ECE 473/573 Fall 2024 - Project 4

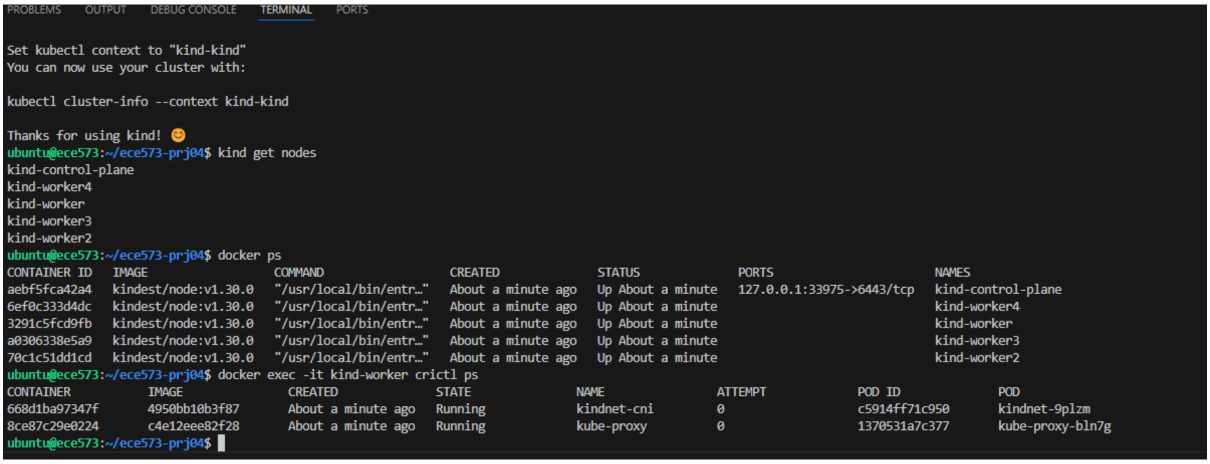
Container Orchestration with Kubernetes

Samman Chouhan

A20561414

II. kind Cluster Setup

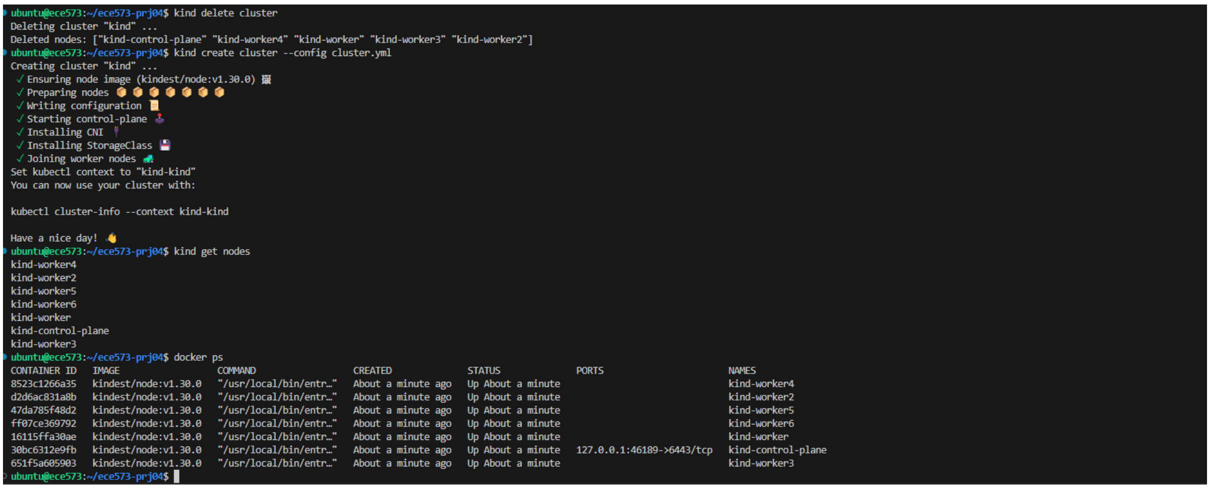
- Modify cluster.yml to include 6 worker nodes. Create the cluster and verify that it is running.



```
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Thanks for using kind! 😊
ubuntu@ece573:~/ece573-prj04$ kind get nodes
kind-control-plane
kind-worker4
kind-worker
kind-worker3
kind-worker2
ubuntu@ece573:~/ece573-prj04$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED              STATUS              PORTS                       NAMES
aebf5fca42a4   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute   127.0.0.1:33975->6443/tcp   kind-control-plane
6ef0c333d4dc   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker4
3291c5fcd9fb   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker
a0306338e5a9   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker3
70c1c51dd1cd   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker2
ubuntu@ece573:~/ece573-prj04$ docker exec -it kind-worker crictl ps
CONTAINER           IMAGE               CREATED              STATE               NAME                ATTEMPT             POD ID              POD
668d1ba97347f       4950bb10b3f87       About a minute ago   Running             kindnet-cni         0                   c5914ff71c950       kindnet-9plzm
8ce87c29e0224       c4e12eee82f28       About a minute ago   Running             kube-proxy          0                   1370531a7c377       kube-proxy-bln7g
ubuntu@ece573:~/ece573-prj04$ 
```



