Cloud Computing Project 2024-2025

Adrian-George Dumitrache
342C1

Introduction

This document outlines my solution to the 2024-2025 CC project. Its purpose is to answer the following questions:

- how?
- why?
- can I see some proof?

But not the following question:
- how can I reproduce these results command by command?

For that, please read the accompanying writeup.md document.

Task 0: Setup

Our virtual machine already has some needed tools preinstalled (Docker for instance), but we still need to install kind and kubectl. To do this, I just read the init scripts for the Kubernetes labs (which do exactly what we need) and used them to make my own script.

Proof:

```
student@hw-adrian-dumitrache02:~$ kubectl version
Client Version: v1.32.2
Kustomize Version: v5.5.0
Server Version: v1.32.2
student@hw-adrian-dumitrache02:~$ docker --version
Docker version 26.1.3, build 26.1.3-0ubuntu1~20.04.1
student@hw-adrian-dumitrache02:~$ kind --version
kind version 0.27.0
student@hw-adrian-dumitrache02:~$ sudo apt update
Get:1 https://cli.github.com/packages stable InRelease [3917 B]
Hit:2 http://security.ubuntu.com/ubuntu focal-security InRelease
Get:3 http://ncit.clouds.archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Hit:4 https://baltocdn.com/helm/stable/debian all InRelease
Hit:6 http://ncit.clouds.archive.ubuntu.com/ubuntu focal-updates InRelease
Get:7 http://ncit.clouds.archive.ubuntu.com/ubuntu focal-backports InRelease [128 kB]
Hit:5 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.32/deb  InRelease
Fetched 397 kB in 1s (514 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
student@hw-adrian-dumitrache02:~$
```

Task 1: Create a Kubernetes Cluster

Now that we have everything installed, we can create the cluster we'll be working on using the `kind` command.

Proof:

```
student@hw-adrian-dumitrache02:~$ ./scripts/task1.sh
Creating cluster "kind" ...
 ✓ Ensuring node image (kindest/node:v1.32.2) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Have a nice day! 👋
Kubernetes control plane is running at https://127.0.0.1:40065
CoreDNS is running at https://127.0.0.1:40065/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
NAME                 STATUS     ROLES           AGE   VERSION
kind-control-plane   NotReady   control-plane   5s    v1.32.2
student@hw-adrian-dumitrache02:~$ kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
kind-control-plane   Ready    control-plane   56s   v1.32.2
student@hw-adrian-dumitrache02:~$
```

Task 2: Install Gitea

Creating a namespace

Gitea's namespace is backed up by the `gitea/namespace.yaml` manifest. I chose this approach so the namespace's existence can be versioned and we're not just relying on creating it when installing Gitea.

Installing Gitea

Simply following the steps given on this [page]() will add the `helm` repo and install gitea to our namespace.

Exposing Gitea

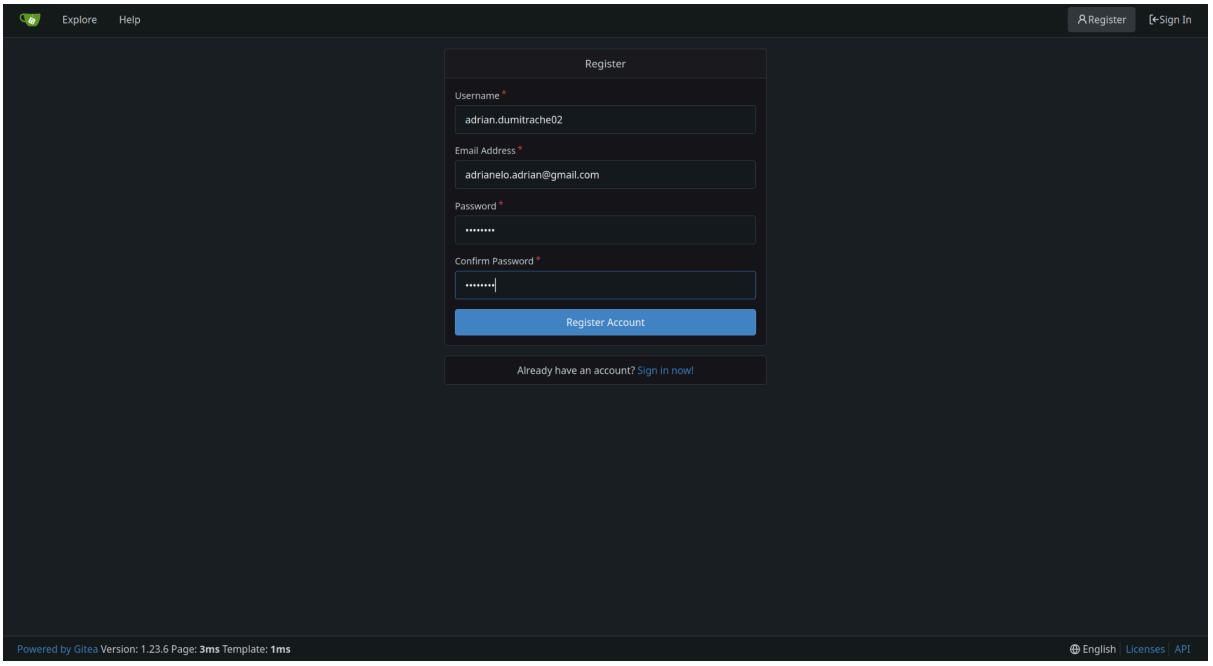
To make the service easily accessible through localhost, simply run:
`kubectl --namespace gitea port-forward svc/gitea-http 3000:3000`

