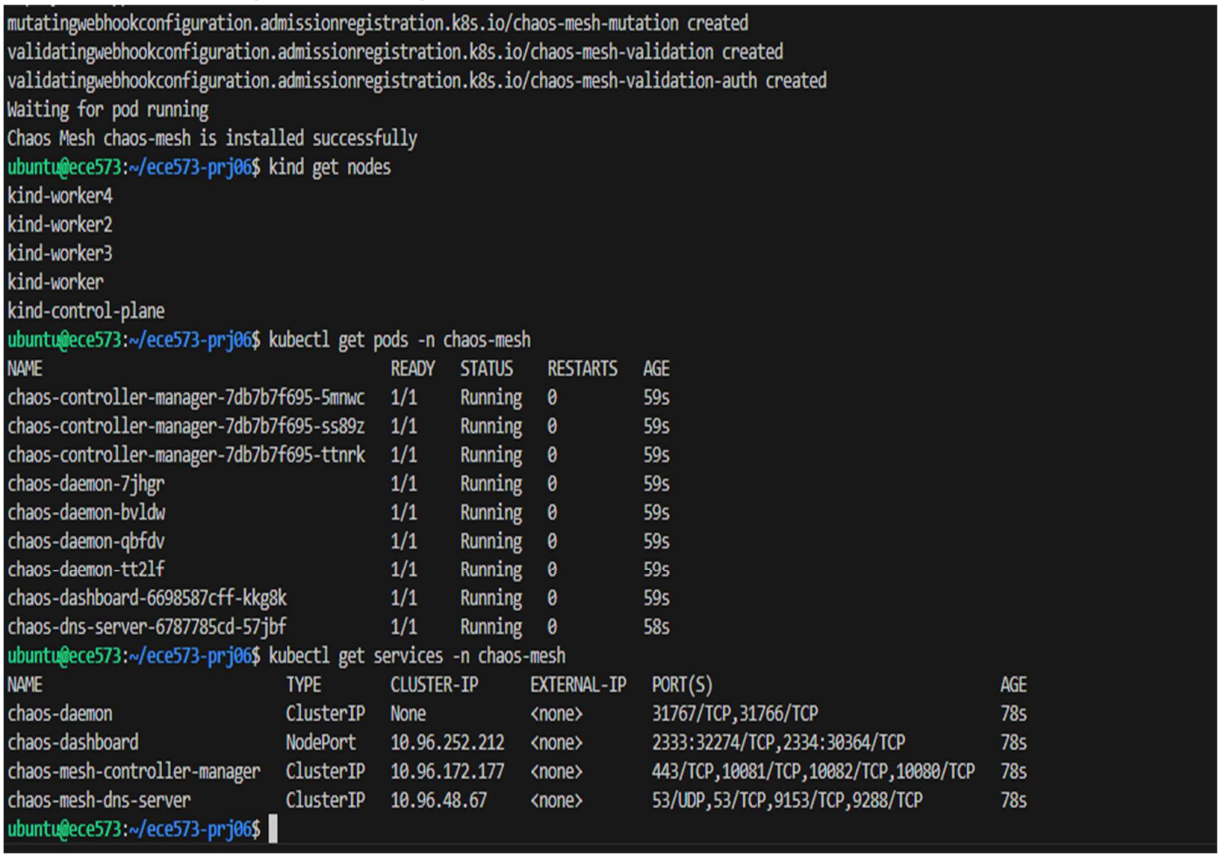
ECE 573 Fall 2024 - Project 6

Chaos Engineering

Samman Chouhan
A20561414

II. Chaos Engineering with Chaos Mesh



```
mutatingwebhookconfiguration.admissionregistration.k8s.io/chaos-mesh-mutation created
validatingwebhookconfiguration.admissionregistration.k8s.io/chaos-mesh-validation created
validatingwebhookconfiguration.admissionregistration.k8s.io/chaos-mesh-validation-auth created
Waiting for pod running
Chaos Mesh chaos-mesh is installed successfully
ubuntu@ece573:~/ece573-prj06$ kind get nodes
kind-worker4
kind-worker2
kind-worker3
kind-worker
kind-control-plane
ubuntu@ece573:~/ece573-prj06$ kubectl get pods -n chaos-mesh
NAME                                       READY   STATUS    RESTARTS   AGE
chaos-controller-manager-7db7b7f695-5mnwc  1/1     Running   0          59s
chaos-controller-manager-7db7b7f695-ss89z  1/1     Running   0          59s
chaos-controller-manager-7db7b7f695-ttnrk  1/1     Running   0          59s
chaos-daemon-7jhgr                         1/1     Running   0          59s
chaos-daemon-bvldw                         1/1     Running   0          59s
chaos-daemon-qbfdv                         1/1     Running   0          59s
chaos-daemon-tt2lf                         1/1     Running   0          59s
chaos-dashboard-6698587cff-kkg8k           1/1     Running   0          59s
chaos-dns-server-6787785cd-57jbf           1/1     Running   0          58s
ubuntu@ece573:~/ece573-prj06$ kubectl get services -n chaos-mesh
NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                              AGE
chaos-daemon                  ClusterIP   None            <none>        31767/TCP,31766/TCP                  78s
chaos-dashboard               NodePort    10.96.252.212   <none>        2333:32274/TCP,2334:30364/TCP        78s
chaos-mesh-controller-manager ClusterIP   10.96.172.177   <none>        443/TCP,10081/TCP,10082/TCP,10080/TCP 78s
chaos-mesh-dns-server         ClusterIP   10.96.48.67     <none>        53/UDP,53/TCP,9153/TCP,9288/TCP      78s
ubuntu@ece573:~/ece573-prj06$
```

- What are the Pods used by Chaos Mesh? Make an educated guess about their functionalities.

chaos-controller-manager-95997648-7tckn, chaos-controller-manager-95997648-8vn2f,
chaos-controller-manager-95997648-szg2v:

Functionality: These pods are probably in charge of overseeing and managing experiments involving chaos. They might be in charge of managing the coordination of fault injection procedures and the general operation of Chaos Mesh.

chaos-daemon-5qcv6, chaos-daemon-mlcvk, chaos-daemon-tkr6j, chaos-daemon-xtgwv:

Functionality: These pods are most likely the chaos daemons that introduce errors into the system. To test the system's robustness, they carry out the designated chaotic experiments and introduce errors.

chaos-dashboard-5dd6c987fb-frr85:

Functionality: This pod probably houses Chaos Mesh's web-based dashboard. The dashboard facilitates user interaction with Chaos Mesh by offering a user interface for organizing and visualizing chaos experiments.