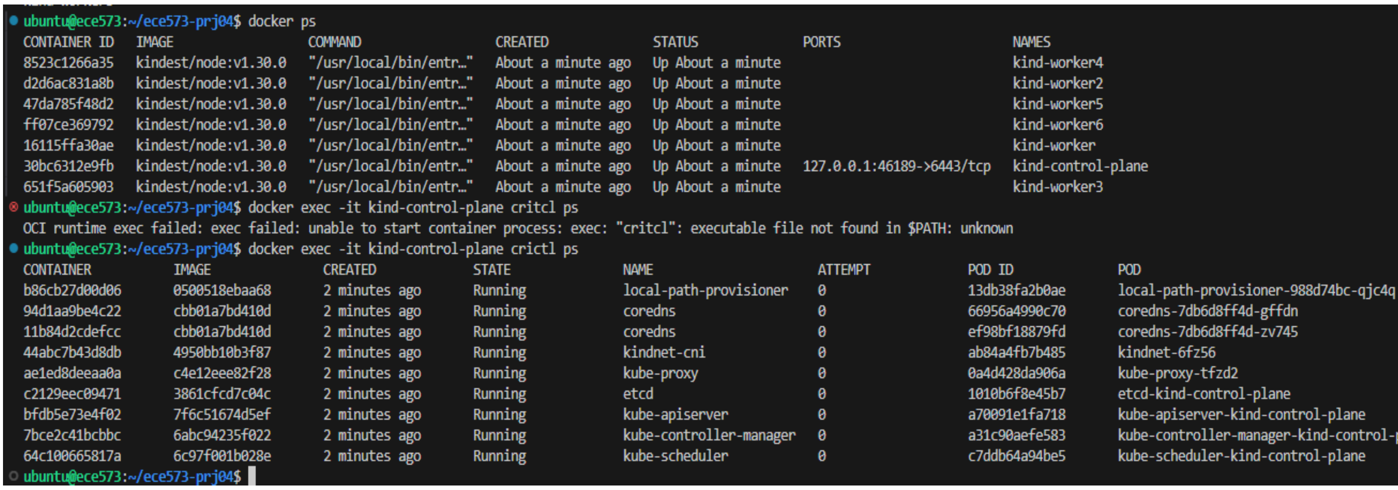
```
ubuntu@ece573:~/ece573-prj04$ kind delete cluster
Deleting cluster "kind" ...
Deleted nodes: ["kind-control-plane" "kind-worker4" "kind-worker" "kind-worker3" "kind-worker2"]
ubuntu@ece573:~/ece573-prj04$ kind create cluster --config cluster.yml
Creating cluster "kind" ...
 ✓ Ensuring node image (kindest/node:v1.30.0) 🖼
 ✓ Preparing nodes 📦 📦 📦 📦 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Have a nice day! 👋
ubuntu@ece573:~/ece573-prj04$ kind get nodes
kind-worker4
kind-worker2
kind-worker5
kind-worker6
kind-worker
kind-control-plane
kind-worker3
ubuntu@ece573:~/ece573-prj04$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED              STATUS              PORTS                       NAMES
8523c1266a35   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker4
d2d6ac831a8b   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker2
47da785f48d2   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker5
ff07ce369792   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker6
16115ffa30ae   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker
30bc6312e9fb   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute   127.0.0.1:46189->6443/tcp   kind-control-plane
651f5a605903   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker3
ubuntu@ece573:~/ece573-prj04$ 
```

The worker nodes were updated from 4 to 6

- Run **crictl ps** in the control plane node to show K8s containers running inside. Name two K8s control plane components from the list.



