Goal 4: Write python code using the Kubernetes-client api that starts/stops the cluster, starts/stops the mysql db, and triggers execution of my program.

So as it turn out, this goal was written before I was familiar with Kubernetes, and portions of it don't make a lot of sense after knowing what I know now.

Kubernetes retains statefulness about your deployments, pods, and services (and any other objects you define) even after a cluster is completely rebooted. If a mysql DB or stock-rsi container is defined and running before a cluster is brought down, Kubernetes will do everything possible to insure it is running when the cluster is brought back. If a pod fails for any reason, it will periodically kill and recreate it in perpetuity (it does however have a configurable a "back off" algorithm that increases the time in between restarts with every failure so as to not self destruct by consuming resources in a restart loop). I was curious how well this worked, so I brought the cluster up without the NFS server for the mysql DB. Kubernetes caught that the pod creation failed and went into a retry loop. After booting the NFS server, the mysql pod came up correctly within a few minutes.

Given how well Kubernetes handles the state of an application, writing code to start/stop the mysql db doesn't make a lot of sense.
For stock-rsi, I talked a bit about how most folks trigger a batch program using a CI/CD pipeline into Kubernetes as part of goal #3.

I did write code to start/stop the cluster, mostly to reduce my expense in Azure. I have scripts to stop individual servers as well as take the whole cluster up and down: https://github.com/olivelawn/cs199-indep-k8s/tree/main/cluster-start-stop/k8s4_rg

Rewriting this code using the python kubernetes-client and/or azure api is likely possible, but might be overkill given the simplicity of the cluster start/stop scripts.