*We will be doing an exercise that involves coding/testing against a local Kubernetes Cluster using the Kubernetes SDK.*

You may need to set up the following ahead of time:

1. You can use Docker for Desktop to enable installation of a local Kubernetes cluster

Example: <https://andrewlock.net/running-kubernetes-and-the-dashboard-with-docker-desktop/>

Alternatively you can use other local Kubernetes clusters, such as Kind or k3s  
Example:  
<https://kind.sigs.k8s.io/docs/user/quick-start/>

<https://k3s.io/>

1. Choose SDK in the language of your choice, and make sure you can communicate with your development cluster via the SDK

Python: <https://github.com/kubernetes-client/python>

Golang: <https://github.com/kubernetes/client-go>

Other languages are also available

\*Make sure your IDE is configured properly to work with these libraries and your chosen language

1. If you use Golang, you can explore “fake” for unit-tests, for other languages explore using some sort of mocking approaches.

<https://pkg.go.dev/k8s.io/client-go/kubernetes/fake>

We would like to design and implement a small program that finds pods in “bad” or “completed” states and forcibly delete them from the cluster, in order to avoid delaying deployments, or causing load on the Kubernetes API server.

* Write a program in your chosen language that scans all pods in a Kubernetes cluster, and finds pods in specific states - “Terminating”, “Completed”, “Failed”
* If the pod has been in this state for more than N minutes (configurable and in argument to the program), delete the pod, either by force or normally.
* Write unit tests

The program should consider the cases where we want to forcefully delete, delete normally, or avoid taking action. You should also consider how this program will run on multiple clusters and how you would monitor the results across multiple clusters.

For testing purposes you can use the following pod manifests, to reproduce a pod in Completed:

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: successful-pod**

**namespace: default**

**spec:**

**containers:**

**- name: main**

**image: busybox:1.28**

**command: ["/bin/sh"]**

**args:**

**- "-c"**

**- "exit 0"**

For testing purposes you can use the following pod manifests, to reproduce a pod in Failed:

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: failed-pod**

**namespace: default**

**spec:**

**containers:**

**- name: main**

**image: busybox:1.28**

**command: ["/bin/sh"]**

**args:**

**- "-c"**

**- "exit 1"**

For testing purposes you can use the following pod manifests, to reproduce a pod stuck in Terminating state:

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: terminating-pod**

**namespace: default**

**spec:**

**terminationGracePeriodSeconds: 50000**

**containers:**

**- name: main**

**image: busybox:1.28**

**command: ["/bin/sh"]**

**args:**

**- "-c"**

**- "sleep 99999"**