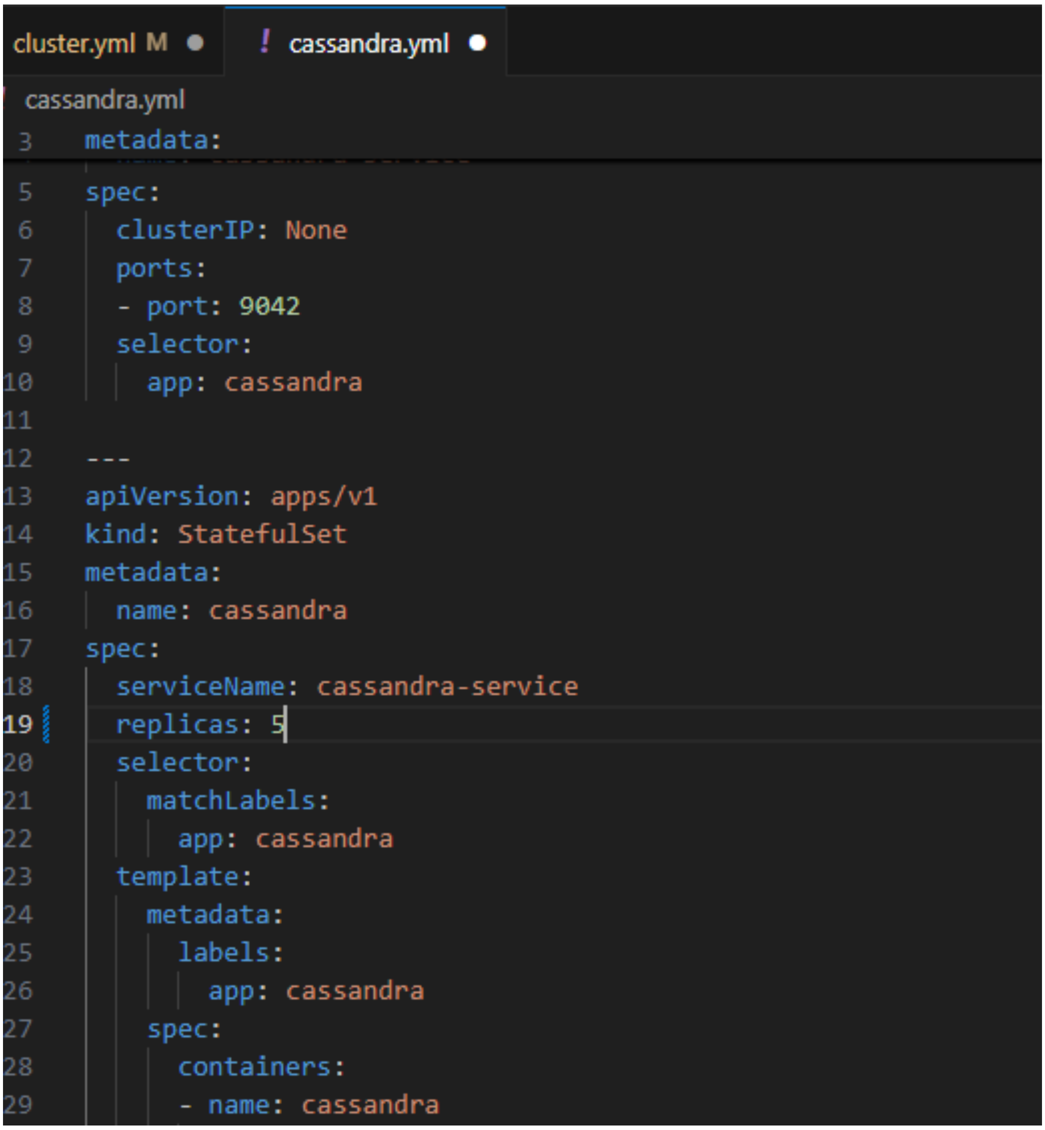
```
ubuntu@ece573:~/ece573-prj04$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED              STATUS              PORTS                       NAMES
8523c1266a35   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker4
d2d6ac831a8b   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker2
47da785f48d2   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker5
ff07ce369792   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker6
16115ffa30ae   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker
30bc6312e9fb   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute   127.0.0.1:46189->6443/tcp   kind-control-plane
651f5a605903   kindest/node:v1.30.0   "/usr/local/bin/entr…"   About a minute ago   Up About a minute                               kind-worker3
ubuntu@ece573:~/ece573-prj04$ docker exec -it kind-control-plane critcl ps
OCI runtime exec failed: exec failed: unable to start container process: exec: "critcl": executable file not found in $PATH: unknown
ubuntu@ece573:~/ece573-prj04$ docker exec -it kind-control-plane crictl ps
CONTAINER           IMAGE               CREATED             STATE               NAME                      ATTEMPT             POD ID              POD
b86cb27d00d06       0500518ebaa68       2 minutes ago       Running             local-path-provisioner    0                   13db38fa2b0ae       local-path-provisioner-988d74bc-qjc4q
94d1aa9be4c22       cbb01a7bd410d       2 minutes ago       Running             coredns                   0                   66956a4990c70       coredns-7db6d8ff4d-gffdn
11b84d2cdefcc       cbb01a7bd410d       2 minutes ago       Running             coredns                   0                   ef98bf18879fd       coredns-7db6d8ff4d-zv745
44abc7b43d8db       4950bb10b3f87       2 minutes ago       Running             kindnet-cni               0                   ab84a4fb7b485       kindnet-6fz56
ae1ed8deeaa0a       c4e12eee82f28       2 minutes ago       Running             kube-proxy                0                   0a4d428da906a       kube-proxy-tfzd2
c2129eec09471       3861cfcd7c04c       2 minutes ago       Running             etcd                      0                   1010b6f8e45b7       etcd-kind-control-plane
bfdb5e73e4f02       7f6c51674d5ef       2 minutes ago       Running             kube-apiserver            0                   a70091e1fa718       kube-apiserver-kind-control-plane
7bce2c41bcbbc       6abc94235f022       2 minutes ago       Running             kube-controller-manager   0                   a31c90aefe583       kube-controller-manager-kind-control-p
64c100665817a       6c97f001b028e       2 minutes ago       Running             kube-scheduler            0                   c7ddb64a94be5       kube-scheduler-kind-control-plane
ubuntu@ece573:~/ece573-prj04$
```

