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

```
haydenyeung@HaydenYeung-virtualbox:~$ cd ~/my-container
haydenyeung@HaydenYeung-virtualbox:~/my-container$ ls
Dockerfile  myapp.js
haydenyeung@HaydenYeung-virtualbox:~/my-container$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
bookstore                 latest    b042770dee7a   8 days ago      226MB
localhost:5000/bookstore  v1        b042770dee7a   8 days ago      226MB
registry                  2         26b2eb03618e   18 months ago   25.4MB
haydenyeung@HaydenYeung-virtualbox:~/my-container$ docker build -t node-web:t41P
 .
```

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED              SIZE
node-web                  t41P      9ea7813b1b7c   About a minute ago   936MB
bookstore                 latest    b042770dee7a   8 days ago           226MB
localhost:5000/bookstore  v1        b042770dee7a   8 days ago           226MB
registry                  2         26b2eb03618e   18 months ago        25.4MB
```

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.

```
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
localhost:5000/node-web   t41P      9ea7813b1b7c   24 minutes ago   936MB
node-web                  t41P      9ea7813b1b7c   24 minutes ago   936MB
registry                  2         26b2eb03618e   18 months ago    25.4MB
```

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl create deployment my-website-t41p
 --image=localhost:5000/node-web:t41P
deployment.apps/my-website-t41p created
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get deployments
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
my-website-t41p   1/1     1            1           5s
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
my-website-t41p-857867cbc5-4kf7v   1/1     Running   0          21s
```

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

Task 2 – Changing the pod summary

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods -o wide
NAME                               READY   STATUS    RESTARTS   AGE   IP            NODE                    NOMINATED
 NODE   READINESS GATES
my-website-t41p-857867cbc5-4kf7v   1/1     Running   0          46m   10.1.186.40   haydenyeung-virtualbox   <none>
        <none>
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods -o custom-columns=NAMESPACE:metadata.namespace,NAM
E:metadata.name,IP:metadata.id,NODE:metadata.node,UID:metadata.uid,STATUS:metadata.status


NAMESPACE   NAME                               IP       NODE     UID                                     STATUS
default     my-website-t41p-857867cbc5-4kf7v   <none>   <none>   3e031758-8ca8-4a2e-b92e-323043a49370   <none>
haydenyeung@HaydenYeung-virtualbox:~/my-container$
haydenyeung@HaydenYeung-virtualbox:~/my-container$
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods -o custom-columns=NAMESPACE:metadata.namespace,NAM
E:metadata.name,IP:.status.podIP,NODE:.spec.nodeName,UID:metadata.uid,STATUS:.status.phase
NAMESPACE   NAME                               IP            NODE                     UID
        STATUS
default     my-website-t41p-857867cbc5-4kf7v   10.1.186.40   haydenyeung-virtualbox   3e031758-8ca8-4a2e-b92e-323043a4
9370   Running
haydenyeung@HaydenYeung-virtualbox:~/my-container$
```

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 …"

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl describe pod my-custom-website-t41p
Name:             my-custom-website-t41p
Namespace:        default
Priority:         0
Service Account:  default
Node:             haydenyeung-virtualbox/10.0.2.15
Start Time:       Tue, 08 Apr 2025 14:31:30 +1000
Labels:           <none>
Annotations:      cni.projectcalico.org/containerID: 3eed3d6c614fa2c2ea3038f59b791c43ce09ea3b0c27797e5d93641e11c4aee
                  cni.projectcalico.org/podIP: 10.1.186.47/32
                  cni.projectcalico.org/podIPs: 10.1.186.47/32
Status:           Running
IP:               10.1.186.47
IPs:
  IP:  10.1.186.47
Containers:
  my-custom-pod-t41p:
    Container ID:   containerd://fabf2278cc3ffd6709160f72c8f2a706f917ae9879251f51db1172201dd1ec90
    Image:          localhost:5000/node-web:t41P
    Image ID:       localhost:5000/node-web@sha256:56faa53ed614c0aa6f227b90a24ab15bec24ff9f813ce945424d27c703fb4dd2
    Port:           8080/TCP
    Host Port:      0/TCP
```

```
    State:          Running
      Started:      Tue, 08 Apr 2025 14:31:31 +1000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-7p6hn (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-7p6hn:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
```

```
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  101s  default-scheduler  Successfully assigned default/my-custom-website-t41p to haydenyeung-virt
ualbox
  Normal  Pulled     100s  kubelet            Container image "localhost:5000/node-web:t41P" already present on machin
e
  Normal  Created    100s  kubelet            Created container: my-custom-pod-t41p
  Normal  Started    100s  kubelet            Started container my-custom-pod-t41p
```