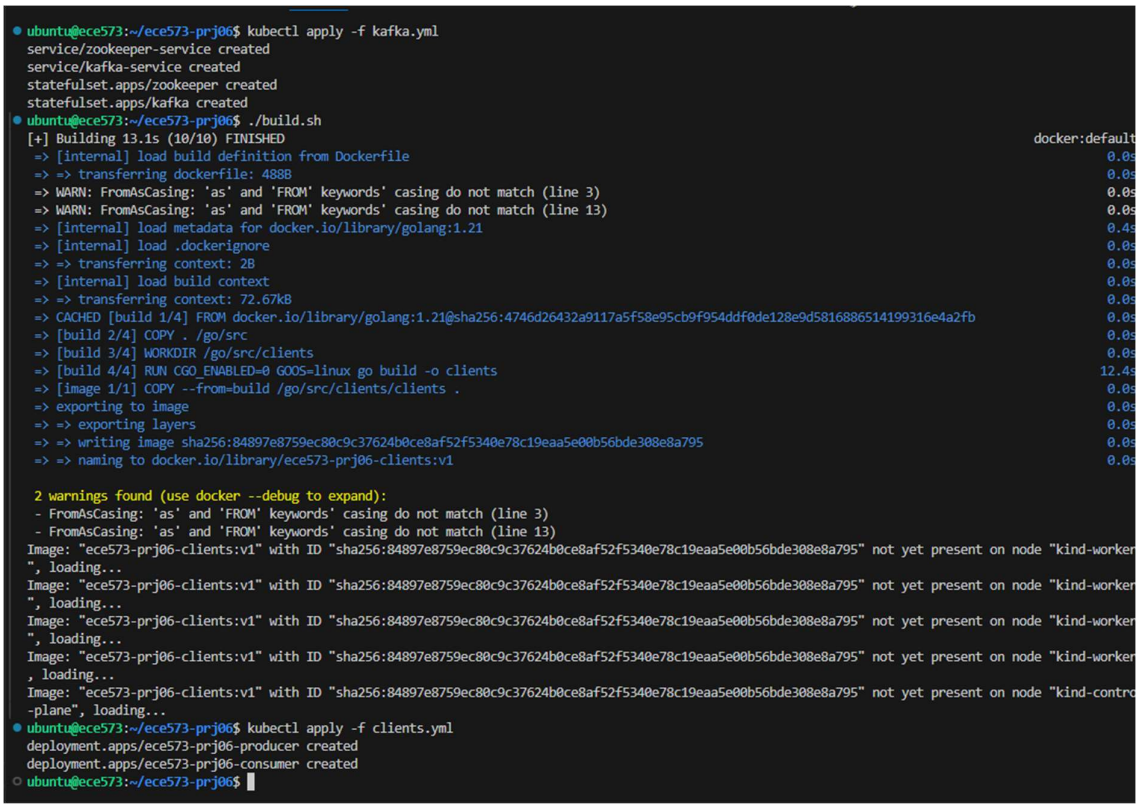
chaos-dns-server-785cc6db5f-6mqln:
Functionality: Within the chaos-mesh namespace, this pod most likely serves as a DNS server for Chaos Mesh, offering domain name resolving services.

- Instead of searching online, where are you going to find the service name and port required by **kubectl port-forward**?

kubectl get services -n chaos-mesh

The services in the "chaos-mesh" namespace will be listed by this command, together with the names, types, cluster IPs, and ports that relate to each service. Locate the "chaos-dashboard" service, which is linked to the Chaos Mesh dashboard, and take note of the port in the "PORT(S)" column. The Chaos Mesh dashboard in the example is accessible on port 2303.

