# Capacitación Kubernetes

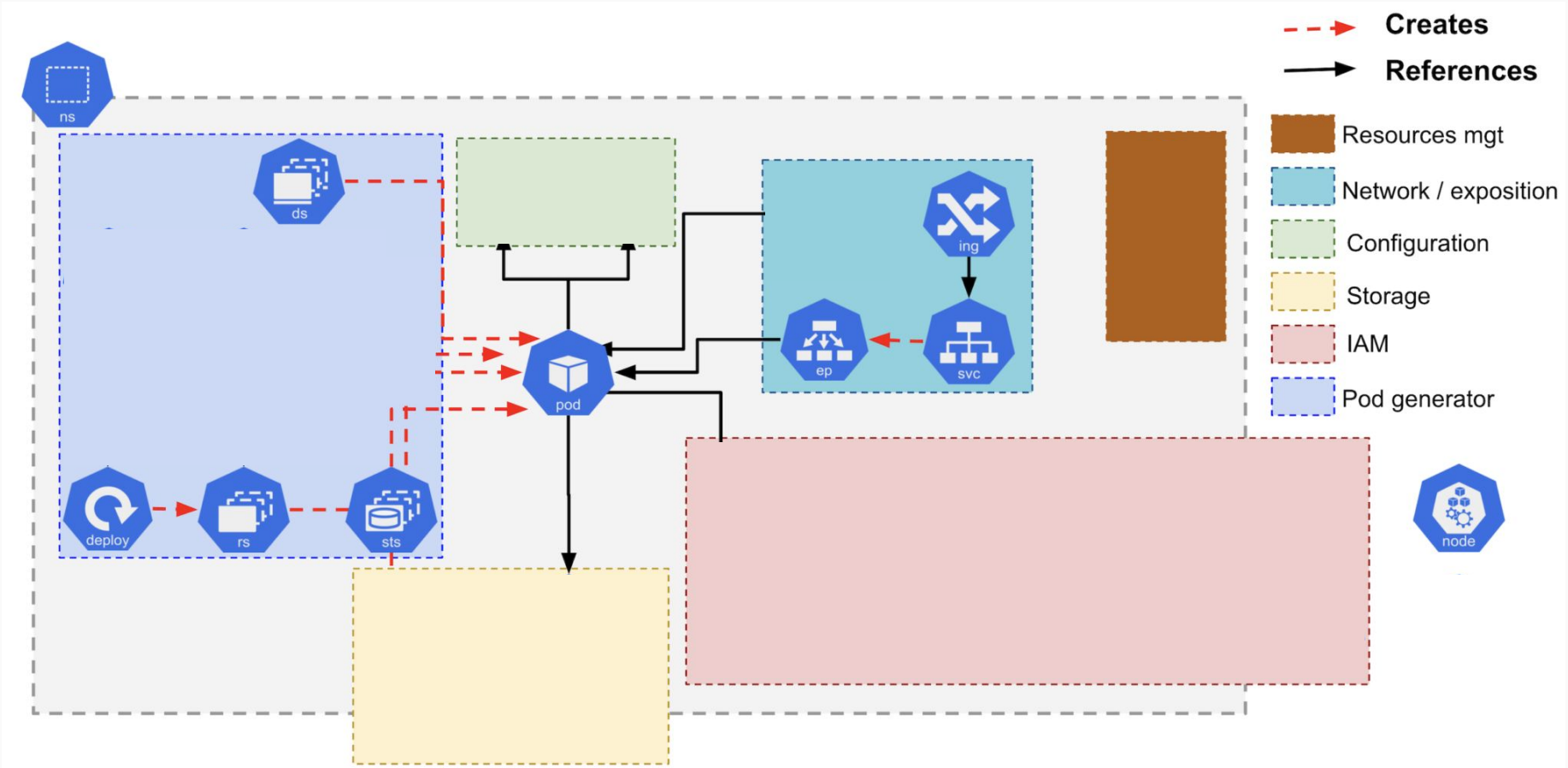
Servicio de capacitación y entrenamiento ARIU


# Clase 4
# Objetos

# Agenda

- Objetos de configuración
  - **Secrets**
  - ConfigMaps
  - Job, Cronjob
- Limitar uso de Recursos
  - Request, Limits
  - LimitRange, ResourceQuota
- Usuario y Control de acceso

# Grupos de Objetos - Resumen



## ¿Qué es un Secret?

- Un Secret es un objeto de Kubernetes utilizado para almacenar información sensible, como contraseñas, tokens, claves de API, certificados y otros datos confidenciales.
- Los contenedores dentro de los Pods pueden acceder a los Secrets como variables de entorno o archivos montados en el sistema de archivos.
- Hay distintos tipos de Secret, el más usado es el Opaque. Almacena datos codificados en base64, como contraseñas y tokens.

