Getting started with Minikube and Docker

Installation of Minikube per the guide:

https://minikube.sigs.k8s.io/docs/start/

Docker on its own:

Create a an image to test, with Alpine Linux and Python. This example also illustrates how additional repositories are added to Alpine, and how to install packages.

Dockerfile:

```
FROM alpine:3.14
RUN  apk --no-cache add python3
ADD repositories /etc/apk/repositories
RUN apk --no-cache add py3-numpy@community
RUN apk add --no-cache bash
CMD /bin/sleep infinity
```

The last command just makes docker run this thing forever; we can replace this later with an actual program, e.g. a Python based web service.

The repositories file contains:

```
http://dl-cdn.alpinelinux.org/alpine/v3.14/main
@community http://dl-cdn.alpinelinux.org/alpine/edge/community
```

Build the image with:

```
docker build -t myalpine .
```

Run it to test:

```
docker run myalpine
```

You can get to a command line in the container with:

```
docker exec -it 5f8704e07920 /bin/bash
```

Here, 5f8704e07920 is the ID of the container, which you can get with:

```
docker ps
```

Once you have built this, you can move it to Minikube in various ways:
https://minikube.sigs.k8s.io/docs/handbook/pushing/

Let’s take one method: building inside MiniKube. This is also useful if you don’t have Docker on your machine. You can get access with ssh:

```
minikube ssh
```

We can create a new directory there (mkdir alpine), cd into it, and edit a Dockerfile with vi:

```
vi Dockerfile
```

Go to insert mode (press i), copy/paste the Dockerfile and save (press escape, then type :wq and press enter). Do the same for the repositories file.

After that, just use the same docker build command. When it's done, the image should automatically be available in Minikube:

```
minikube image ls --format table
```

You can directly run this in a pod without creating a new deployment (for testing for instance) like this:

```
kubectl run my-container --image= docker.io/library/myalpine --image-pull-policy=Never --
restart=Never
```

Once this is verified to work, you can delete the pod.

```
kubectl get pods
kubectl delete pod name_of_the_pod
```

Now we can create a proper deployment. Normally, you'd run something like this:

```
kubectl create deployment my-alpine-deplyoment –image=docker.io/library/myalpine
```

However, this will try to pull the image and fail. We have to specify the imagePullPolicy on the image (like we did when we ran it directly). You can do this by hand, but a better way is to do it through the yaml file describing the deployment. For that, we first get the basic yaml file, output it from a dry-run of the above command:

```
kubectl create deployment my-alpine-deplyoment --image=docker.io/library/myalpine -o
yaml --dry-run=client > deployment.yaml
```

Editing the created file, you should find this near the bottom:

```
spec:
    containers:
    - image: docker.io/library/myalpine
          name: myalpine
          resources: {}
```

Add the following line below the resources: {} directive:

```
imagePullPolicy: Never
```

You can now create the deployment with:

```
kubectl apply -f deployment.yaml
```

The results should be visible with:

```
kubectl get deployments
kubectl get pods
```

You can connect to the Alpine container with:

```
kubectl exec --stdin --tty my-alpine-deplyoment-7b997b7875-rcxqg -- /bin/bash
```

where the exact name of the pod is the one returned with the ‘get pods’ command.