1.

This week, I learned different types of storage being used in Kubernetes and how to work / interact with them through Lab Manual. For instance, I learned how create additional initContainer along side with the main container that used to house the nodeJS application as well as mounting those two containers to a same volume where they can access the common file like datestamp.txt. In addition, I learned to use PV, PVC & hostPath to create a Pod. I also learned about configMap but still not yet fully undestand its functionality so I will spend more time to work on it.

These are all the important things that I learned from this week lecture.

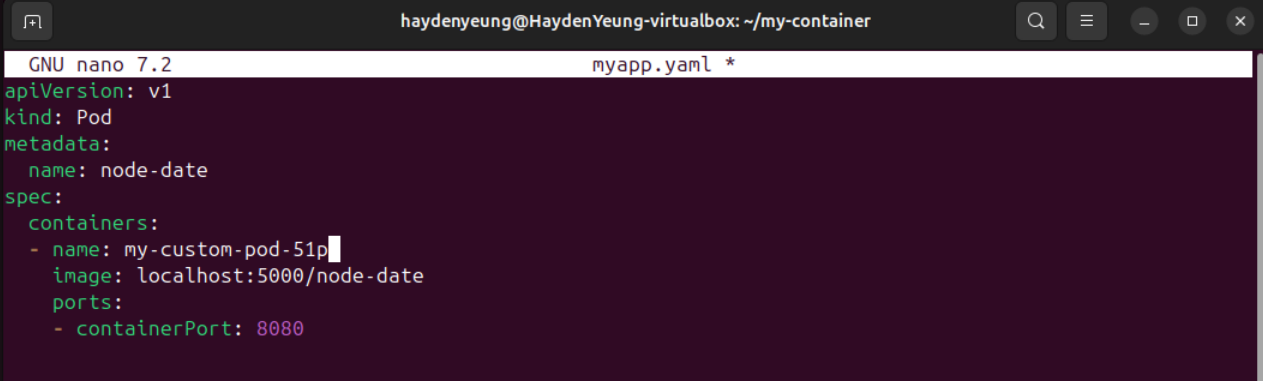
2.

Task 1 – Remembering Kubernetes

A screenshot of a computer

AI-generated content may be incorrect.

Created an image named localhost:5000/node-date and pushed it to the local repository of Docker.



Wrote ‘myapp.yaml’ to deploy a custom pod using the pushed image above.

A screen shot of a computer

AI-generated content may be incorrect.

Pod was successfully created from ‘myapp.yaml’ file

A screenshot of a computer

AI-generated content may be incorrect.

Result obtained from “Automation with an emptyDir and an init Container”- because the instruction on creating myapp.js in Lab Week 5 was ‘res.write(', I was created on '+data);’ so this is expected and different from the result shown in the Lab manual.

A computer screen shot of a program

AI-generated content may be incorrect.

Successfully created both Persistent Volume & Persistent Volume Claim based on their respective .yaml file & checked with ‘kubectl get pvc’ command.

A computer screen with white text

AI-generated content may be incorrect.

A computer screen shot of a computer

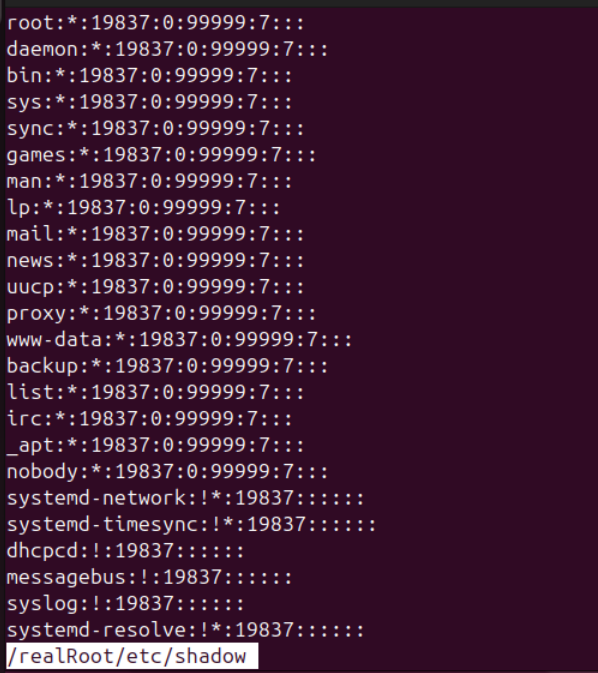
AI-generated content may be incorrect.

This was expected as datestamp.txt was not yet created in “Decoupled storage” activities. Only when this files was added that resulted in the display in the “time-of-creation” prompt appear.

A screenshot of a computer

AI-generated content may be incorrect.

The created pod in activity “The Danger of hostPath” can really access host /home folder because no “denied access” was observed



/shadow was also be able to be accessed.

A screenshot of a computer screen

AI-generated content may be incorrect.

Created Configmap was inspected by ‘kubectl get configmap ….’

A computer screen shot of a program

AI-generated content may be incorrect.

Successfully created “node-date-configmap” from its respective .yaml file

A screenshot of a computer

AI-generated content may be incorrect.

