Lab 6 – Kubernetes

# Basics:

* Kubernetes, container orchestration: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>
  + Cluster, Pods, Services etc.
  + Used to manage a cluster of hosts/containers
  + Why do we need orchestration? What functionalities are provided by k8s, which exceed those of a simple dockerd daemon?
  + Alternative implementations: OpenShift, Rancher, Nomad, commercial services: e.g. AWS ECS, Azure AKS, Google GKE
* [Helm](https://helm.sh/) – the package manager for Kubernetes

# Prerequisites

* Preferably a Linux environment (Linux VM, or Windows/WSL 2)
* Kubernetes cluster. While installing a full-fledged cluster from scratch is complex, you can use:
  + Amazon EKS: [Managed Kubernetes - Amazon Elastic Kubernetes Service (EKS) - AWS](https://aws.amazon.com/eks/)
  + Minikube: [https://minikube.sigs.k8s.io/docs/start](https://minikube.sigs.k8s.io/docs/start/) (simple, only 1-node cluster)
  + Kind: [https://kind.sigs.k8s.io](https://kind.sigs.k8s.io/) (Kubernetes in Docker, possible to emulate multiple nodes on a single machine)

## Notes

* Minikube, by default, uses its own internal Docker daemon. This daemon doesn’t know anything about images built previously. Prepare your environment by directing it to access the internal docker daemon by using the $(minikube docker-env) command and rebuild your images. This way images will be available within the k8s cluster. (<https://medium.com/bb-tutorials-and-thoughts/how-to-use-own-local-doker-images-with-minikube-2c1ed0b0968>

# Assignments

Create a Kubernetes application (7p)

* 1. Create a k8s cluster using Amazon Elastic Kubernetes Service (EKS)
     + You can also use minikube, kind, or any other Kubernetes distribution, or existing cluster.
  2. Using Helm, install an [NFS server and provisioner](https://github.com/kubernetes-sigs/nfs-ganesha-server-and-external-provisioner) in the cluster.
     + Go to charts/nfs-server-provisioner for a README.
     + Pay attention to configuration parameters, in particular, override storageClass.name which denotes the name of the StorageClass that you’ll have to use when creating Persistent Volume Claims.
  3. Create a [Persistent Volume Claim](https://kubernetes.io/docs/concepts/storage/persistent-volumes/) which will bind to a NFS Persistent Volume [provisioned dynamically](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#provisioning) by the provisioner installed in the previous step.
  4. Create a [Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) with a HTTP server (e.g., apache or nginx). The web content directory should be mounted as a volume using the PVC created in the previous step.
  5. Create a [Service](https://kubernetes.io/docs/concepts/services-networking/) associated with the Pod(s) of the HTTP server Deployment.
  6. Create a [Job](https://kubernetes.io/docs/concepts/workloads/controllers/job/) which mounts the PVC and copies a sample content through the shared NFS PV.
  7. Test the HTTP server by showing the sample web content in a browser.

# Submission

Submit the following:

* A link to a github repository with the entire source code. In the README, describe commands required to run the application.
* A report containing:
  + Short description of running the application (e.g. screenshots)
  + Architecture diagram of the created application with a description explaining the role of the components and their connections.