III. Pod Faults



```
ubuntu@ece573:~/ece573-prj06$ kubectl apply -f clients.yml
deployment.apps/ece573-prj06-producer created
deployment.apps/ece573-prj06-consumer created
ubuntu@ece573:~/ece573-prj06$ kubectl logs -l app=ece573-prj06-producer
2024/11/28 01:11:33 test: 222000 messages published
2024/11/28 01:11:34 test: 223000 messages published
2024/11/28 01:11:34 test: 224000 messages published
2024/11/28 01:11:34 test: 225000 messages published
2024/11/28 01:11:34 test: 226000 messages published
2024/11/28 01:11:35 test: 227000 messages published
2024/11/28 01:11:35 test: 228000 messages published
2024/11/28 01:11:35 test: 229000 messages published
2024/11/28 01:11:35 test: 230000 messages published
2024/11/28 01:11:35 test: 231000 messages published
ubuntu@ece573:~/ece573-prj06$ kubectl logs -l app=ece573-prj06-consumer
2024/11/28 01:11:40 test: received 62000 messages, last (0.100932)
2024/11/28 01:11:40 test: received 63000 messages, last (0.909160)
2024/11/28 01:11:41 test: received 64000 messages, last (0.067701)
2024/11/28 01:11:42 test: received 65000 messages, last (0.416805)
2024/11/28 01:11:43 test: received 66000 messages, last (0.873129)
2024/11/28 01:11:44 test: received 67000 messages, last (0.911850)
2024/11/28 01:11:45 test: received 68000 messages, last (0.251373)
2024/11/28 01:11:46 test: received 69000 messages, last (0.095371)
2024/11/28 01:11:46 test: received 70000 messages, last (0.190123)
2024/11/28 01:11:47 test: received 71000 messages, last (0.382761)
ubuntu@ece573:~/ece573-prj06$ kubectl delete -f clients.yml
deployment.apps "ece573-prj06-producer" deleted
deployment.apps "ece573-prj06-consumer" deleted
ubuntu@ece573:~/ece573-prj06$  kubectl exec kafka-0 -- kafka-topics --bootstrap-server localhost:9092 --describe test
Topic: test     TopicId: Tt_byL2wSD29ONoiAvlvig PartitionCount: 4       ReplicationFactor: 3    Configs:
        Topic: test     Partition: 0    Leader: 0       Replicas: 0,2,1 Isr: 0,2,1
        Topic: test     Partition: 1    Leader: 2       Replicas: 2,1,0 Isr: 2,1,0
        Topic: test     Partition: 2    Leader: 1       Replicas: 1,0,2 Isr: 1,0,2
        Topic: test     Partition: 3    Leader: 0       Replicas: 0,1,2 Isr: 0,1,2
ubuntu@ece573:~/ece573-prj06$
```

```
ubuntu@ece573:~/ece573-prj06$ kubectl apply -f pod-failure.yml
podchaos.chaos-mesh.org/pod-failure created
ubuntu@ece573:~/ece573-prj06$ kubectl get pods
NAME          READY   STATUS             RESTARTS     AGE
kafka-0       1/1     Running            0            5m54s
kafka-1       0/1     CrashLoopBackOff   2 (7s ago)   5m49s
kafka-2       0/1     CrashLoopBackOff   2 (7s ago)   5m43s
zookeeper-0   1/1     Running            0            5m54s
ubuntu@ece573:~/ece573-prj06$  kubectl exec kafka-2 -- kafka-topics --bootstrap-server localhost:9092 --describe test
error: Internal error occurred: unable to upgrade connection: container not found ("kafka")
ubuntu@ece573:~/ece573-prj06$ kubectl exec kafka-2 -- kafka-topics --bootstrap-server localhost:9092 --describe test
error: Internal error occurred: unable to upgrade connection: container not found ("kafka")
ubuntu@ece573:~/ece573-prj06$  kubectl exec kafka-2 -- kafka-topics --bootstrap-server localhost:9092 --describe test
error: Internal error occurred: unable to upgrade connection: container not found ("kafka")
ubuntu@ece573:~/ece573-prj06$ kubectl exec kafka-0 -- kafka-topics --bootstrap-server localhost:9092 --describe test
Topic: test     TopicId: Tt_byL2wSD29ONoiAvlvig PartitionCount: 4       ReplicationFactor: 3    Configs:
        Topic: test     Partition: 0    Leader: 0       Replicas: 0,2,1 Isr: 0
        Topic: test     Partition: 1    Leader: 0       Replicas: 2,1,0 Isr: 0
        Topic: test     Partition: 2    Leader: 0       Replicas: 1,0,2 Isr: 0
        Topic: test     Partition: 3    Leader: 0       Replicas: 0,1,2 Isr: 0
ubuntu@ece573:~/ece573-prj06$ kubectl delete -f pod-failure.yml
podchaos.chaos-mesh.org "pod-failure" deleted
ubuntu@ece573:~/ece573-prj06$ kubectl get pods
NAME          READY   STATUS    RESTARTS        AGE
kafka-0       1/1     Running   0               19m
kafka-1       1/1     Running   9 (2m18s ago)   19m
kafka-2       1/1     Running   9 (2m34s ago)   19m
zookeeper-0   1/1     Running   0               19m
ubuntu@ece573:~/ece573-prj06$  kubectl exec kafka-2 -- kafka-topics --bootstrap-server localhost:9092 --describe test
Topic: test     TopicId: Tt_byL2wSD29ONoiAvlvig PartitionCount: 4       ReplicationFactor: 3    Configs:
        Topic: test     Partition: 0    Leader: 0       Replicas: 0,2,1 Isr: 0,1,2
        Topic: test     Partition: 1    Leader: 0       Replicas: 2,1,0 Isr: 0,1,2
        Topic: test     Partition: 2    Leader: 0       Replicas: 1,0,2 Isr: 0,1,2
        Topic: test     Partition: 3    Leader: 0       Replicas: 0,1,2 Isr: 0,1,2
ubuntu@ece573:~/ece573-prj06$
```