Name - Kindnet-cni

Name – etcd

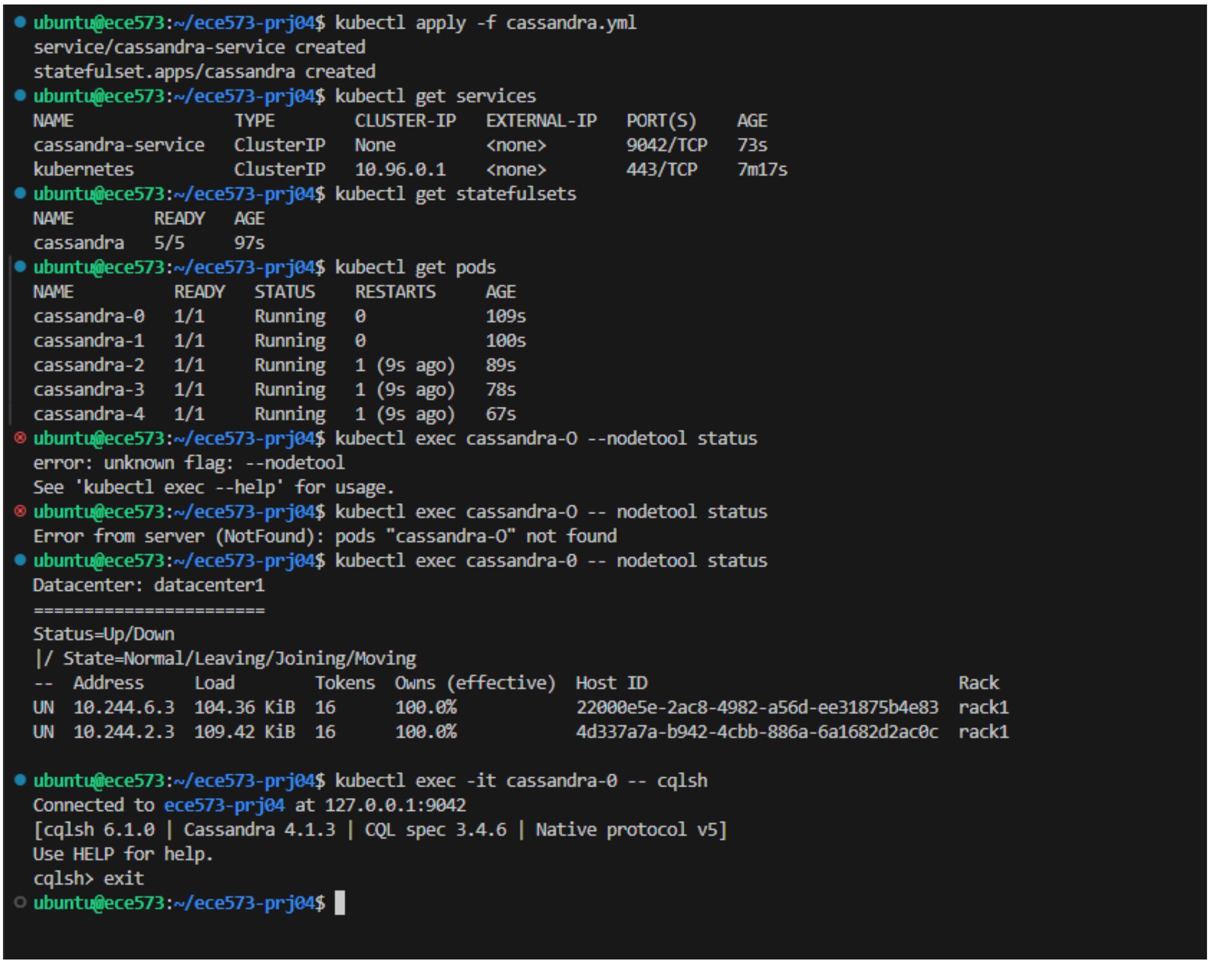
III. The Cassandra Service

- Modify cassandra.yml to include 5 Cassandra replicas. Create the service and verify that Cassandra is running properly.



```yaml
metadata:
spec:
  clusterIP: None
  ports:
  - port: 9042
  selector:
    app: cassandra

---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: cassandra
spec:
  serviceName: cassandra-service
  replicas: 5
  selector:
    matchLabels:
      app: cassandra
  template:
    metadata:
      labels:
        app: cassandra
    spec:
      containers:
      - name: cassandra
```



```
ubuntu@ece573:~/ece573-prj04$ kubectl apply -f cassandra.yml
service/cassandra-service created
statefulset.apps/cassandra created
ubuntu@ece573:~/ece573-prj04$ kubectl get services
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
cassandra-service   ClusterIP   None         <none>        9042/TCP   73s
kubernetes          ClusterIP   10.96.0.1    <none>        443/TCP    7m17s
ubuntu@ece573:~/ece573-prj04$ kubectl get statefulsets
NAME        READY   AGE
cassandra   5/5     97s
ubuntu@ece573:~/ece573-prj04$ kubectl get pods
NAME          READY   STATUS    RESTARTS     AGE
cassandra-0   1/1     Running   0            109s
cassandra-1   1/1     Running   0            100s
cassandra-2   1/1     Running   1 (9s ago)   89s
cassandra-3   1/1     Running   1 (9s ago)   78s
cassandra-4   1/1     Running   1 (9s ago)   67s
ubuntu@ece573:~/ece573-prj04$ kubectl exec cassandra-O --nodetool status
error: unknown flag: --nodetool
See 'kubectl exec --help' for usage.
ubuntu@ece573:~/ece573-prj04$ kubectl exec cassandra-O -- nodetool status
Error from server (NotFound): pods "cassandra-O" not found
ubuntu@ece573:~/ece573-prj04$ kubectl exec cassandra-0 -- nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load        Tokens  Owns (effective)  Host ID                               Rack
UN  10.244.6.3   104.36 KiB  16      100.0%            22000e5e-2ac8-4982-a56d-ee31875b4e83  rack1
UN  10.244.2.3   109.42 KiB  16      100.0%            4d337a7a-b942-4cbb-886a-6a1682d2ac0c  rack1

ubuntu@ece573:~/ece573-prj04$ kubectl exec -it cassandra-0 -- cqlsh
Connected to ece573-prj04 at 127.0.0.1:9042
[cqlsh 6.1.0 | Cassandra 4.1.3 | CQL spec 3.4.6 | Native protocol v5]
Use HELP for help.
cqlsh> exit
ubuntu@ece573:~/ece573-prj04$ █
```

- Explain the "resources" section for the cassandra container from cassandra.yml . What are the difference between "limits" and "requests"?