B/ From “custom-t41p.yaml”

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cni.projectcalico.org/containerID: 3eed3d6c614fa2c2ea3038f59b791c43ce09ea3b0c27797e5d93641e11c4aee4
    cni.projectcalico.org/podIP: 10.1.186.47/32
    cni.projectcalico.org/podIPs: 10.1.186.47/32
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"name":"my-custom-website-t41p","namespace":"defaul
t"},"spec":{"containers":[{"image":"localhost:5000/node-web:t41P","name":"my-custom-pod-t41p","ports":[{"containerPort
":8080}]}]}}
  creationTimestamp: "2025-04-08T04:31:30Z"
  name: my-custom-website-t41p
  namespace: default
  resourceVersion: "237948"
  uid: 01087af8-78dc-4cfd-8784-a0263b465807
spec:
  containers:
  - image: localhost:5000/node-web:t41P
    imagePullPolicy: IfNotPresent
    name: my-custom-pod-t41p
    ports:
    - containerPort: 8080
      protocol: TCP
```

```
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-7p6hn
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: haydenyeung-virtualbox
  preemptionPolicy: PreemptLowerPriority
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
```

```
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: kube-api-access-7p6hn
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          expirationSeconds: 3607
          path: token
      - configMap:
          items:
          - key: ca.crt
            path: ca.crt
          name: kube-root-ca.crt
      - downwardAPI:
          items:
          - fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
            path: namespace
status:
  conditions:
```

```
  - lastProbeTime: null
    lastTransitionTime: "2025-04-08T04:31:31Z"
    status: "True"
    type: PodReadyToStartContainers
  - lastProbeTime: null
    lastTransitionTime: "2025-04-08T04:31:30Z"
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2025-04-08T04:31:31Z"
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2025-04-08T04:31:31Z"
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2025-04-08T04:31:30Z"
    status: "True"
    type: PodScheduled
  containerStatuses:
  - containerID: containerd://fabf2278cc3ffd6709160f72c8f2a706f917ae9879251f51db1172201dd1ec90
    image: localhost:5000/node-web:t41P
    imageID: localhost:5000/node-web@sha256:56faa53ed614c0aa6f227b90a24ab15bec24ff9f813ce945424d27c703fb4dd2
```

```
    lastState: {}
    name: my-custom-pod-t41p
    ready: true
    restartCount: 0
    started: true
    state:
      running:
        startedAt: "2025-04-08T04:31:31Z"
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-7p6hn
      readOnly: true
      recursiveReadOnly: Disabled
  hostIP: 10.0.2.15
  hostIPs:
  - ip: 10.0.2.15
  phase: Running
  podIP: 10.1.186.47
  podIPs:
  - ip: 10.1.186.47
  qosClass: BestEffort
  startTime: "2025-04-08T04:31:30Z"
```

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?

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl run --image=busybox -it --restart=Never --rm my-busybox
If you don't see a command prompt, try pressing enter.
/ # wget 10.1.186.47:8080 -O -
Connecting to 10.1.186.47:8080 (10.1.186.47:8080)
writing to stdout
Greeting, ::ffff:10.1.186.19, this is my-custom-website-t41p. How are you today?
-                    100% |*********************************************************************|    81  0:00:00 ETA
written to stdout
/ #
```

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl port-forward my-custom-website-t41p 8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
Handling connection for 8080
```

```
haydenyeung@HaydenYeung-virtualbox: ~
haydenyeung@HaydenYeung-virtualbox:~$ curl localhost:8080
Greeting, ::ffff:127.0.0.1, this is my-custom-website-t41p. How are you today?
haydenyeung@HaydenYeung-virtualbox:~$
```

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-website-t41p-857867cbc5-4kf7v    1/1     Running   0          92m
haydenyeung@HaydenYeung-virtualbox:~/my-container$ nano my-secure-website-t41p.yaml
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl  apply -f my-secure-website-t41p.yaml
pod/my-custom-website-t41p created
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
my-custom-website-t41p              0/2     ContainerCreating   0          6s
my-website-t41p-857867cbc5-4kf7v    1/1     Running             0          95m
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-custom-website-t41p              2/2     Running   0          43s
my-website-t41p-857867cbc5-4kf7v    1/1     Running   0          95m
```

