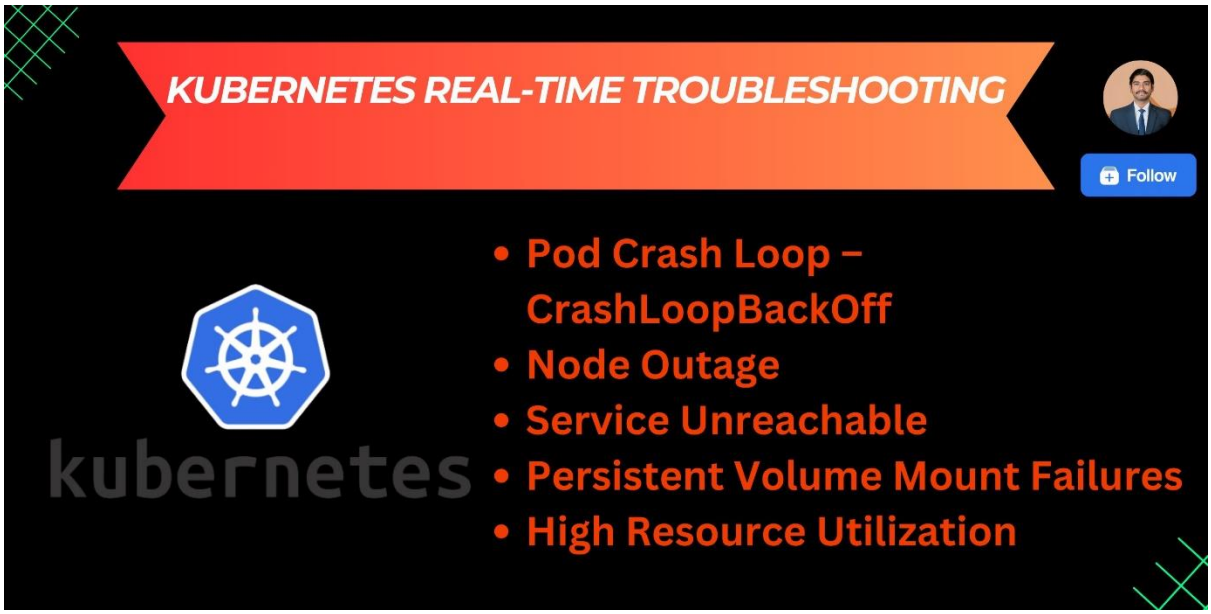
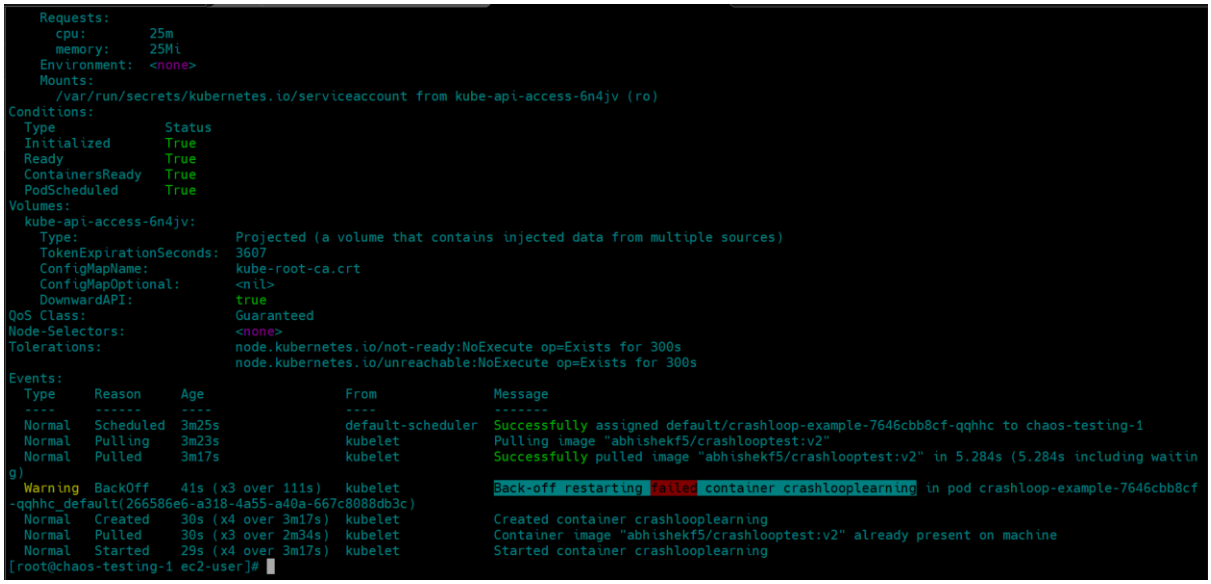
Kubernetes Real-Time Troubleshooting

Introduction 🌐

Welcome to the world of Kubernetes troubleshooting, where every challenge is an opportunity to sharpen your skills and emerge victorious. Join us as we embark on a journey through common real-time scenarios, unraveling mysteries, and uncovering solutions along the way.



