Kubernetes: Operational Issues & Troubleshooting

The Common Errors:

1) Line ending issues: As a lot of developers will be working on different operating systems like Windows and Linux. The developers might end the lines of the code with different endings. The Windows code files end with CLF and the Linux files end with LF.
2) You need to check the line endings. You can check this on the VS code editor in the right-hand bottom.
3) Git Repos has the same issue. If the developers push their changes, the git will automatically try adding the line endings like CLF and LF to the shell script, config, and YMAL files. You need to create a .gitattributes file and exclude them from git modifications.
4) *.sh = text eol=lf
5) *.config = text eol=lf

You can use the above file and the file structure to avoid these kinds of issues. Still, it will solve only 80% of issues. You need to go through the file manually and make changes. The developers usually push the sh files and config files into the docker containers. But when you are trying to execute them you will get an error like the file doesn't exist or the file is not found.

Troubleshooting:

1) Deployments
2) Pods

You need to check both of the events and the status of deployments and pods. These will help you resolve the issues in the deployments in the Kubernetes.

Kubectl get deploy [deployment_name]→ This to help you in checking the deployments. It might be showing the deployment is created but the pods are not getting created.

Kubectl describe deploy [deployment_name] → This will show the description of the deployments.

Here, you need to check the status and the events. You will understand the issue.

If the deployment is trying to run the services but the pods are not getting up, then you need to check the status and the events of the pods.

Kubectl get pods → You can understand the issue here. If the application is crashloopback, then the application and trying to restart again and again. This is a code issue and the developer needs to look into it.

Amar Gajula

You can check the logs by kubectl logs <pod_name>

ErrImagepull:

You need to follow the same events and status for both deploy and the pods.

Once you understand the status of the deployment is pending, then the issues lie in the pods/you can check the logs.

Kubectl get pods → If you see an issue like errimagepull or error, then you need to check the logs of the particular pod. It is an issue with the image is not being specified or the image being unavailable in the container registry. Most of the developers will be referring to the custom docker images in the private registries, the developer might have missed the name of the image or the wrong version might have been mentioned.

Pod Restarting the issue:

Once the deployment creates successfully, then you notice the pods are getting restarted continuously.

First, you need to check the events and status of the deployment using

Kubectl describe deploy [Deployment_name]

Once you see the desired state is fulfilled from the status and you need to check events. The events show the pods are getting crashed and restarting continuously.

You need to check the status and events of the pods:

Kubectl get pods → It will show you the restart count and other details.

You need to check the logs of the pod → kubectl describe po <pod_name>

It might be an intermediate latency and intermediate network issue. The configuration of the network might be wrong or the endpoints, ports, and also the DNS.

You need to check and change the configuration and you can run the docker containers locally to check whether everything is up and running.

Mounts: Secrets and Configmaps issues:

Whichever the issue, you need to check the status and the events of the deployments and the pods.

Once the check deployment status and the events, you can understand the pods are struggling to get created or they are in the pending state.

You need to check the logs of the pods → kubectl describe po <pod_name>

Here, you will see an error message like this → FailedMount – MountVolue.setup failed for volume config map.

You can check the available configmaps: kubectl get cm → It will show you there is no configmaps found.

Most probably, the developer might have missed the secrets included in the deployment manifest. It is a simple issue. You just need to mention the same and redeploy your application.

Scheduling:

Once the check deployment status and the events, you can understand the pods are struggling to get created or they are in the pending state.

You need to check the logs of the pods → kubectl describe po <pod_name>

Here, you will see an error message like this → 0/1 nodes are available – insufficient memory.

This issue might occur in different scenarios:

- You might have mentioned the pods needs to be created in a specific node/specific label. Turns out, the node selector is not available.
- Another one is caused by the resource.yml file. You might have mentioned the specific memory or the cpu to the pods and the resources are not satisfied. You need to increase the memory and the cpu values to satisfy the deployment.
- You might be facing high traffic and the pods are getting used up and your auto-scaler is creating multiple nodes and eventually, you run out of space in the VM.

