# Applied Containers & Orchestration

## Grading

- 25% part 1
- 25% part 2
- 50% part 3

## Semester Project

The semester long project will be to build a fully containerized key-value store system. The system should consist of an api, web service, and database. The api should be stateless REST api with endpoints for reading and writing key value pairs. The web service will simply be for serving website static files (html, css, js, etc.). To interact with your system, you will open your website, and manage your key value pairs.

Throughout the semester there will be several deadlines where we will be building up the functionality of this system from a very simple bare deployment to a production grade system.

This project can be an individual, or group project. If you elect for forming a group, please

### Part 1:

Build a minimal fully containerized key-value system in docker-compose. The system should consist of an api, web service, and database (simple database like sqlite is ok for this stage).

Requirements for this stage:

- Functioning api service with REST endpoints for:
    - Listing keys
    - Creating keys
    - Reading keys
    - Updating keys
    - Deleting keys
- Api uses database (sqlite ok for this stage)
- Minimal website that allows for the above endpoints to be interacted through html
- Api service is independent container
- Web service is independent container

## Part 2:

At this stage, we will add many of the components to our existing docker-compose system to make our system closer to production grade. It is at this point that your system should have full functionality from the perspective of the user.

Requirements for this stage:

- Add an edge router (ex: traefik/nginx) to handle all your incoming requests
- Place in-memory caching layer in front of web static service (ex: varnish)
- Database that the api is using should be an independent service at this point (ex: mariadb/mongo)
- Api should have request deduplication service in front of it (should address cache stampede / three stooges problem)
- Key-Value read operation endpoints should be cached (ex: redis/etcd/memcached)
- New services introduced in this step should be independent containers (api cache service, request deduplication service, edge router service, web static cache layer)

## Part 3

Part 3 is primarily taking what you have built in docker-compose, and converting that setup to a working kubernetes cluster. Main things considered for this step include locking down security, and proper packaging of your application.

Requirements for this stage:

- All deployments should exist in a single kubernetes configuration. You may not use helm to install external services like mariadb.
- All containers from the previous step should be packaged in proper form using good use of kubernetes api resource types.
- Database deployment should be backed up by persistent volume.
- Sidecar architecture should be used where appropriate.
- All traffic should be handled through the edge router (traefik/nginx).
- All pods within the system should have proper network policy applied to them.
- All pods within the system should have proper container security policy applied to them.
- All in-cluster networking should be handled through kubernetes Services. Proper use of kubernetes systems for service discovery should be used.
- Horizontal Pod Autoscalers should be applied to api deployment.
- Proper resource requests and limits should be specified on all pods throughout the system.
- Database and cache deployments can be single pod deployments.