Scenario 1: Pod Crash Loop – CrashLoopBackOff



```
    Requests:
      cpu:        25m
      memory:     25Mi
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-6n4jv (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  kube-api-access-6n4jv:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   Guaranteed
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  3m25s              default-scheduler  Successfully assigned default/crashloop-example-7646cbb8cf-qqhhc to chaos-testing-1
  Normal   Pulling    3m23s              kubelet            Pulling image "abhishekf5/crashlooptest:v2"
  Normal   Pulled     3m17s              kubelet            Successfully pulled image "abhishekf5/crashlooptest:v2" in 5.284s (5.284s including waitin
g)
  Warning  BackOff    41s (x3 over 111s)   kubelet            Back-off restarting failed container crashlooplearning in pod crashloop-example-7646cbb8cf
-qqhhc_default(266586e6-a318-4a55-a40a-667c8088db3c)
  Normal   Created    30s (x4 over 3m17s)  kubelet            Created container crashlooplearning
  Normal   Pulled     30s (x3 over 2m34s)  kubelet            Container image "abhishekf5/crashlooptest:v2" already present on machine
  Normal   Started    29s (x4 over 3m17s)  kubelet            Started container crashlooplearning
[root@chaos-testing-1 ec2-user]#
```

FOLLOW – Prasad Suman Mohan (for more updates)

Symptoms: Your pod keeps crashing and restarting repeatedly, disrupting service availability.

Diagnosis: Check the pod logs (`kubectl logs <pod_name>`) to identify the cause of the crash. Look for error messages or exceptions indicating issues with application startup or runtime.

```
Every 2.0s: kubectl get pods                                chaos-testing-1: Sun May  5 13:03:12 2024

NAME                                READY   STATUS             RESTARTS      AGE
crashloop-example-7646cbb8cf-qqhhc  0/1     CrashLoopBackOff   4 (24s ago)   5m33s
```

Solution:

1. Update the pod's YAML configuration to increase resource limits if the crash is due to resource constraints.
2. Review the application code and dependencies for any bugs or issues causing crashes.
3. Check for misconfigured environment variables or volumes that might be causing startup failures.
4. Monitor system metrics to identify any underlying issues with the node's health or resource utilization.

Scenario 2: Node Outage

```
  Mounts:
    /bitnami/apache from apache-data (rw)
    /bitnami/joomla from joomla-data (rw)
    /var/run/secrets/kubernetes.io/serviceaccount from default-token-j6zrx (ro)
nditions:
 Type           Status
 PodScheduled   False
lumes:
 joomla-data:
   Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same names
ace)
   ClaimName:  ornery-lemur-joomla-joomla
   ReadOnly:   false
 apache-data:
   Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same names
ace)
   ClaimName:  ornery-lemur-joomla-apache
   ReadOnly:   false
 default-token-j6zrx:
   Type:        Secret (a volume populated by a Secret)
   SecretName:  default-token-j6zrx
   Optional:    false
oS Class:       Burstable
ode-Selectors:  <none>
lerations:      node.alpha.kubernetes.io/notReady:NoExecute for 300s
                node.alpha.kubernetes.io/unreachable:NoExecute for 300s
vents:
 Type     Reason            Age                  From               Message
 ----     ------            ----                 ----               -------
 Warning  FailedScheduling  15h (x277 over 17h)  default-scheduler  No nodes are available tha
 match all of the predicates: Insufficient cpu (1).
 Warning  FailedScheduling  3m (x71 over 23m)    default-scheduler  No nodes are available tha
 match all of the predicates: Insufficient cpu (1).
```

Symptoms: Pods on a specific node become unavailable, leading to service disruption.

Diagnosis: Use `kubectl describe node <node_name>` to gather information about the node's status, including conditions, capacity, and events. Look for any indications of node failures or network issues.

Solution:

1. If the node is unreachable, check the underlying infrastructure (e.g., cloud provider console or on-premises hardware) for connectivity issues.
2. If the node is experiencing resource exhaustion, consider scaling up the node or redistributing workloads to other nodes.
3. Use Kubernetes node maintenance features to gracefully drain and evacuate pods from the affected node before performing maintenance tasks or node replacement.

Scenario 3: Service Unreachable

Symptoms: Users report being unable to access a Kubernetes service.

Diagnosis: Verify the service's endpoints (`kubectl get endpoints <service_name>`) to ensure they are correctly associated with healthy pods. Check for any network policies or firewall rules that might be blocking traffic.

```
[root@chaos-testing-1 02-CrashLoopBackOff]# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   20m
[root@chaos-testing-1 02-CrashLoopBackOff]# kubectl get svc
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1    <none>        443/TCP        21m
nginx-service   NodePort    10.99.2.30   <none>        80:30001/TCP   4s
[root@chaos-testing-1 02-CrashLoopBackOff]# kubectl get endpoints nginx-service
NAME            ENDPOINTS                                   AGE
nginx-service   10.17.0.10:80,10.17.0.11:80,10.17.0.9:80   15s
[root@chaos-testing-1 02-CrashLoopBackOff]#
```

Solution:

1. If the service endpoints are incorrect or missing pods, investigate the underlying pod deployment or replica set for issues.
2. Review network policies and firewall rules to ensure they allow traffic to and from the service's pods.
3. Use `kubectl describe service <service_name>` to identify any misconfigurations or issues with the service's configuration.