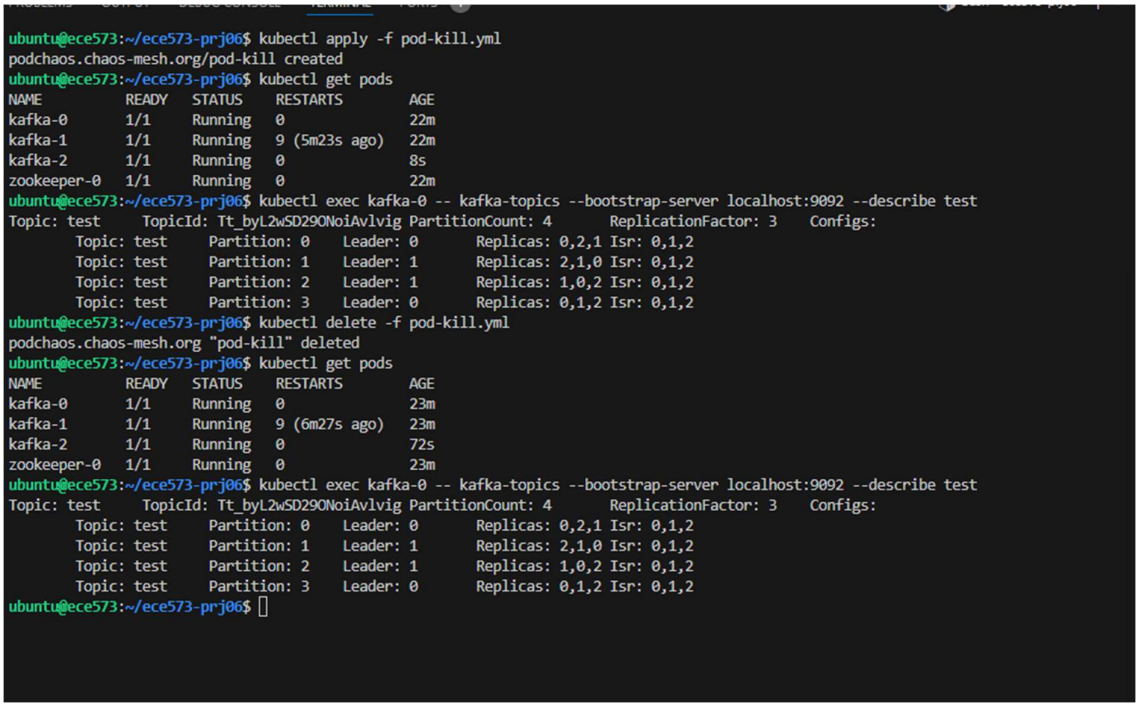
- In pod-failure.yml, which part defines where the faults happen? I.e. how to define which Pods are affected?

The selector field in the pod-failure.yml file contains the information defining the location of the defects. Which Pods are impacted by the issue is specifically determined by the label selectors in the selector field. In this instance, pods with the label app: kafka are affected.
The pertinent portion of the YAML is as follows:

selector:
labelSelectors:
app: kafka

This implies that the pod-failure action will target any pods that have the label app: kafka.
Pods that match the designated label selectors will be affected by the error.