Successfully generated result with datetime value after using ‘curl localhost:8080’

Challenge Task – Change the ConfigMap

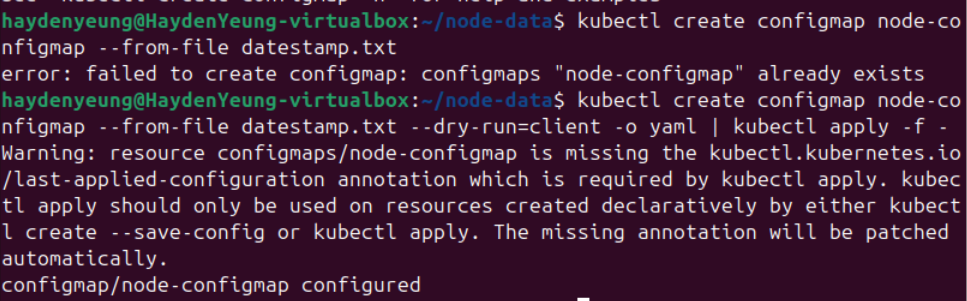
1/ Proceed with command: kubectl edit configmap <configmap-name>, which is node-configmap in this case.

2/ Change the value stored in datestamp.txt to ‘Hello Kubernetes ConfigMap Update’

A screenshot of a computer error

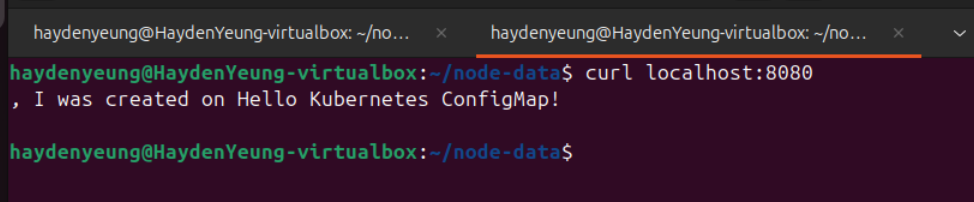
AI-generated content may be incorrect.

3/ Updated the current configmap node-configmap with new value from datestamp.txt

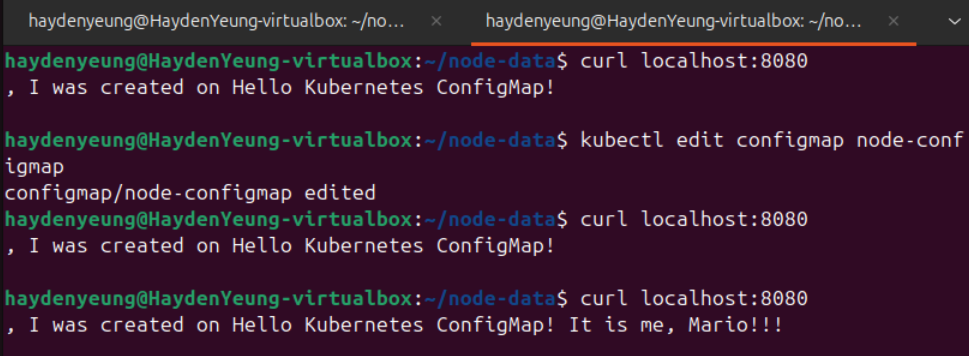


Had to used to below command, suggested from Gemini 2.5 Pro, to ‘updated’ the current node-configMap

4/ Check again with ‘curl localhost:8080’



5/ I decided to retry again: because the first time I did exit the “port-forward” and this time I did the changing value while still “port-forward” and it shown as below:



It had to took a bit of time to reflected the change.

3.

An example of a Kubernetes storage class providing temporary local storage is **emptyDir**. This storage class creates an empty volume that exists only for the lifecycle of a pod, making it ideal for ephemeral data (Kubernetes, 2025). Key features include its simplicity, as it requires no external provisioning, and its tight integration with the pod’s lifecycle—data is automatically deleted when the pod terminates. However, it lacks persistence, meaning data is lost if the pod restarts or is rescheduled. Applications like caching layers or temporary scratch space in web servers often use emptyDir, as they require fast, local storage for short-lived data during processing, such as intermediate computation results in machine learning workloads (Kubernetes, 2025).

**4.**

A Kubernetes storage class that provides block storage is **aws-ebs**, which leverages Amazon Elastic Block Store (EBS) for persistent, high-performance storage (Kubernetes, 2025). Features of aws-ebs include low-latency access, support for dynamic provisioning, and the ability to attach volumes to a single pod, ensuring dedicated storage with consistent performance. It also offers snapshot capabilities for backups and encryption for security. Block storage like aws-ebs is well-suited for applications requiring high I/O performance, such as databases (e.g., MySQL or PostgreSQL), where direct, raw access to storage is critical for handling transactional workloads and ensuring data durability (Amazon Web Services, 2025).

**References**

Amazon Web Services. (2025). *Amazon Elastic Block Store (EBS)*. <https://aws.amazon.com/ebs/>

Kubernetes. (2025). *Storage classes*. <https://kubernetes.io/docs/concepts/storage/storage-classes/>