Scenario 4: Persistent Volume Mount Failure

```
IP:
Controllers:    StatefulSet/web
Containers:
  nginx:
    Container ID:
    Image:              gcr.io/google_containers/nginx-slim:0.8
    Image ID:
    Port:               80/TCP
    State:              Waiting
      Reason:           ContainerCreating
    Ready:              False
    Restart Count:      0
    Volume Mounts:
      /usr/share/nginx/html from www (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-gxlqv (ro)
    Environment Variables:      <none>
Conditions:
  Type          Status
  Initialized   True
  Ready         False
  PodScheduled  True
Volumes:
  www:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  www-web-0
    ReadOnly:   false
  default-token-gxlqv:
    Type:       Secret (a volume populated by a Secret)
    SecretName: default-token-gxlqv
QoS Class:      BestEffort
Tolerations:    <none>
Events:
  FirstSeen     LastSeen        Count   From                                                    SubObjectPath   Type            Reason          Message
  ---------     --------        -----   ----                                                    -------------   --------        ------          -------
  1h            1m              31      {kubelet ip-[redacted]-west-2.compute.internal}                         Warning         FailedMount     Unable to mount volumes for pod "web-0_def
ault(63fdad58-7675-11e7-a96e-0698361089de)": timeout expired waiting for volumes to attach/mount for pod "default"/"web-0". list of unattached/unmounted volumes=[www]
  1h            1m              31      {kubelet ip-[redacted]s-west-2.compute.internal}                        Warning         FailedSync      Error syncing pod, skipping: timeout expir
ed waiting for volumes to attach/mount for pod "default"/"web-0". list of unattached/unmounted volumes=[www]
```

Symptoms: Pods fail to start or encounter errors due to issues with mounting persistent volumes.

Diagnosis: Examine the pod's events (`kubectl describe pod <pod_name>`) for mount-related errors. Check the persistent volume (PV) and persistent volume claim (PVC) statuses for any issues.

Solution:

1. Ensure the PV and PVC are correctly provisioned and bound, with the appropriate access modes and storage classes.
2. Verify that the storage backend (e.g., NFS server, cloud storage) is accessible and functioning correctly.
3. Check for any permissions or authentication issues that might prevent the pod from mounting the volume.
4. Use `kubectl exec` to troubleshoot inside the pod and diagnose any filesystem or mount-related issues.

Scenario 5: High Resource Utilization

Symptoms: Nodes or pods experience high resource utilization, leading to performance degradation or instability.

Diagnosis: Monitor system metrics using tools like Prometheus or Kubernetes dashboard to identify nodes or pods with high CPU, memory, or disk usage. Look for any resource contention or spikes in traffic.

```
surender@kube-srv1: ~
surender@kube-srv1:~$
surender@kube-srv1:~$ kubectl get hpa webapp-hpa --watch
NAME         REFERENCE            TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
webapp-hpa   Deployment/webapp    0%/50%    1         10        1          18m
webapp-hpa   Deployment/webapp    21%/50%   1         10        1          19m
webapp-hpa   Deployment/webapp    249%/50%  1         10        1          19m
webapp-hpa   Deployment/webapp    167%/50%  1         10        4          19m
webapp-hpa   Deployment/webapp    70%/50%   1         10        5          20m
webapp-hpa   Deployment/webapp    57%/50%   1         10        6          20m
webapp-hpa   Deployment/webapp    56%/50%   1         10        6          20m
webapp-hpa   Deployment/webapp    51%/50%   1         10        7          20m
webapp-hpa   Deployment/webapp    48%/50%   1         10        7          21m
webapp-hpa   Deployment/webapp    49%/50%   1         10        7          21m
                                  ↑ CPU utilization is now under 50%        ↑ Number of Pods are increased to 7

surender@kube-srv1:~$
surender@kube-srv1:~$ kubectl run -it load-generator --rm --image=busybox -- /bin/sh -c "while sleep 0.01;
 do wget -qO - http://webapp-svc; done" ← Running the load-generator Pod
If you don't see a command prompt, try pressing enter.
DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!D
ONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DO
NE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DON
E!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE
!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!
DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!D
ONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DO
NE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DONE!DON
```

Solution:

1. Scale up the affected pods or nodes to handle increased workload demands.
2. Optimize resource requests and limits for pods to ensure efficient resource allocation.
3. Identify and optimize resource-intensive applications or workloads to reduce resource consumption.
4. Implement autoscaling for pods or nodes to dynamically adjust resource allocation based on workload demands.

In the up-coming parts, we will discussion on more troubleshooting steps for the different Kubernetes based scenarios. So, stay tuned for the and follow @Prasad Suman Mohan for more such posts.