The "resources" section of the cassandra.yml file sets the container's CPU and RAM resources and limitations. This is crucial for controlling resource allocation and scheduling in a Kubernetes cluster. The "resources" section includes two crucial fields: limits and requests.
Limits: Limits specify the maximum CPU and memory resources a container can consume.They serve as an upper limit or constraint on resource utilization.If a container exceeds its limits, Kubernetes may throttle or terminate it to prevent excessive resource consumption.Limits ensure cluster stability and fairness by prohibiting individual containers from monopolizing available resources.

Requests specify the initial CPU and memory resources requested by a container at startup. Kubernetes relies on these resource demands to make scheduling decisions. It assigns the pod to nodes with sufficient resources to meet the requests.The Kubernetes scheduler uses resource requests to ensure pods are assigned to nodes with sufficient resources. In the cassandra.yml file, resource requests are specified as follows:

- Below the "resources" section you will find the section "env" to setup environment variables. Where was the corresponding part for Project 3? Explain the difference between the two.

In Project 3, environment variables such as "env" were not explicitly set in the YAML setup. Instead, the project relied on runtime environment variables or command-line arguments within the Go code. Project 3 used command-line arguments to configure environment variables such as CASSANDRA_CLUSTER_NAME and CASSANDRA_SEEDS, unlike Project 4's configuration file.
The primary distinction between the two projects is the technique to setting environment variables:

Project 3 involved dynamically setting environment variables while running Go client programs in the terminal. To launch the writer or reader programs, you can specify variables such as consistency level and seed node via the command line. These values were supplied as parameters to Go applications, which configured connections to the Cassandra cluster.
In Project 4, the configuration file (YAML) has a "env" section where you can define environment variables for containers in the Kubernetes cluster. The configuration file defines environment variables that can be accessible by Kubernetes cluster components.

- How does the Cassandra service know which Pods are part of the service?

In Kubernetes, services, such as Cassandra, employ labels and selectors to identify which pods belong to their service. Labels are key-value pairs used to identify and categorize items, including pods. Selectors are used to match items to certain labels. Here's how the Cassandra service works:

Pod Labeling: In your StatefulSet configuration, you have defined labels for the Cassandra pods:

template:

metadata:

labels:

app: cassandraT

his labels each Cassandra pod with the key "app" and the value "cassandra."

Service Selector: In your service configuration, you have specified a selector that defines which

pods are part of the service:

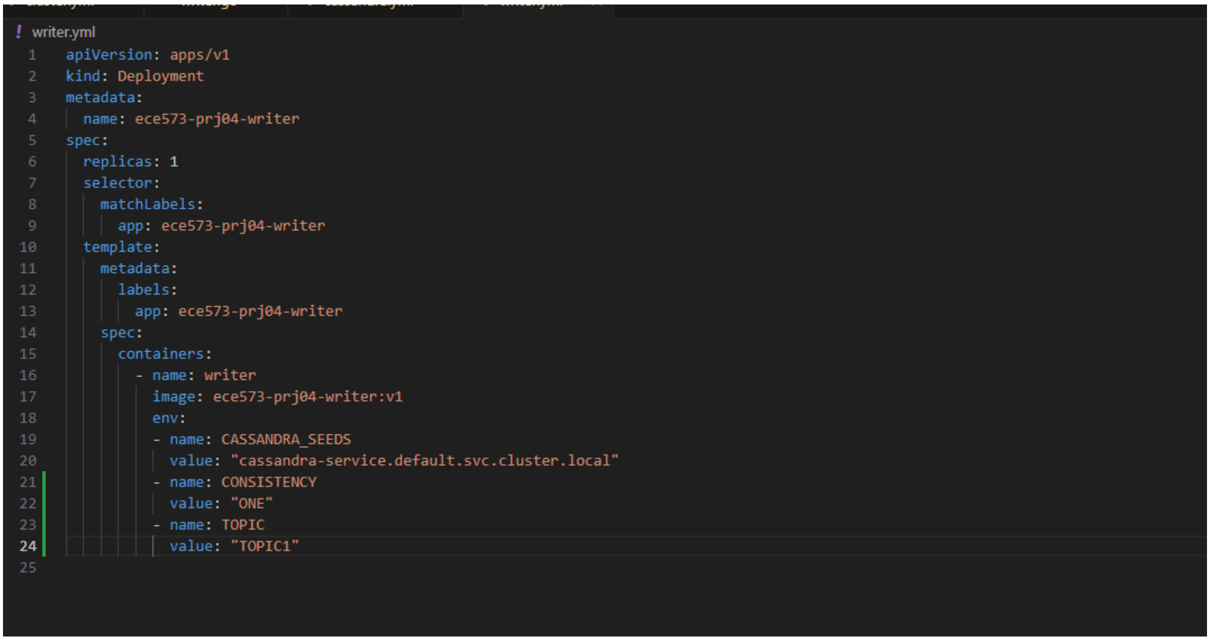
selector:

app: cassandra

This selector instructs the Cassandra service to include pods labeled "app: cassandra" within the service.
Matching Labels: The Kubernetes service controller continuously monitors and labels the cluster's pods. Pods labeled "app: cassandra" are considered part of the Cassandra service as they match the service's selector.
So the service knows which pods are included by picking pods with the provided label. This separates the service from individual pod IP addresses, which may vary due to scaling or rescheduling. Labels are used to specify the group of pods included in the service, providing a more flexible approach. The label-based method simplifies managing and scaling services in a dynamic Kubernetes environment.

