Problems will occur in any system, and Kubernetes provides some great tools to help locate and fix problems when they occur within a cluster. In this lesson, we will go through the process of debugging an issue in Kubernetes. We will use our knowledge of `kubectl get` and `kubectl describe` to locate a broken pod, and then explore various ways of editing Kubernetes objects to fix issues.

Relevant Documentation

- https://kubernetes.io/docs/tasks/debug-application-cluster/debug-application/
- https://kubernetes.io/docs/tasks/debug-application-cluster/debug-pod-replication-controller/
- https://kubernetes.io/docs/tasks/debug-application-cluster/debug-service/

Lesson Reference

I prepared my cluster before the video by creating a broken pod in the `nginx-ns` namespace:

```
kubectl create namespace nginx-ns
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: nginx-ns
spec:
  containers:
  - name: nginx
    image: nginx:1.158
```

Exploring the cluster to locate the problem

```
kubectl get pods

kubectl get namespace

kubectl get pods --all-namespaces

kubectl describe pod nginx -n nginx-ns
```

Fixing the broken image name

Edit the pod:

```
kubectl edit pod nginx -n nginx-ns
```

Change the container image to `nginx:1.15.8`.

Exporting a descriptor to edit and re-create the pod.

Export the pod descriptor and save it to a file:

```
kubectl get pod nginx -n nginx-ns -o yaml --export > nginx-pod.yml
```

Add this liveness probe to the container spec:

```
livenessProbe:
  httpGet:
    path: /
    port: 80
```

Delete the pod and recreate it using the descriptor file. Be sure to specify the namespace:

```
kubectl delete pod nginx -n nginx-ns

kubectl apply -f nginx-pod.yml -n nginx-ns
```