## Buenas Prácticas

- No almacenar Secrets directamente en archivos YAML o código fuente.
- Utilizar herramientas de gestión de Secrets

Se pueden crear desde un archivo o un comando

```
kubectl create secret generic db-user-pass
--from-file=./username.txt --from-file=./password.txt
```

## Ejemplos de creación de Secret

```
kubectl create secret generic db-user-pass \
--from-literal=username=admin \
--from-literal=password='S!B\*d$zDsb=
```

```
echo -n 'admin' | base64
YWRtaW4=
echo -n '1f2d1e2e67df' | base64
MWYyZDFlMmU2N2Rm
```

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

- Borrar cosas anteriores
  - k delete deployment nginx
  - k delete statefulset.apps/postgresql-db
  - k delete ds node-exporter
- Secrets
- Crear secret desde archivo y literal
  - k create secret generic postgres.env --from-file=postgres.env --from-literal=POSTGRES_PASSWORD=postgres -n prueba
  - kdescribe secret postgres.env
- Crear secret desde manifiesto
- Codificar user y pass en base64
  - echo -n 'admin' | base64
  - echo -n 'password' | base64

- Aplicar y mostrar en formato yaml
  - k apply -f 01-secret-postgres.yml
  - k get secret postgres2.env -o yaml
- Decodificar password
  - k get secret postgres2.env -o jsonpath="{.data.password}" | base64 -d
- Aplicar deploy de postgres referenciando secrets
  - k apply -f 02-postgres-deploy.yml
- Verificar que se cargaron los secret como ENV dentro del pod
  - k get pod
  - k exec -it postgres-<ID> -- env | grep PGTZ
  - k exec -it postgres-<ID> -- env | grep admin

# Agenda

- Objetos de configuración
  - Secrets
  - **ConfigMaps**
  - Job, Cronjob
- Limitar uso de Recursos
  - Request, Limits
  - LimitRange, ResourceQuota
- Usuario y Control de acceso

## ¿Qué es un ConfigMap?

- ConfigMap es utilizado para almacenar datos de configuración.
- Permite separar la configuración del contenedor del código fuente de la aplicación.
- Se puede usar
  - Reemplazar un archivo (o varios) dentro del Pod
  - Conjunto de variables de entorno