Keep this running in a terminal.

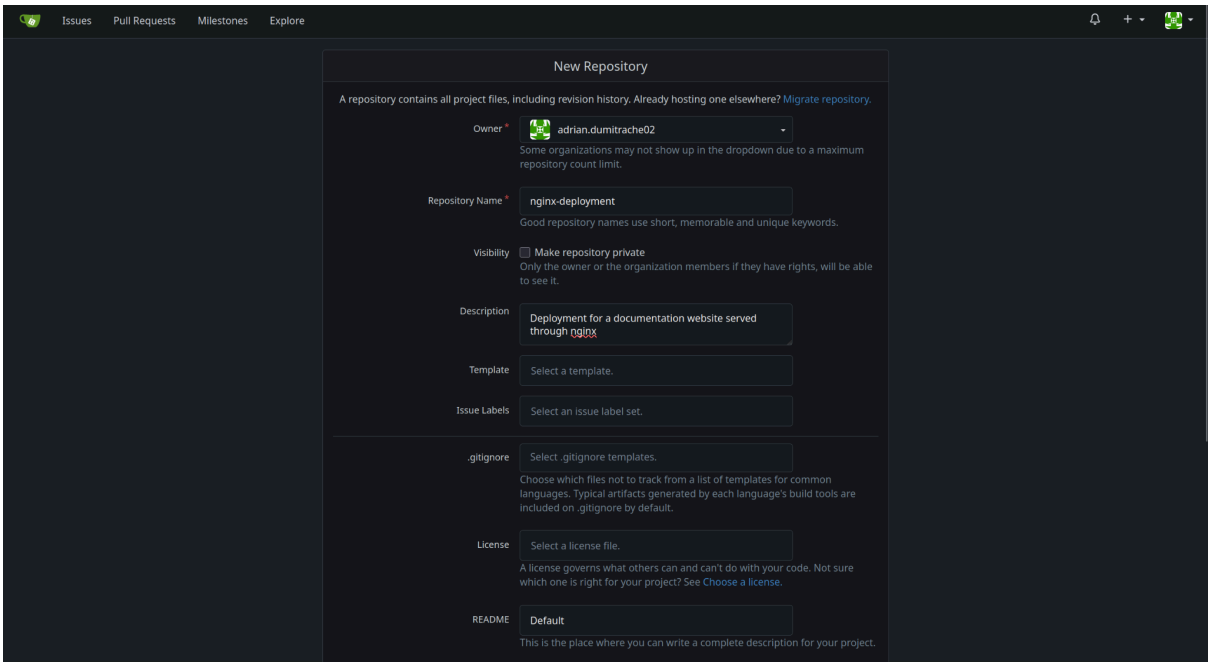
Creating the repo

We can now access Gitea's web interface.