IV. Build and Deploy an Application

- Correct writer.yml as mentioned above and verify everying is running properly.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ece573-prj04-writer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ece573-prj04-writer
  template:
    metadata:
      labels:
        app: ece573-prj04-writer
    spec:
      containers:
        - name: writer
          image: ece573-prj04-writer:v1
          env:
          - name: CASSANDRA_SEEDS
            value: "cassandra-service.default.svc.cluster.local"
          - name: CONSISTENCY
            value: "ONE"
          - name: TOPIC
            value: "TOPIC1"
```

- How does the writer deployment connect to the Cassandra service? In particular, where does "cassandra-service.default.svc.cluster.local" come from?

  The writer deployment in your Kubernetes cluster accesses the Cassandra service via the DNS name "cassandra-service.default.svc.cluster.local." Let's look at the DNS name and its origin:
  "cassandra-service": This section of the DNS name indicates the Cassandra service name to which the writer deployment connects. The service name is defined in the "CASSANDRA_SEEDS" environment variable in your writer.yml file. "default": This indicates the namespace that contains the Cassandra service. Namespaces are a sensible approach for Kubernetes to organize and separate resources. If you don't specify a namespace for your service, the "default" namespace will be utilized. "svc.cluster.local" is the domain suffix for services within the cluster. Kubernetes automatically resolves DNS for services in the cluster using this domain.
  Kubernetes resolves the fully qualified domain name (FQDN) "cassandra-service.default.svc.cluster.local" to the Cassandra service's IP address. The DNS name is produced automatically based on the service name and namespace, allowing pods in the same Kubernetes cluster to communicate with the

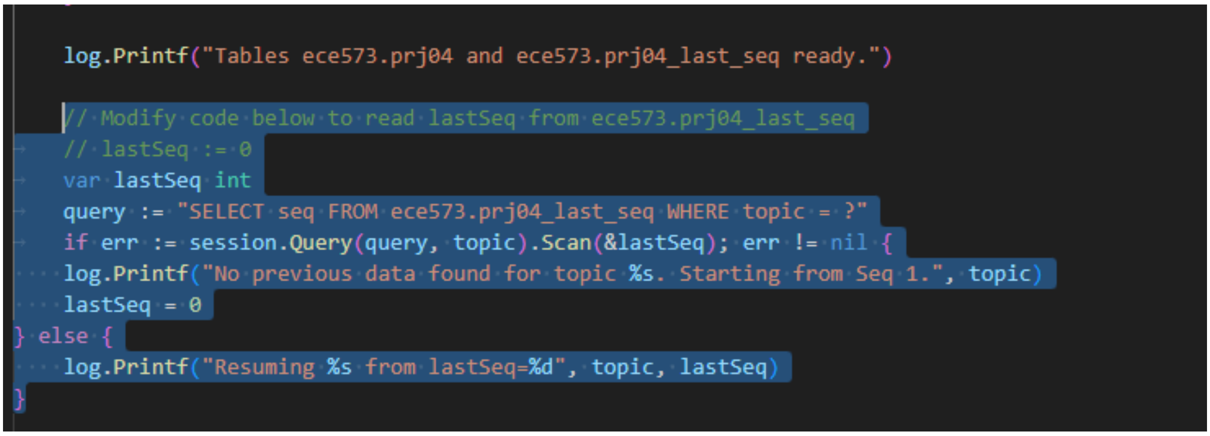
Cassandra service.