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    configMap:
      name: myconfigmap
```

- Crear configmap desde archivo
  - kubectl create configmap nginx-conf --from-file=nginx.conf
- Crear configmap desde manifiesto
  - k apply -f 03-nginx-configmap.yml
  - k describe cm nginx-html
- Aplicar deploy
  - k apply -f 04-nginx-deploy.yml
- Verificar que los archivos se crearon correctamente
  - k exec -it nginx-58d494d94b-8zbgz -- ls -l /var/www

- Verificar IP de pod y hacer curl
  - k get pod -o wide
  - > curl <IP>
  - > curl <IP>/index.html
- Editar index2 de manifiesto de configmap
  - nano 03-configmap-nginx.yml
- Aplicar y verificar
  - k apply -f 03-configmap-nginx.yml
  - > curl <IP>
- Delete pod para que tome nuevo configmap
  - k get pod
  - k delete pod nginx-<ID>
  - > curl <IP>

# Agenda

- Objetos de configuración
  - Secrets
  - ConfigMaps
  - **Job, Cronjob**
- Limitar uso de Recursos
  - Request, Limits
  - LimitRange, ResourceQuota
- Usuario y Control de acceso

## Job y CronJob

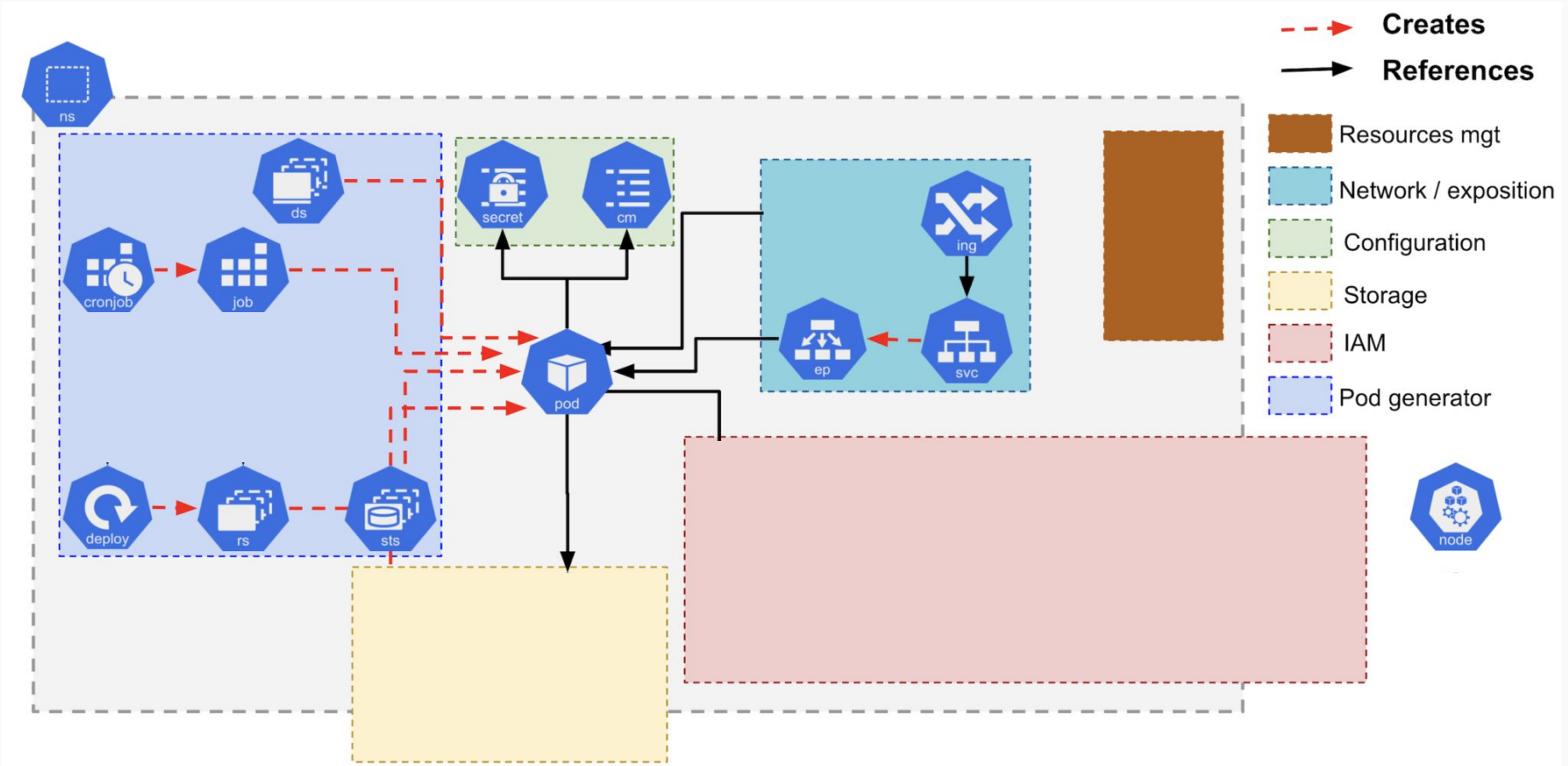
- Job y CronJob son objetos de Kubernetes para administrar tareas y trabajos por lotes.
- Permiten ejecutar tareas una sola vez o de forma periódica según un cronograma.
- Si la tarea falla, se realizarán nuevos intentos según el límite de reintento
- CronJob ejecuta trabajos basados en un cronograma predefinido (schedule).
- Un Job en Kubernetes utiliza una plantilla de Pod (PodTemplate) para crear uno o más Pods que llevarán a cabo las tareas específicas

## Job y Cronjob

- Crear JOB que muestra fecha de ejecución
  - k apply 05-job-date.yml
- Mostrar Jobs corriendo
  - k get pods
- Mostrar logs del pod
  - k logs date-<id>
- Mostrar que el Job se completó
  - k get jobs
  - k get pods
- Eliminar Job
  - k delete job date-job

- Crear CronJob
  - k apply -f 06-cronjob.yml
- Mostrar CronJob
  - k get cronjob
- Mostrar Job y Pod
  - k get job
  - k get pod
- Mostrar logs de Pod
  - k logs date-job-<id>
- Esperar un par de minutos y mostar pods ejecutados
  - k get pod

# Grupos de Objetos - Resumen



# Agenda

- Objetos de configuración
  - Secrets
  - ConfigMaps
  - Job, Cronjob
- **Limitar uso de Recursos**
  - Request, Limits
  - LimitRange, ResourceQuota
- Usuario y Control de acceso (RBAC)

## Introducción

- Los recursos en Kubernetes son críticos para el rendimiento y la estabilidad de las aplicaciones en contenedores.
- Kubernetes ofrece varias formas de controlar y gestionar los recursos en los Pods.
- No es obligatorio declarar ningún límite ni solicitud, pero es una buena práctica

➢ Request
➢ Limit
➢ Limit Range
➢ Resource Quote

## Requests y Limits

- Al especificar un Pod, se pueden indicar las necesidades de recursos para cada contenedor
  - CPU
  - Ram
  - Almacenamiento efímero
  - Huge-page
  - Recurso extendidos
- También se pueden especificar límites para cada contenedor de los mismos recursos.

**Requests y Limits**

- Los requests definidos en el Pod lo usa el scheduler para decidir en que nodo programarlo
- Los Limits los usar el Kubelet de cada nodo para controlar la cantidad de recursos utilizados por un Pod

Es importante especificar solicitudes y límites adecuados para optimizar los recursos y evitar problemas de rendimiento.

# Límites de recursos - Requests and Limits

## Ejemplos:

```yaml
resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

```yaml
resources:
  requests:
    ephemeral-storage: "2Gi"
  limits:
    ephemeral-storage: "4Gi"
```

```yaml
requests:
  cpu: 2
  example.com/foo: 1
limits:
  example.com/foo: 1
```

- Eliminar recursos anteriores
  - k delete deployment.apps/postgres deployment.apps/nginx cronjob.batch/date-job service/nginx-html-lb
- Aplicar deploy de nginx con Request and Limits
  - k apply -f 07-req.yml
- Describir pod para ver Request and Limits
  - k get pods
  - k describe pod nginx-<ID>
- Editar manifiesto y aumentar Request y Limits a `cpu: "5"`
  - nano 07-req.yml
  - k apply -f 07-req.yml
- Describir pod para ver falla
  - k get pods
  - k describe pod nginx-<ID>
- Disminuir Request a `cpu: "250m"` y Limits a `cpu: "500m"`
  - nano 07-req.yml
  - k apply -f 07-req.yml

- Ejecutar comando `yes` para estresar CPU
  - k get pods -o wide
  - k exec -it nginx-<ID> -- yes > /dev/null
- Entrar a nodo donde está ejecutándose el pod y ejecutar `top`
  - top
- Aumentar Limits a `cpu: "1000m"`
  - nano 07-req.yml
  - k apply -f 07-req.yml
- Ejecutar comando `yes` para estresar CPU
  - k get pods -o wide
  - k exec -it nginx-<ID> -- yes > /dev/null
- Entrar a nodo donde está ejecutándose el pod y ejecutar `top`
  - top

## Limit Range

- LimitRange es un objeto en Kubernetes que define límites y valores por defecto para recursos de CPU y memoria en un namespace.
- Proporciona control a nivel de namespace.
- Los valores de Limits y Requests en la definición del Pod prevalecen sobre los definidos en LimitRange para ese Pod específico.
- Si un Pod especifica valores de límite (limit) y petición (request) que exceden los límites establecidos por el LimitRange (rango de límites) correspondiente, entonces el Pod **no** se creará y se producirá un error durante el proceso de programación.