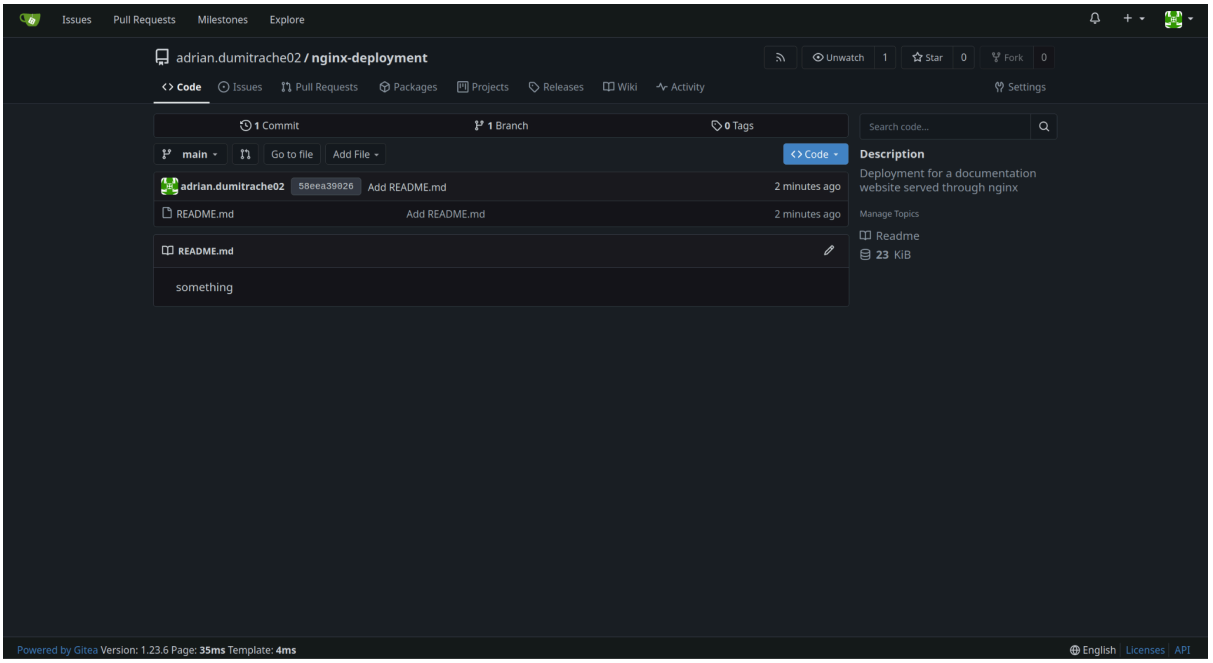
We can register a new user:



Create a new repository:



And there we go:



Task 3: Install ArgoCD

Creating a namespace

ArgoCD's namespace is backed up by the `argocd/namespace.yaml` manifest. I chose this approach so the namespace's existence can be versioned and we're not just relying on creating it when installing ArgoCD.

Installing ArgoCD and ArgoCD CLI

We can use the official docs and the CI/CD lab script to install both of them.

Exposing ArgoCD

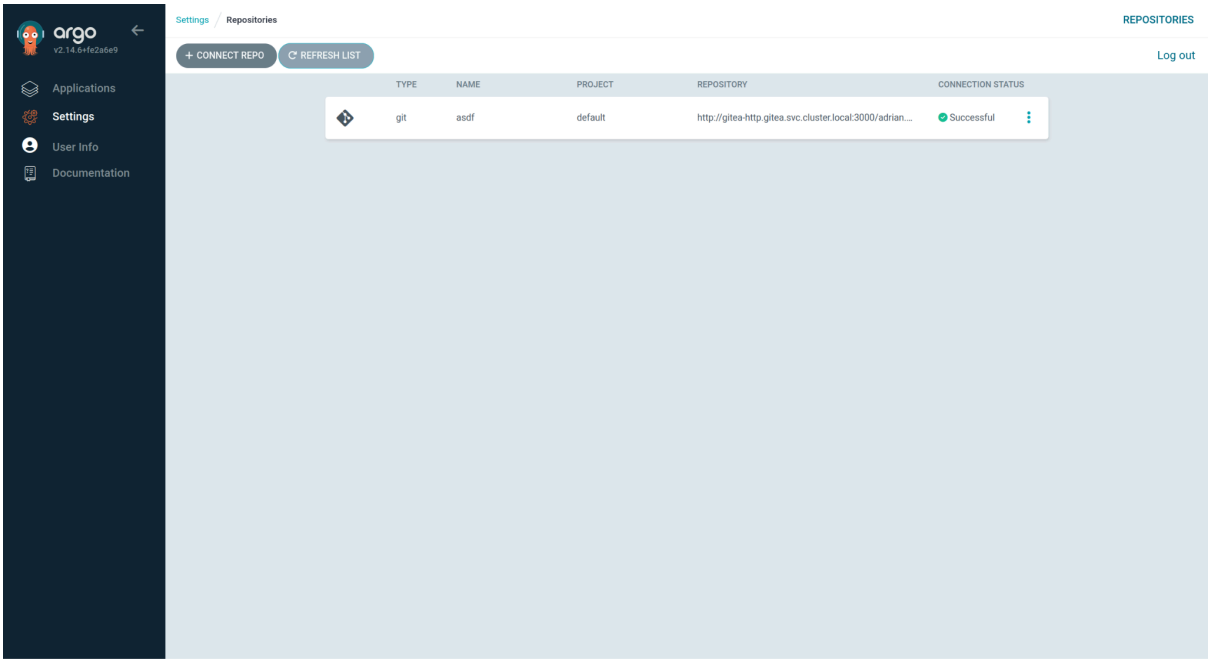
To make the service easily accessible through localhost, simply run:
`kubectl --namespace argocd port-forward svc/argocd-server 8080:8080`

