1. Lesson Summary

* In this week, I learned how to interact with Kubernetes like how to changing the pod summary, learn the difference between “kubectl describe …” method and .yaml (extracted and put into a .yaml file) – eventhough they are 100%, just that the latter will providing more details that benefit for debuggng process while the other is best for most of the cases. In addition to lab knowledge, I also gained the information of K8s pods like: its conceptual mechanisms, lifecycle, different kinds of containers that can be present within a pod and sidecar container technique.
* These are all important for the upcoming tasks and my future career so I feel that they are all equally important.

2. Lab Activities

Task 1 – Remembering Kubernetes

A screen shot of a computer

AI-generated content may be incorrect.

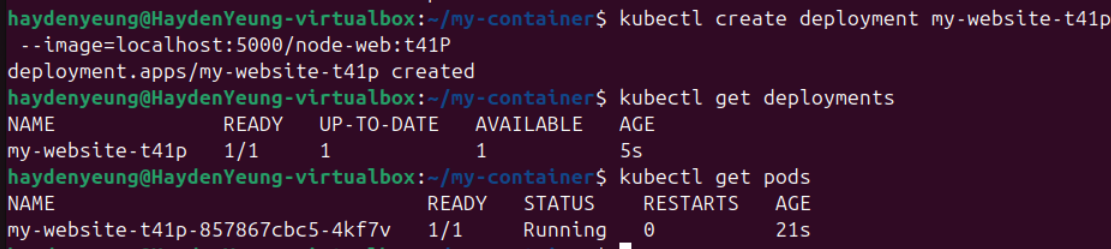
A computer screen shot of a number

AI-generated content may be incorrect.

I rebuilt “node-web” image through command ‘docker build …’, then tagged it with “localhost:5000/<image-name>:tag – for it to be able to be pushed to local repository.

A screenshot of a computer

AI-generated content may be incorrect.



Re-deploy the “my-website” application through “kubectl create deployment…” command

Task 2 – Changing the pod summary

A screenshot of a computer screen

AI-generated content may be incorrect.

At first, I thought that the value of IP, NODE, and STATUS can be accessed through “metadata.” just like the other parameters. However, I managed to “locate” the right values through the help from Grok AI.

Task 3 – What’s the difference

A/ From “kubectl describe …”

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

B/ From “custom-t41p.yaml”

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

I noticed that: inspecting .yaml file will give user more details upon pod’s specification than using describe method, such as we can see the time when Probes being used and more details in other sections that being introduced briefly in “describe” method. I would assume that “describe …” method work best in most cases while .yaml is better for debugging or for user to dig deeper to effectively manipulate it.

Task 4 – What just happened?

A computer screen shot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

I followed the intructions given from the Lab manual.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

“kubectl logs my-custom-website-t41p -c my-custom-pod-t41p”

A screenshot of a computer screen

AI-generated content may be incorrect.

“kubectl logs my-custom-website-t41p -c envoy”

A computer screen shot of a program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**How does the Envoy proxy access the NodeJS application in separate containers?**

* Envoy accesses the NodeJS app using localhost:8080 because both containers are in the same Kubernetes pod, sharing the same network namespace.

→ This means they share the same IP and can communicate directly via localhost, without needing external routing.

* The NodeJS app listens on port 8080, as defined in the YAML (containerPort: 8080).

**How does the Envoy proxy know where the NodeJS application is located?**

* The Envoy proxy, using the pre-configured image luksa/kubia-ssl-proxy:1.0, is set up to forward requests to localhost:8080.
* This configuration is embedded in the image (from *Kubernetes in Action*), so Envoy knows to target localhost:8080 where the NodeJS app is running, without requiring service discovery since both containers share the pod’s network.

**How did Kubernetes know which container to forward ports to?**

* Kubernetes uses the containerPort definitions in the pod’s YAML to map ports to specific containers.
* In the YAML, my-custom-pod-t41p declares containerPort: 8080, while envoy declares containerPort: 8443 (HTTPS) and 9901 (admin).

3. Why a Single Pod is Appropriate for the First Web Service

* For the first web service managing static content, a single Pod with two containers—one for the web server and a sidecar to download and periodically update static content (e.g., HTML files, images, JavaScript libraries) from a central master server—is appropriate.
* The web server and sidecar are tightly coupled, as the sidecar directly supports the web server by keeping content updated, sharing the same lifecycle and node. In Kubernetes, containers within a single Pod share the same network namespace, enabling efficient communication via localhost (e.g., the web server accesses updated content at localhost), and they can share storage volumes seamlessly (Kubernetes, 2023).
* This setup ensures low-latency interaction and simplifies content synchronization, as noted in discussions on sidecar patterns where such containers are ideal for auxiliary tasks like content syncing (Burns, 2018).
* Using separate Pods would be inappropriate because it introduces complexity, requiring network communication (e.g., via a Kubernetes Service) between Pods, which adds latency and overhead for a local, synchronous task.
* Additionally, separate Pods could lead to scaling issues, as the web server and content updater must remain co-located and scale together to maintain content consistency.

4. Why Two Pods are Appropriate for the Second Web Service

* For the second web service handling dynamic content (e.g., customer data, order data, catalogue data, shipping calculations), deploying two separate Pods—one for the web server and another for the database (or replica)—is optimal.
* The web server and database have distinct roles, resource needs, and scaling requirements:
  + The web server manages HTTP requests and may scale horizontally to handle traffic spikes, while the database requires consistent storage and often scales vertically or via replication for high availability.
  + Separate Pods allow independent scaling, deployment, and management, which is crucial for dynamic, database-driven services (Kubernetes Authors, 2023).
* Kubernetes best practices recommend against combining loosely coupled components in a single Pod, as it limits flexibility (Burns, 2018).
* A single Pod would be inappropriate because it forces the web server and database to share the same lifecycle and resources, risking data consistency if the web server restarts and affects the database.
* Furthermore, a single Pod restricts network and storage isolation, making it harder to secure and manage the database independently, which is critical for sensitive dynamic data.

References

Burns, B. (2018). Designing distributed systems: Patterns and paradigms for scalable, reliable services. O'Reilly Media.

Kubernetes. (2023). Pods. Kubernetes documentation. https://kubernetes.io/docs/concepts/workloads/pods/