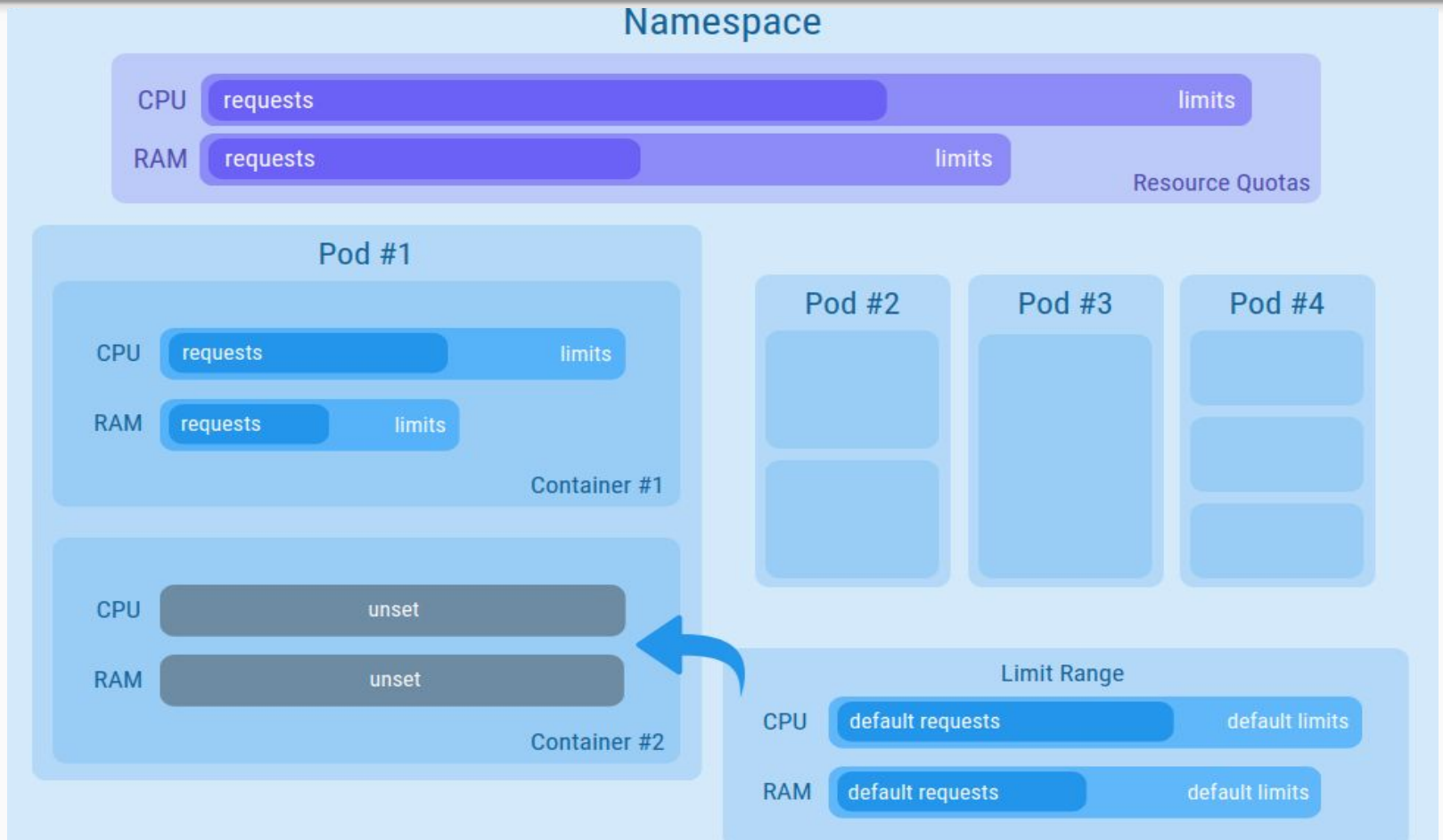
## ResourceQuota

- Limitar los recursos consumidos por los recursos de Kubernetes en un namespace

## Casos de uso

- Utilizar ResourceQuota para evitar que un espacio de nombres consuma todos los recursos del clúster.
- Limitar el uso de recursos en un espacio de nombres de prueba.
- Asignar solo la cantidad de Hardware aportado por un grupo de trabajo.