Keep this running in a terminal.

Setting up the repository

While we can use the UI to do this, I preferred using a manifest to store it so it can more easily be scripted. Therefore we can use the k8s secret defined in `argocd/repository.yaml`. We can use the web UI to check that it was created. We can login using `admin` and the password returned by `argocd admin initial-password -n argocd | head -n 1`

Proof:



Gotcha: if the repository is empty you'll get an unhelpful error, add anything to the repository and it should work.

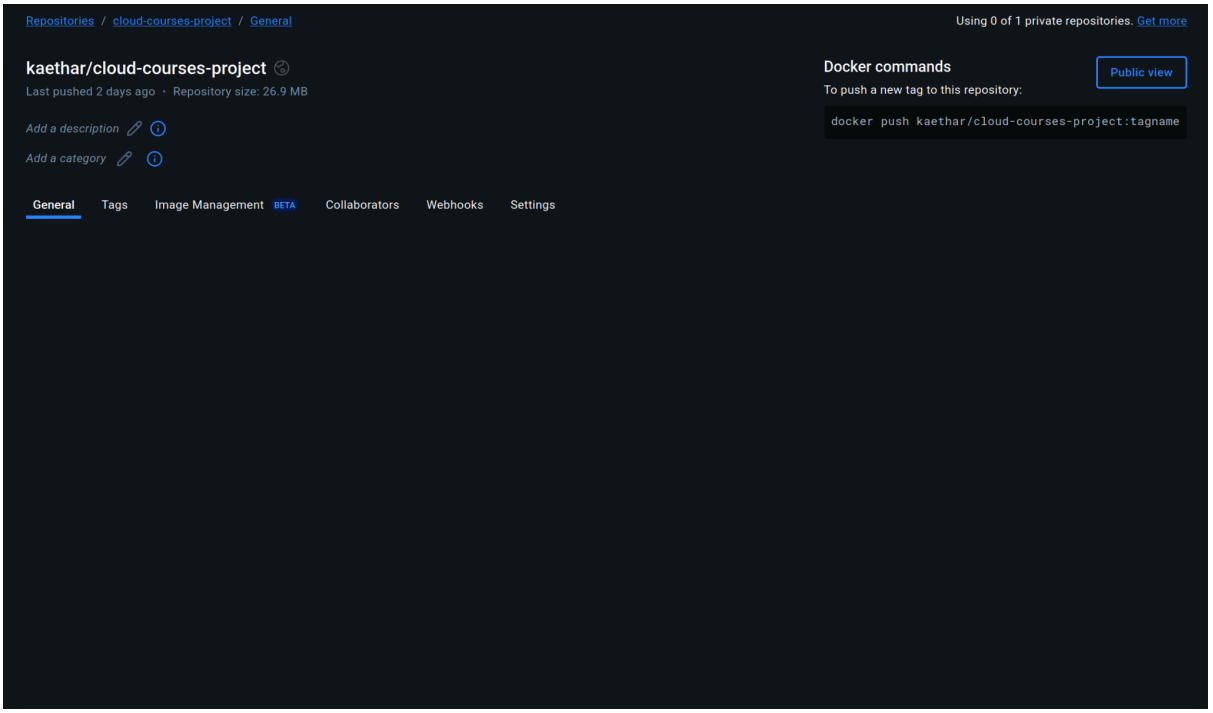
Task 4: Deploy Nginx with ArgoCD

To deploy [cloud-courses]() through Nginx I decided to modify the repository's makefile to include two stages:

1) Build: basically copy the flow from the [deploy script](), the end result are static files that can be served directly from nginx
2) Serve: based on an Nginx image, simply moves the static files from the build step to `/usr/share/nginx/html`, from where Nginx can serve the files automatically

This approach lets us have nicely packaged solution for building and deploying, all we have to do is write manifests for this Docker image!

