# **Preparation**

* You need to install Docker, Kubernetes (comes with Docker, or you can use any cloud provider, say Google, when new customers get $300 in free credits to spend on Google Cloud) and kubectl command like tool, and Helm
* We will use <https://github.com/spring-projects/spring-petclinic> as a demo application for Dockerization and running in Kubernetes

# **Main**

1. **Docker part** (2 point)
   * Clone <https://github.com/spring-projects/spring-petclinic> project and build it
   * Create a simple docker file that only starts the spring-petclinic application. You can use

FROM openjdk:11-jre

As a base image

* + As using root user is not secure, update the docker file to create and use a different user.
  + You want to control how much memory your java application consumes. Update the docker file to pass Java Args to the application. Run the container and pass -Xms -Xmx arguments.
  + It's a Sprig Boot application and there are multiple ways how you can set application properties. Let's try to create an application.properties file and mount it into a container so the Spring Boot app can read it.

For the next task, if you are using the Desktop version of Kubernetes that comes with Docker you need to set imagePullPolicy: Never so it will not try to pull it and will use the image built in the previous task. If you are using any cloud provider you need to make sure the image is accessible in your Kubernetes cluster.

1. **Kubernetes part** (2 point)
   * Create a Kubernetes deployment manifest for spring-petclinic application. It will consist of one container. Run the deployment using

kubectl apply -f your\_filename.yaml

You can see it by running kubectl get deployment or kubectl get pods. Note: we are not specifined a namespace here as using a default one.

* + It's not enough just to run a deployment let's expose it with a Service to make it available for other applications in the cluster. You can verify the app is accessible by running the following command

kubectl run -i --tty curl --rm --image=tutum/curl --restart=Never -- sh

By running it you will start a new pod using tutum/curl image (with preinstalled curl tool) and connect to this pod. Now you can run curl service\_name:port to check the service you just created. If you are using a Kubernetes Desktop you can update your Service and change the type to NodePort. Ater, you apply this change the application will become accessible on your localhost. Note: you have either specify the node port in a range of 30000 - 32767 or not specify it but inspect your Service after deployment to find what port it uses kubectl get service.

* + In the Docker section, we mounted application.properties file as a volume. Let's do the same here using a ConfigMap. Create ConfigMap and use it in your Deployment as a volume.
  + Let's play with Replicas. Increase the number of replicas in your deployment and kubectl apply -f your\_filename.yaml. See how many pods do you have kubectl get pods. Now let's kill one pod using kubectl delete pod pod\_name and run kubectl get pods again. I will see how Kubernetes will create a new pod instead of killed one.

1. **Helm part** (1 point)
   * Create a Helm chart for spring-petclinic application. The easy way to create a chart is to run

helm create spring-petclinic

The hard way is to template the manifests from the Kubernetes task section. The chart should include at least a deployment, service, and config map.

* + Deploy the application using Helm using helm install command. When it's ready you can run the same checks as in the Kubernetes task section.
  + Delete the application using Helm helm delete

# **Extra miles**

**Ingress task**.