# Límites de recursos - Comparativa

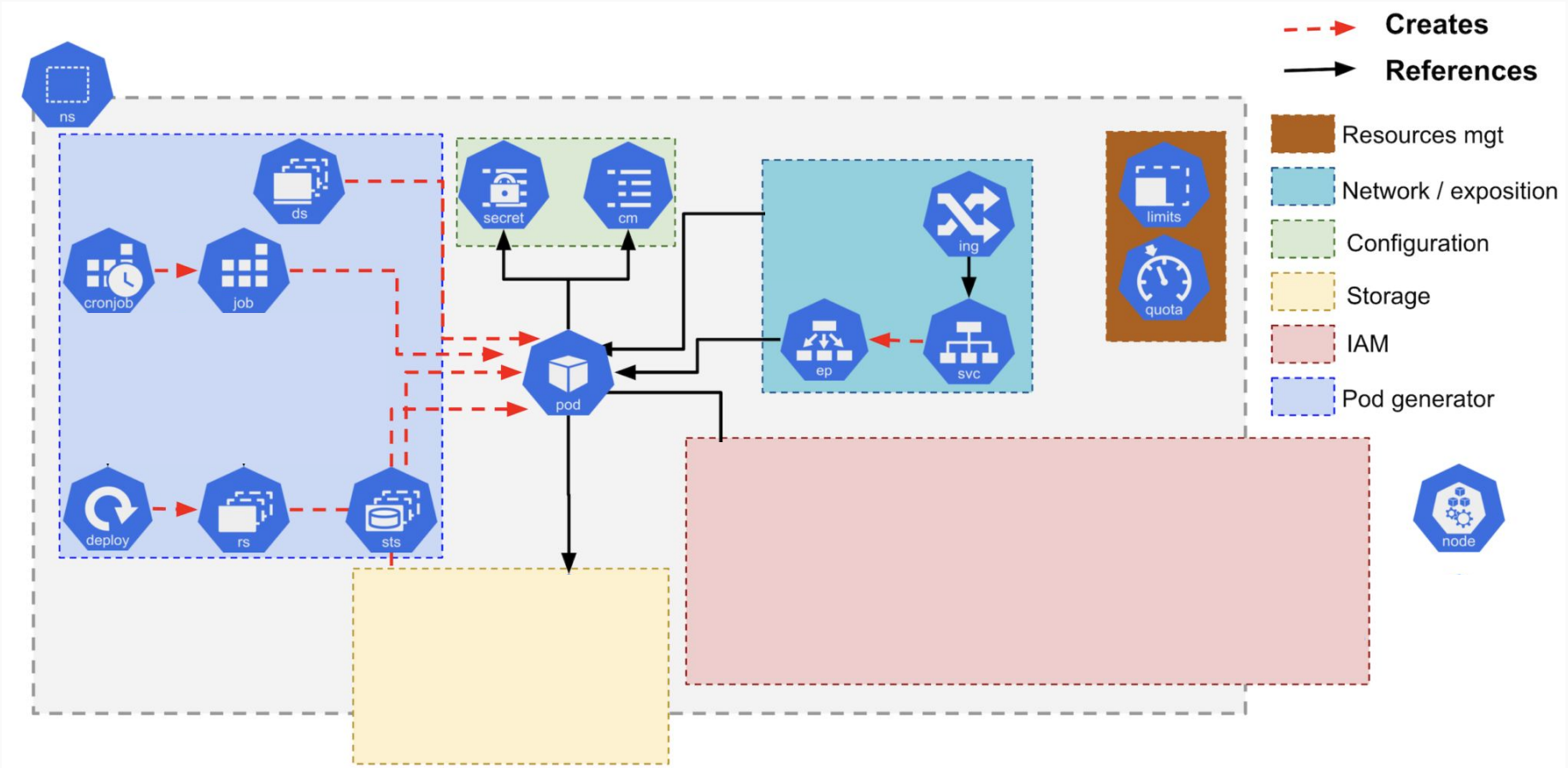


- Eliminar recursos anteriores
  - k delete deployment.apps/nginx
- Crear LimitRange
  - k apply -f 08-limitrange.yml
  - k get limitrange
  - k describe limitrange
- Aplicamos deployment nginx sin `resources`
  - k apply -f 09-nginx-lr.yml
  - k describe pod
- Agregamos `resources` con Limits en `cpu: "2"`
  - nano 09-nginx-lr.yml
- Borramos deploymen anterior y aplicamos
  - k delete deployment.apps/nginx
  - k apply -f 09-nginx-lr.yml
- Mostramos que no se levanta el pod
  - k get all
- Describimos Replicaset para mostrar restricción
  - k describe replicaset.apps/nginx-<ID>

- Eliminar recursos anteriores
  - k delete deployment.apps/nginx
- Aplicar `ResourceQuotas`
  - k apply -f 10-resourcequota.yml
  - k apply -f 11-resourcequota-objets.yml
  - k get resourcequota
- Aplicar deployment nginx
  - k apply -f 12-nginx-rq.yml
  - k get all
- Describir ReplicaSet para ver fallo
  - k describe replicaset
- Editamos deployment para agregar Resource de Memoria
  - nano 12-nginx-rq.yml
  - k apply -f 12-nginx-rq.yml
  - k get pod
- Verificamos uso de quota
  - k get resourcequota

- Editamos deployment para setear 7 réplicas
  - nano 12-nginx-rq.yml
  - k apply -f 11-nginx-rq.yml
- Mostrar Falla por máximo de pods=3
  - k get resourcequota
  - k get all
  - k describe replicaset.apps/nginx-<ID>
- Eliminar Restricción de cantidad de pods
  - k delete resourcequota object-counts
- Eliminar un pod para que `ReplicaSet` intente recrear la cantidad de pods deseada
  - k get pod
  - k delete pod nginx-<ID>
- Verificar cantidad de pods que levantaron, y mostrar fallos
  - k get all
  - k get resourcequota
  - k describe replicaset.apps/nginx-<ID>

# Grupos de Objetos - Resumen



# Agenda

- Objetos de configuración
  - Secrets
  - ConfigMaps
  - Job, Cronjob
- Limitar uso de Recursos
  - Request, Limits
  - LimitRange, ResourceQuota
- **Usuario y Control de acceso**

**Introducción**

- Los usuarios o aplicaciones se autentican mediante credenciales.
- Es posible restringir el acceso de distintas características en namespaces o en todo el cluster.
- Evitar acceso de administrador para todos. Definir roles específicos según responsabilidades.