To connect to the Cassandra service in a Kubernetes cluster, use "cassandra-service.default.svc.cluster.local" with your service name and default namespace. This FQDN allows pods to interface with services in a consistent and dynamic manner, regardless of their IP addresses.

- We add retrying logic to the writer program in Project 3. Do we need it for Project 4?

The need for retrying logic in the writer program for Project 4 is determined by the project's specific requirements.
In Project 4, you will work with a Kubernetes cluster and deploy services to it. Kubernetes is a container orchestration and management platform with robust reliability and fault tolerance characteristics. These features include automatically restarting failing containers and monitoring service health. Kubernetes supports scaling and load balancing. However, whether you need retrying logic in the writer program depends on the nature of the operations and the specific use case. If you are performing write operations to a database orservice within the Kubernetes cluster and want to ensure that failed writes are retried, you may still need to implement retrying logic in your writer program. It could be necessary in situations where you want to provide additional resilience beyond what Kubernetes offers.
In summary, while Kubernetes provides some level of fault tolerance and reliability, the need for retrying logic in the writer program for Project 4 will depend on the specific requirements and considerations of your project and whether you want to implement custom retry behavior forwrite operations.

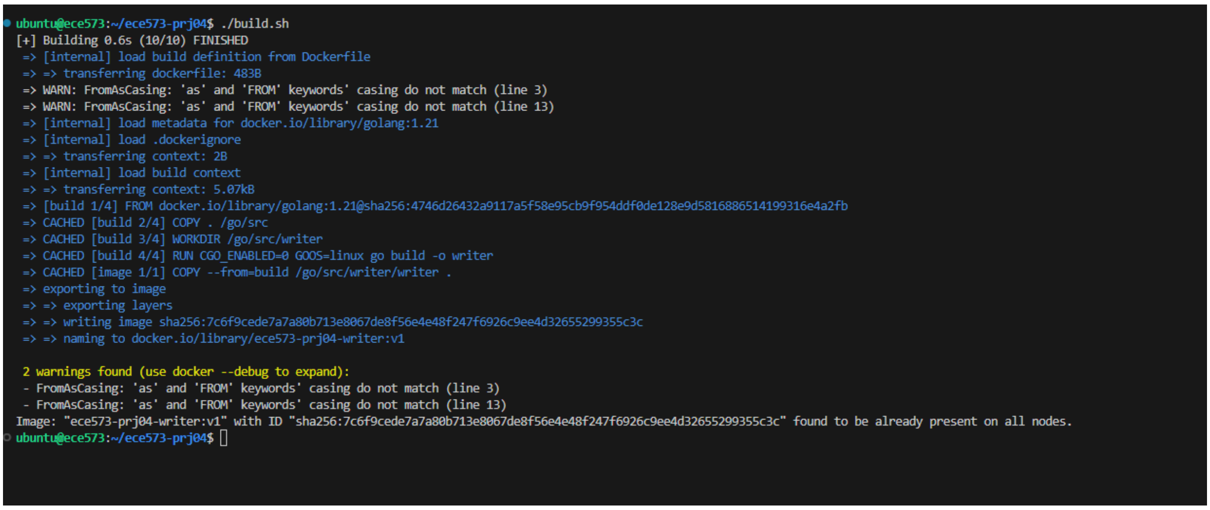
V. Stateless Application

- Modify writer.go as mentioned above and verify everying is running properly. You will need to delete Pods to trigger writer to restart and show log messages indicating lastSeq reading from Cassandra.



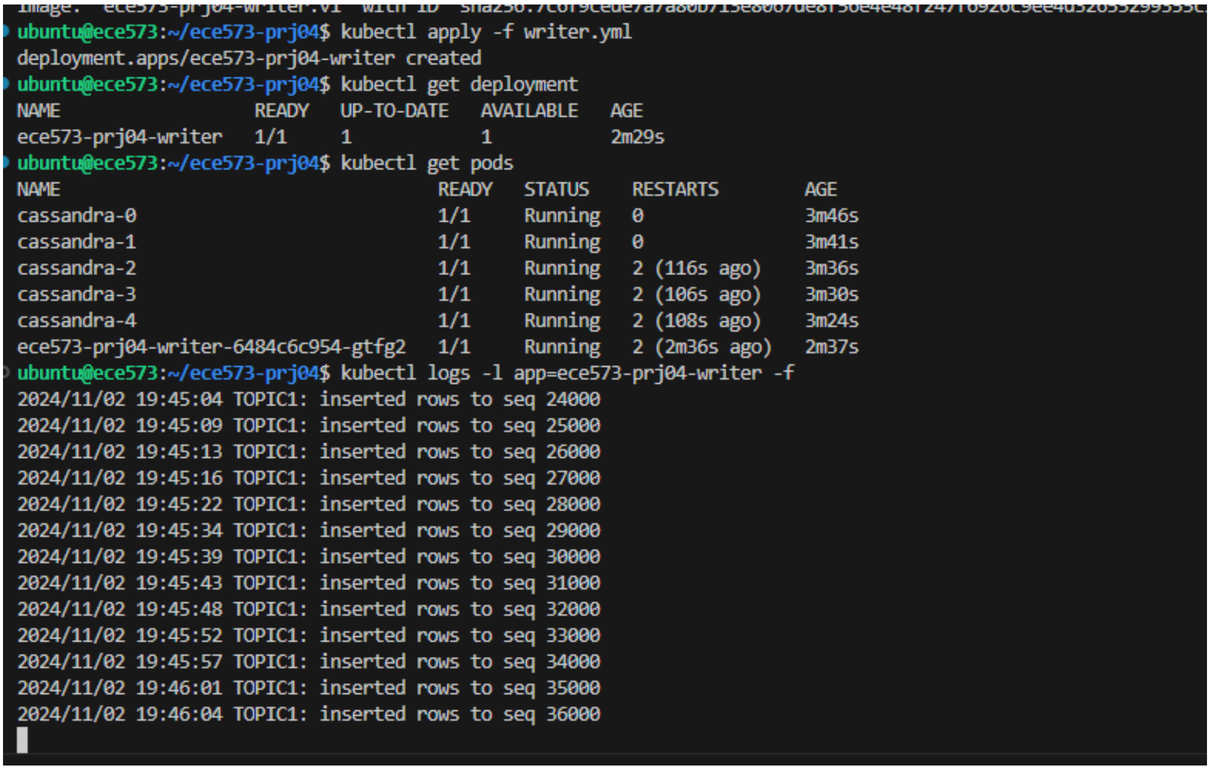
```
    log.Printf("Tables ece573.prj04 and ece573.prj04_last_seq ready.")