I followed the intructions given from the Lab manual.

```
haydenyeung@HaydenYeung-virtualbox:~$ curl localhost:8080
Greeting, ::ffff:127.0.0.1, this is my-custom-website-t41p. How are you today?
haydenyeung@HaydenYeung-virtualbox:~$ curl https://localhost:8443 --insecure
Greeting, ::ffff:127.0.0.1, this is my-custom-website-t41p. How are you today?
haydenyeung@HaydenYeung-virtualbox:~$
```

```
haydenyeung@HaydenYeung-virtualbox:~$ kubectl logs my-custom-website-t41p
Defaulted container "my-custom-pod-t41p" out of: my-custom-pod-t41p, envoy
Server is now running on host my-custom-website-t41p port 8080..., any question?
Processing request for / from ::ffff:127.0.0.1
Processing request for / from ::ffff:127.0.0.1
```

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

```
haydenyeung@HaydenYeung-virtualbox:~$ kubectl logs my-custom-website-t41p -c my-custom-p
od-t41p
Server is now running on host my-custom-website-t41p port 8080..., any question?
Processing request for / from ::ffff:127.0.0.1
Processing request for / from ::ffff:127.0.0.1
```

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

```
haydenyeung@HaydenYeung-virtualbox:~$ kubectl logs my-custom-website-t41p -c envoy
[2025-04-08 05:09:05.700][1][info][main] [source/server/server.cc:255] initializing epo
h 0 (hot restart version=11.104)
[2025-04-08 05:09:05.701][1][info][main] [source/server/server.cc:257] statically linke
 extensions:
[2025-04-08 05:09:05.701][1][info][main] [source/server/server.cc:259]   envoy.dubbo_pr
xy.protocols: dubbo
[2025-04-08 05:09:05.701][1][info][main] [source/server/server.cc:259]   envoy.thrift_p
oxy.transports: auto, framed, header, unframed
[2025-04-08 05:09:05.701][1][info][main] [source/server/server.cc:259]   envoy.resource
monitors: envoy.resource_monitors.fixed_heap, envoy.resource_monitors.injected_resource
[2025-04-08 05:09:05.701][1][info][main] [source/server/server.cc:259]   envoy.dubbo_pr
xy.route_matchers: default
[2025-04-08 05:09:05.701][1][info][main] [source/server/server.cc:259]   envoy.dubbo_pr
xy.serializers: dubbo.hessian2
[2025-04-08 05:09:05.701][1][info][main] [source/server/server.cc:259]   envoy.transpor
_sockets.upstream: envoy.transport_sockets.alts, envoy.transport_sockets.raw_buffer, en
oy.transport_sockets.tap, envoy.transport_sockets.tls, raw_buffer, tls
```

```
[2025-04-08 05:09:05.711][1][info][main] [source/server/server.cc:459] runtime: layers:
  - name: base
    static_layer:
      {}
  - name: admin
    admin_layer:
      {}
[2025-04-08 05:09:05.711][1][info][config] [source/server/configuration_impl.cc:103] loa
ding tracing configuration
[2025-04-08 05:09:05.711][1][info][config] [source/server/configuration_impl.cc:69] load
ing 0 static secret(s)
[2025-04-08 05:09:05.711][1][info][config] [source/server/configuration_impl.cc:75] load
ing 1 cluster(s)
[2025-04-08 05:09:05.713][1][info][upstream] [source/common/upstream/cluster_manager_imp
l.cc:171] cm init: all clusters initialized
[2025-04-08 05:09:05.713][1][info][config] [source/server/configuration_impl.cc:79] load
ing 1 listener(s)
[2025-04-08 05:09:05.718][1][info][config] [source/server/configuration_impl.cc:129] loa
ding stats sink configuration
[2025-04-08 05:09:05.718][1][info][main] [source/server/server.cc:533] all clusters init
ialized. initializing init manager
[2025-04-08 05:09:05.718][1][info][config] [source/server/listener_manager_impl.cc:725]
all dependencies initialized. starting workers
```

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/