**Ejemplos útiles**

- Otorgar permisos específicos a namespaces de desarrollo y producción. Usando un solo cluster para ambas funciones
- Ejemplo: Responsable de Guarani en UNICEN; Responsable de aplicación en SIU

# Usuario y control de acceso - Definición

**APIGroups**

- Los recursos en Kubernetes están organizados en grupos de recursos (APIGroups)
- Un APIGroup es una colección lógica de recursos relacionados que están disponibles a través de la API del clúster.
- Ejemplo: apps, extensions, batch, networking.k8s.io, etc.

**Resources**

- Los recursos son los objetos específicos que se pueden crear, leer, actualizar o eliminar dentro del clúster de Kubernetes.
- Ejemplo: pods, deployments, services, configmaps, jobs, etc.

**Verbs**

- Los verbos representan las acciones que se pueden realizar en los recursos.
- Ejemplo de verbos comunes: get, list, watch, create, update, delete, entre otros.

**Rol**

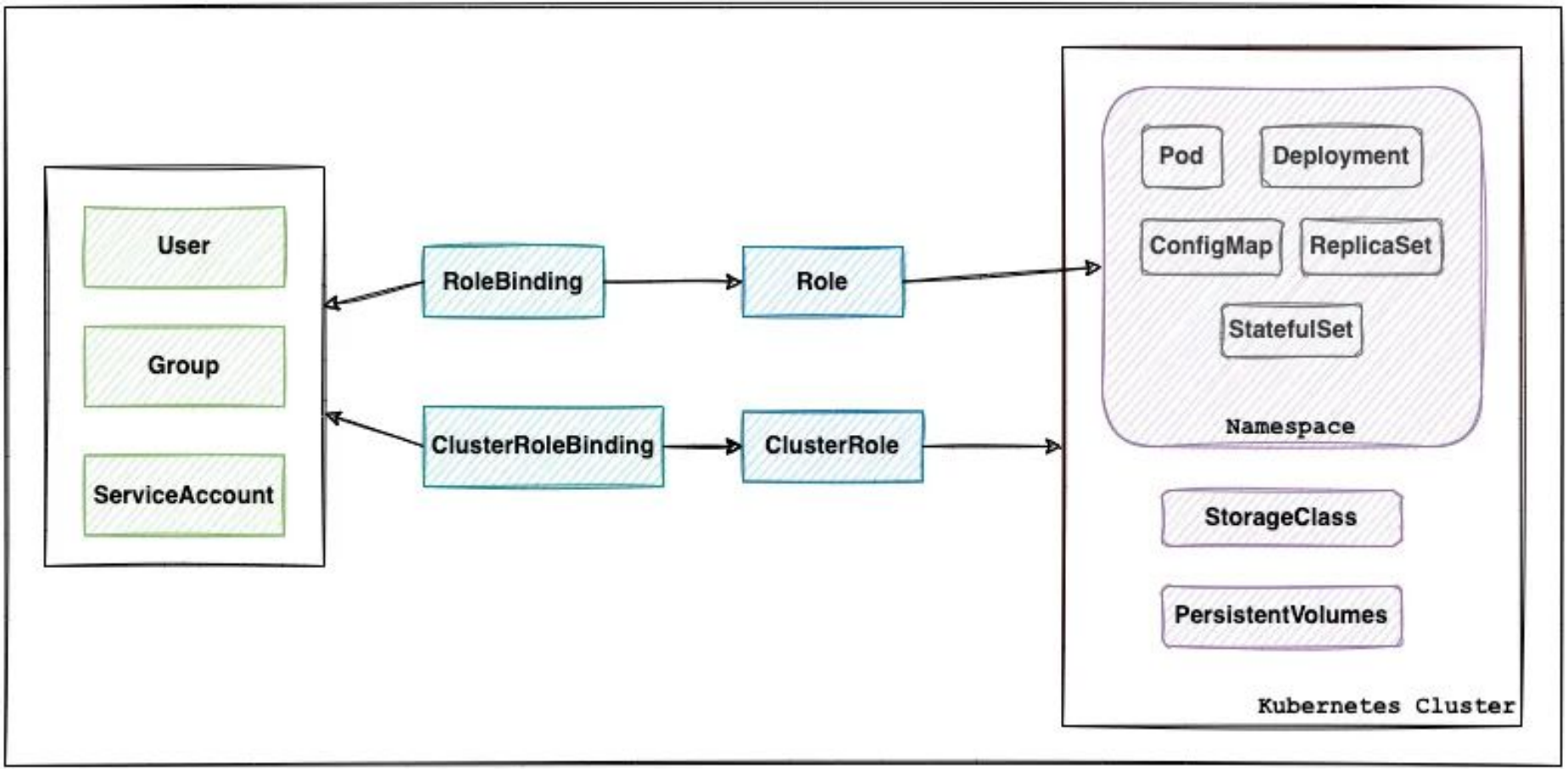
- Solo es válido dentro del espacio de nombres específico.
- Adecuado para otorgar permisos a recursos específicos como Pods o ConfigMaps.
- Para asignar un Role a un usuario o grupo, se necesita un **RoleBinding**.

