cronjobs. As it is a `Role`, and not a `ClusterRole`, it is going to be applied to a single namespace: `team-a`. For more details about roles in Kubernetes, check out the official documentation.

Now, let's create all of this:

```
kubectl create -f sa-role-rolebinding-team-a.yml
```

You should see the three components being created.

3. We now need to get the name of the service account's secret. To do this, run the following command and copy the name of the secret

   ```
   kubectl describe sa team-a-user -n team-a
   ```

   ```
   ~ kubectl describe sa team-a-user -n team-a
   Name:                team-a-user
   Namespace:           team-a
   Labels:              <none>
   Annotations:         <none>
   Image pull secrets:  <none>
   Mountable secrets:   team-a-user-token-wd62n
   Tokens:              team-a-user-token-wd62n
   Events:              <none>
   ```

   I can see the secret name is `team-a-user-token-wd62n`

   We now need to get the service account's Token and the Certificate Authority. For this, we are going to read them using kubectl. Now, as Kubernetes secrets are base64 encoded, we'll also need to decode them.

   To get the User Token:

   ```
   kubectl get secret team-a-user-token-xxxxx -n team-a -o "jsonpath={.data.token}" | base64 -D
   ```

To get the Certificate:

```
kubectl get secret team-a-user-token-xxxxx -n team-a -o
"jsonpath={.data['ca\.crt']}"
```

4. We can now create a new Kube config file with the data that we got from the previous steps:

```
apiVersion: v1
kind: Config
preferences: {}

# Define the cluster
clusters:
- cluster:
   certificate-authority-data: PLACE CERTIFICATE HERE
   server: https://YOUR_KUBERNETES_API_ENDPOINT
 name: kubernetes

# Define the user
users:
- name: team-a-user
 user:
   as-user-extra: {}
   client-key-data: PLACE CERTIFICATE HERE
   token: PLACE TOKEN HERE

# Define the context: linking a user to a cluster
contexts:
- context:
   cluster: kubernetes
   namespace: team-a
   user: team-a-user
 name: kubernetes

# Define the current context
current-context: kubernetes
```

You can now share this config file with the researchers of team-a, and they will be restricted to access the team-a namespace only.