You need to check the manifest.yml file for the resource restriction:

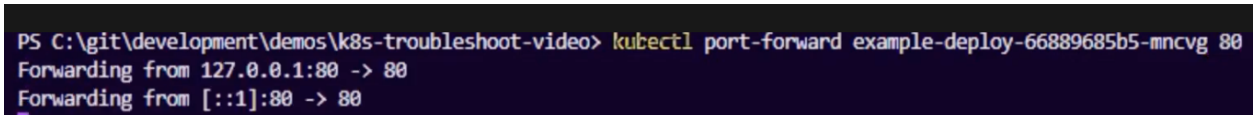
Kubectl get deploy <deployment_name> -o YAML

You need to navigate to the resource block and check whether everything is proper or some memory or the cpu units might have got high values. You need to change that to normal and apply the deployment again.

Ports:

The pods are up but the traffic is not getting assigned to the pods: Usually, it is an issue with the ports that are placed on the pods. You can that by → kubectl describe pods <name of the pod>

Sometimes, the developers write a code and mention that the pods will be exposing different ports and assign the different ports to the pods.



When you hit 127.0.0.1:80 in the browser. The respected page should load.

Traffic:

You performed all the troubleshooting above as mentioned but you are still not able to get the traffic on your microservices or the pods. You need to check the services.

Kubectl get svc

Kubectl describe svc <name of the service>

```
PS C:\git\development\demos\k8s-troubleshoot-video> kubectl get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
example-service   ClusterIP   10.97.177.209   <none>        80/TCP    43s
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP   23d
PS C:\git\development\demos\k8s-troubleshoot-video> kubectl describe svc example-service
Name:              example-service
Namespace:         default
Labels:            <none>
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"name":"example-service","namespace":"default"},"s
Selector:          app=example-app-mistate
Type:              ClusterIP
IP:                10.97.177.209
Port:              <unset>  80/TCP
TargetPort:        5000/TCP
Endpoints:         <none>
Session Affinity:  None
Events:            <none>
PS C:\git\development\demos\k8s-troubleshoot-video>
```

You see the endpoints is not mentioned.

Whenever we create the service, we will mention the selector in deployment and also in the service. Here, you can see the deployment selector is app=example-app but in the service, the selector is mentioned as app=example-app-misstate. It should same as the deployment selector.

```
Selector:          app=example-app-mistate
Type:              ClusterIP
IP:                10.97.177.209
Port:              <unset>  80/TCP
TargetPort:        5000/TCP
Endpoints:         <none>
Session Affinity:  None
Events:            <none>
PS C:\git\development\demos\k8s-troubleshoot-video> kubectl get deploy
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
example-deploy   2/2     2            2           29m
PS C:\git\development\demos\k8s-troubleshoot-video> kubectl get deploy --show-labels
NAME             READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
example-deploy   2/2     2            2           29m   app=example-app
```

Service Troubleshoot:

You see the pods are up and running. The service and the deployment selectors are matching but still, the service for the microservices is refused.

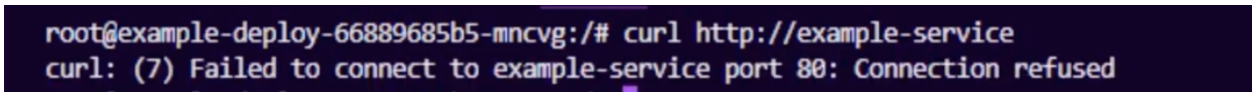
Here, you need to check the status and the events of both deployment and pods.

Here, we can use the curl and request a service inside the container.

Kubectl exec -it <name of the pod> /bash

Once you are inside the container, run the command as mentioned below.

Curl http://name_of_the_service



In this case, you need to check the service description and the target port.

Kubectl describe svc <name of the service> -o YAML

Here, you can notice the target port is mentioned wrong as it should be 80 but might have mentioned something different.

Note: Always run the docker containers locally before having them running on the prod or test environments.