    // Modify code below to read lastSeq from ece573.prj04_last_seq
    // lastSeq := 0
    var lastSeq int
    query := "SELECT seq FROM ece573.prj04_last_seq WHERE topic = ?"
    if err := session.Query(query, topic).Scan(&lastSeq); err != nil {
    log.Printf("No previous data found for topic %s. Starting from Seq 1.", topic)
    lastSeq = 0
} else {
    log.Printf("Resuming %s from lastSeq=%d", topic, lastSeq)
}
```



```
ubuntu@ece573:~/ece573-prj04$ ./build.sh
[+] Building 0.6s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 483B
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 3)
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 13)
 => [internal] load metadata for docker.io/library/golang:1.21
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 5.07kB
 => [build 1/4] FROM docker.io/library/golang:1.21@sha256:4746d26432a9117a5f58e95cb9f954ddf0de128e9d5816886514199316e4a2fb
 => CACHED [build 2/4] COPY . /go/src
 => CACHED [build 3/4] WORKDIR /go/src/writer
 => CACHED [build 4/4] RUN CGO_ENABLED=0 GOOS=linux go build -o writer
 => CACHED [image 1/1] COPY --from=build /go/src/writer/writer .
 => exporting to image
 => => exporting layers
 => => writing image sha256:7c6f9cede7a7a80b713e8067de8f56e4e48f247f6926c9ee4d32655299355c3c
 => => naming to docker.io/library/ece573-prj04-writer:v1

 2 warnings found (use docker --debug to expand):
 - FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 3)
 - FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 13)
Image: "ece573-prj04-writer:v1" with ID "sha256:7c6f9cede7a7a80b713e8067de8f56e4e48f247f6926c9ee4d32655299355c3c" found to be already present on all nodes.
ubuntu@ece573:~/ece573-prj04$ 
```



```
 image: ece573-prj04-writer:v1 with ID sha256:7c0f9cede7a7a80b713e8007de8130e4e481247f0920c9ee40320332995355c
ubuntu@ece573:~/ece573-prj04$ kubectl apply -f writer.yml
deployment.apps/ece573-prj04-writer created
ubuntu@ece573:~/ece573-prj04$ kubectl get deployment
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
ece573-prj04-writer   1/1     1            1           2m29s
ubuntu@ece573:~/ece573-prj04$ kubectl get pods
NAME                                   READY   STATUS    RESTARTS        AGE
cassandra-0                            1/1     Running   0               3m46s
cassandra-1                            1/1     Running   0               3m41s
cassandra-2                            1/1     Running   2 (116s ago)    3m36s
cassandra-3                            1/1     Running   2 (106s ago)    3m30s
cassandra-4                            1/1     Running   2 (108s ago)    3m24s
ece573-prj04-writer-6484c6c954-gtfg2   1/1     Running   2 (2m36s ago)   2m37s
ubuntu@ece573:~/ece573-prj04$ kubectl logs -l app=ece573-prj04-writer -f
2024/11/02 19:45:04 TOPIC1: inserted rows to seq 24000
2024/11/02 19:45:09 TOPIC1: inserted rows to seq 25000
2024/11/02 19:45:13 TOPIC1: inserted rows to seq 26000
2024/11/02 19:45:16 TOPIC1: inserted rows to seq 27000
2024/11/02 19:45:22 TOPIC1: inserted rows to seq 28000
2024/11/02 19:45:34 TOPIC1: inserted rows to seq 29000
2024/11/02 19:45:39 TOPIC1: inserted rows to seq 30000
2024/11/02 19:45:43 TOPIC1: inserted rows to seq 31000
2024/11/02 19:45:48 TOPIC1: inserted rows to seq 32000
2024/11/02 19:45:52 TOPIC1: inserted rows to seq 33000
2024/11/02 19:45:57 TOPIC1: inserted rows to seq 34000
2024/11/02 19:46:01 TOPIC1: inserted rows to seq 35000
2024/11/02 19:46:04 TOPIC1: inserted rows to seq 36000
```

- What happens if consistency is broken so that the writer doesn't obtain the most recent last seq? Will this cause an issue for the writer?

If consistency is lost and the writer doesn't get the most current last sequence number from the ece573.prj04_last_seq database, they may start with an outdated or inaccurate sequence number. This can affect data integrity and sequence synchronization between the writer and Cassandra database.

Issues that might persist :

Incorrect Resumption: Failure to acquire the most current final sequence number can result in duplicate or skipped entries in the Cassandra database.

Data Inconsistency: The writer's data may not match the sequence in the database. Inconsistencies can arise when data is not written sequentially or synchronized with other system components.

Data loss: Starting from an erroneous sequence number can result in overwriting or missing essential data, resulting to corruption.

To avoid these difficulties, writers should retrieve the last sequence number consistently. To ensure accurate data retrieval, Cassandra queries should use the proper consistency level to represent the most recent database state.

The provided code sets the consistency level based on the CONSISTENCY environment variable. To achieve the correct level of consistency in queries, this variable must be properly configured. Setting CONSISTENCY=QUORUM can improve consistency, but may negatively influence performance.

Maintain data integrity by adjusting your application's setup based on its consistency requirements.