- Read pod-kill.yml and perform an experiment with it using **kubectl apply** and **kubectl delete**. Explain how Kafka reacts to this fault.

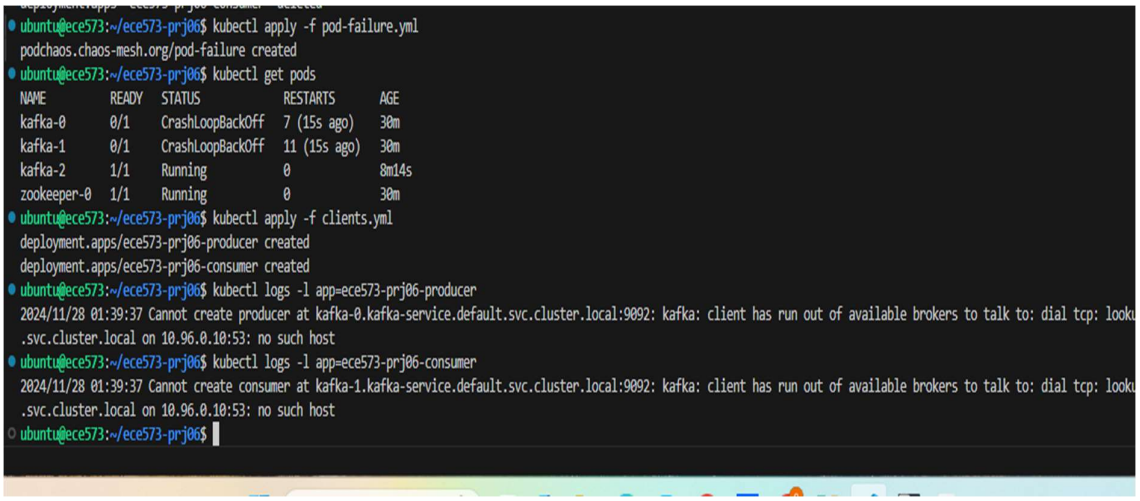


A) Noted Kafka's Reaction: After noticing that the Pod has ended, Kafka will act to adjust to its loss.
The In-sync Replicas (Isr) will be updated in accordance with the reelection of the leaders for the impacted partitions. In order to maintain synchronization among the remaining replicas, Kafka will redistribute the workload.
Justification:
A pod-kill action that targets pods using the label app: kafka is defined in the pod-kill.yml file.
One Pod (value: "1") is killed by the defect for a predetermined amount of time.
When the grace period is set to zero, the pod ends instantly and there is no waiting period.

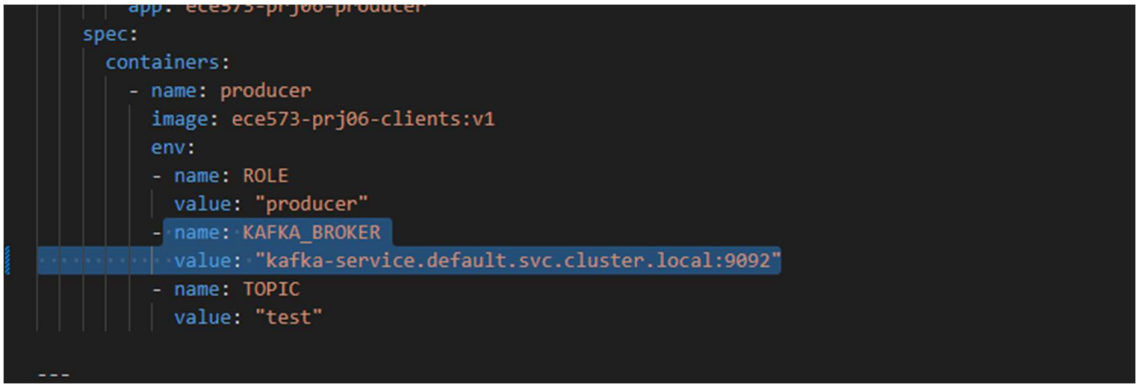
- What is the difference from the two fault types pod-failure and pod-kill?