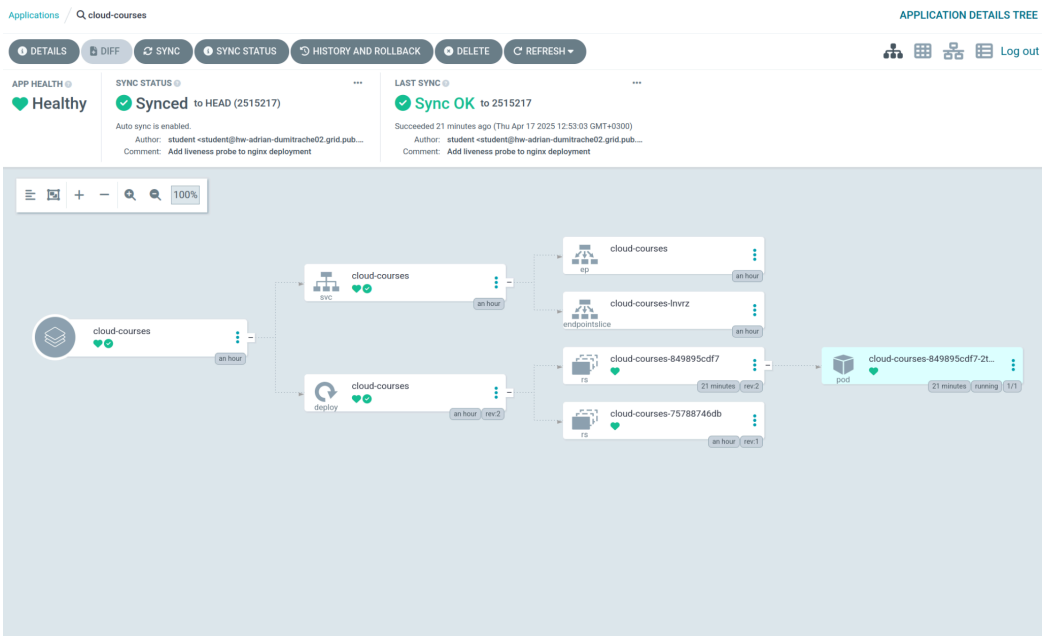
We can upload this image to a repository like Docker hub for easy access in k8s. I tried using Gitea as a repository, but it doesn't seem to work over http :/.



To deploy our app to the cluster, we have to write manifests for its deployment and service in `website/{deployment|service}.yaml`. These are fairly standard.

For automatic deployment to the cluster using ArgoCD we'll also have to write an application manifest. This can be found in `argocd/website-application.yaml`, it mostly just points to the repository, the target cluster and enables automatic syncing. We'll apply this manifest manually.

Time to test! We can simply push the entire project folder to Gitea, this will prompt ArgoCD to deploy our application to the cluster.



To further test our automatic deployments, we can make trivial changes to the manifests (e.g: change the number of replicas), push them to Gitea and check the updates in the UI and using `kubectl`.

Task 5: Add Liveness Probe to Nginx

Adding a liveness probe to `website/deployment.yaml` and pushing it to Git automatically deploys it. I used an HTTP probe on the index page for simplicity.

To check that the probe is working correctly, we can read the nginx logs and see that every couple of seconds we're getting a request to the index page from `kube-probe`.

```
10.244.0.1 - - [17/Apr/2025:10:10:13 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:10:23 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:10:33 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:10:43 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:10:53 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:11:03 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:11:13 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:11:23 +0000] "GET / HTTP/1.1" 200 32768 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:11:33 +0000] "GET / HTTP/1.1" 200 32768 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:11:43 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:11:53 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:12:03 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:12:13 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:12:23 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:12:33 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:12:43 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:12:53 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:13:03 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:13:13 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:13:23 +0000] "GET / HTTP/1.1" 200 32768 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:13:33 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:13:43 +0000] "GET / HTTP/1.1" 200 32768 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:13:53 +0000] "GET / HTTP/1.1" 200 32768 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:14:03 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:14:13 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:14:23 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
10.244.0.1 - - [17/Apr/2025:10:14:33 +0000] "GET / HTTP/1.1" 200 69980 "-" "kube-probe/1.32" "-"
student@hw-adrian-dumitrache02:~/project$
```