**Definir un Rol**

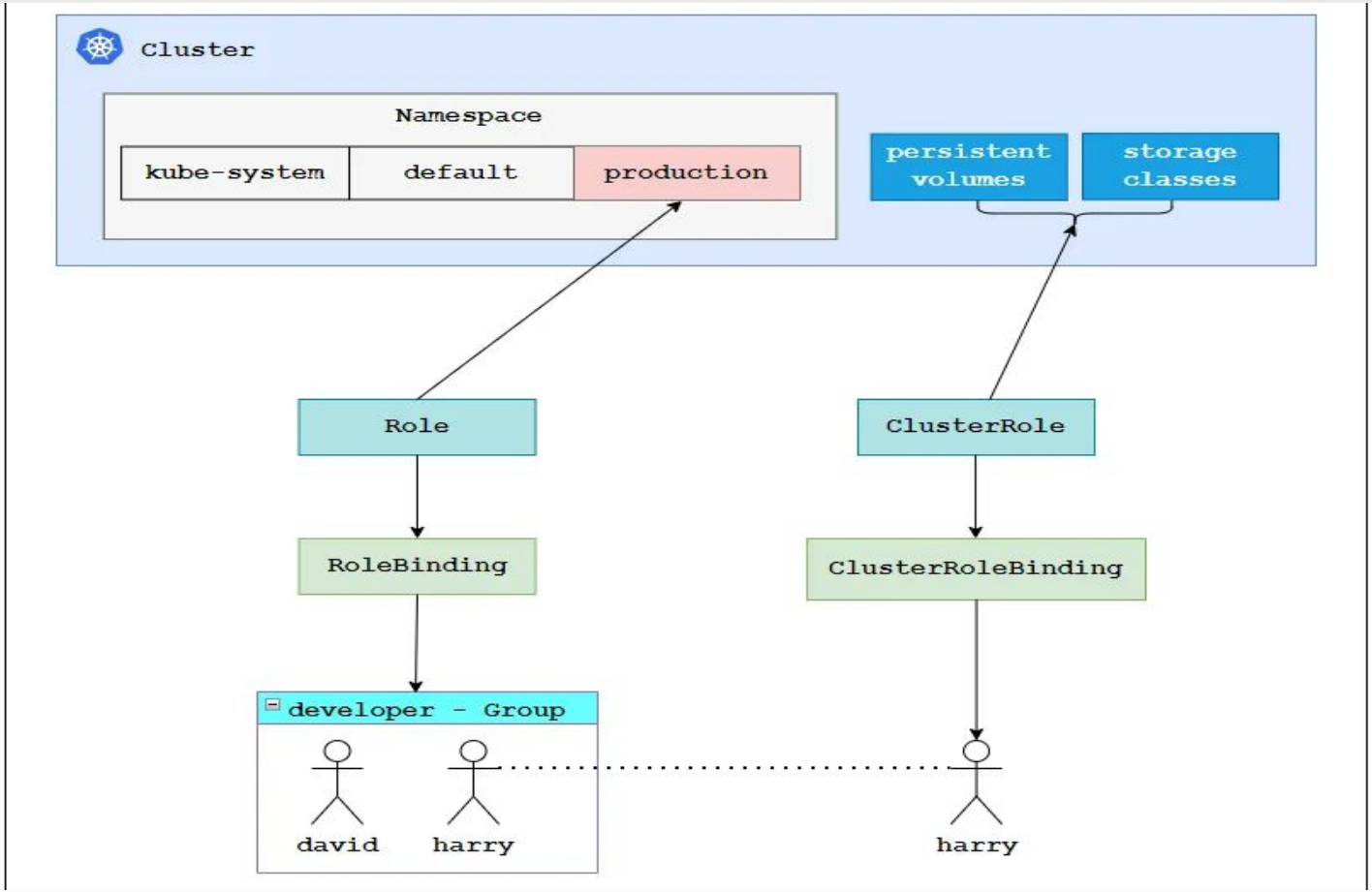
- Especificar APIGroup, Recursos y acciones que deseas controlar.