Pod-failure and pod-kill are two defect kinds that differ primarily in what they do:
Failure of the pod: By putting pods into a RunContainerError state, this defect type mimics a failure scenario. The impacted pods are marked as failed for the length of the fault, which is applied. Two pods are configured to fail for 3600 seconds in the given example.
pod-kill: Specific pods are expressly terminated by this fault category. It permits the Pods to be rescheduled after killing them for a predetermined amount of time. One pod is killed for 60 seconds in the above scenario.

- Use **kubectl apply** to inject pod-failure.yml again and then start the clients. Are producer and consumer working properly? Modify clients.yml so the clients can work when two random Kafka Pods fail. Don't forget to remove the fault by **kubectl delete** before you would like to inject it again.



Modification in Client.yml for both consumer and producer



```
ubuntu@ece573:~/ece573-prj06$ kubectl logs -l app=ece573-prj06-consumer
2024/11/28 01:42:29 Cannot create consumer at kafka-1.kafka-service.default.svc.cluster.local:9092: kafka: client has run out of available brokers to talk to: dial tcp
.svc.cluster.local on 10.96.0.10:53: no such host
ubuntu@ece573:~/ece573-prj06$ kubectl apply -f clients.yml
deployment.apps/ece573-prj06-producer unchanged
deployment.apps/ece573-prj06-consumer configured
ubuntu@ece573:~/ece573-prj06$ kubectl get pods
NAME                                    READY   STATUS             RESTARTS        AGE
ece573-prj06-consumer-5c8955c4c6-khtrg  1/1     Running            0               4s
ece573-prj06-producer-74fb6d5ffd-xl6jm  1/1     Running            0               78s
kafka-0                                 0/1     CrashLoopBackOff   8 (2m32s ago)   35m
kafka-1                                 0/1     RunContainerError  12 (9s ago)     35m
kafka-2                                 1/1     Running            0               13m
zookeeper-0                             1/1     Running            0               35m
ubuntu@ece573:~/ece573-prj06$ kubectl logs -l app=ece573-prj06-producer
2024/11/28 01:44:25 test: 515000 messages published
2024/11/28 01:44:25 test: 516000 messages published
2024/11/28 01:44:26 test: 517000 messages published
2024/11/28 01:44:26 test: 518000 messages published
2024/11/28 01:44:26 test: 519000 messages published
2024/11/28 01:44:26 test: 520000 messages published
2024/11/28 01:44:26 test: 521000 messages published
2024/11/28 01:44:26 test: 522000 messages published
2024/11/28 01:44:26 test: 523000 messages published
2024/11/28 01:44:27 test: 524000 messages published
ubuntu@ece573:~/ece573-prj06$ kubectl logs -l app=ece573-prj06-consumer
2024/11/28 01:44:23 test: received 8000 messages, last (0.255230)
2024/11/28 01:44:23 test: received 9000 messages, last (0.694018)
2024/11/28 01:44:24 test: received 10000 messages, last (0.681253)
2024/11/28 01:44:25 test: received 11000 messages, last (0.850199)
2024/11/28 01:44:25 test: received 12000 messages, last (0.368187)
2024/11/28 01:44:26 test: received 13000 messages, last (0.408403)
2024/11/28 01:44:27 test: received 14000 messages, last (0.818107)
2024/11/28 01:44:27 test: received 15000 messages, last (0.852005)
2024/11/28 01:44:28 test: received 16000 messages, last (0.703784)
2024/11/28 01:44:28 test: received 17000 messages, last (0.139423)
ubuntu@ece573:~/ece573-prj06$
```

Ln