Si se quiere controlar recursos a nivel de cluster se debe crear un **ClusterRole** y usar un **ClusterRoleBinding** para asociarlo

# Usuario y control de acceso - Roles



# Usuario y control de acceso - Roles



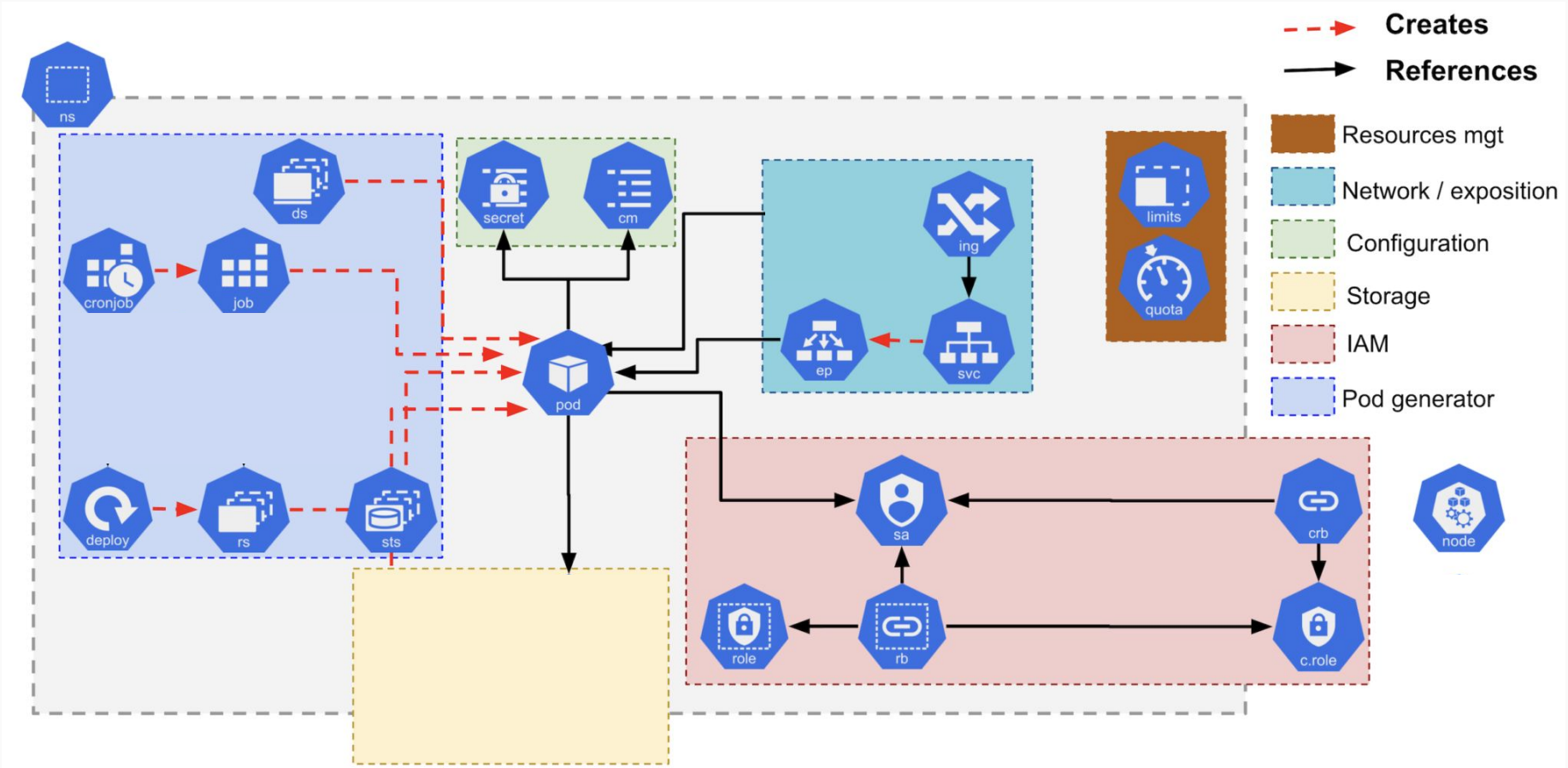
1. Se reciben tres parámetros de entrada en el script: USER, NAMESPACE y CLUSTER (nombre a darle para la configuración)
2. Crea un nuevo namespace con el nombre proporcionado.
3. Usa los archivos "admin-role.yaml" y "admin-rolebinding.yaml" para crear los Roles y RoleBindings necesarios para otorgar permisos de administrador en el namespace recién creado.
4. Genera un nuevo par de claves (pública y privada) para el usuario (USER).
5. El certificado del usuario se crea con el comando "kubectl certificate approve" y se almacena en archivos con extensión ".key" y ".crt".
6. Luego, el script muestra comandos para configurar la conexión en el cliente del usuario.

## Usuario y control de acceso - Script de creación

- Revisar script
  - cd certificados-usuarios
  - nano user-config.sh
- Crear certificados de usuario
  - sh user-config.sh user1 ns-user1 cluster-curso
- Mostrar certificados creados
  - ls -l
- Configurar acceso a usuario nuevo
  - kubectl config set-cluster cluster-curso --server=https://ed-k8s.siu.edu.ar:6443 --certificate-authority=cluster-curso.crt --embed-certs=true
  - kubectl config set-credentials user1 --client-key=user1.key --client-certificate=user1.crt --embed-certs=true
  - kubectl config set-context user1@cluster-curso --cluster=cluster-curso --user=user1
  - kubectl config use-context user1@cluster-curso

- Mostrar contexto
  - k config get-contexts
  - cat ~/.kube/clusters/config.ep
- Eliminar acceso de usuario a cluster
  - kubectl config delete-cluster cluster-curso
  - kubectl config delete-user user1
  - kubectl config delete-context user1@cluster-curso

# Grupos de Objetos - Resumen



# Trabajo entre clases

Preparación para la siguiente clase

- Reforzar los contenidos vistos en la clase con práctica en el cluster
- Realizar Autoevaluación de los conceptos vistos en clase

A R I U

Asociación Redes de Interconexión Universitaria