Debug and log your Kubernetes application

Introduction

Kubernetes enables scalable and resilient deployment of containerized applications. However, diagnosing problems in a distributed system can be difficult. Debugging tools such as logging, event inspection, and command-line troubleshooting are critical to ensuring application health.

This project demonstrates how to deploy a sample Guestbook application and troubleshoot it using Kubernetes-native debug and logging methods.

Objectives

Learn how to deploy an application Imperatively

Expose the app through a Kubernetes Service

Check Pod status and cluster events

View application logs

Debug problematic Pods

Technologies Used

- Kubernetes cluster
- Kubectl CLI
- Docker container image: ibmcom/guestbook:v1

Implementation Steps

1. Deploy the Guestbook Application

Creates a Deployment object and pulls the Guestbook image into Pods.

kubectl create deployment guestbook --image=ibmcom/guestbook:v1

```
controlplane:~$ kubectl create deployment guestbook --image=ibmcom/guestbook:v1
deployment.apps/guestbook created
controlplane:~$
controlplane:~$
```

To get basic information about your pods you can use this simple command:

```
controlplane:~$ kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
guestbook-86cf798756-75p5h   1/1     Running   0          28s
controlplane:~$
controlplane:~$
```

2. Describe Pod Details

But you can get much more information if you describe a specific pod, like this:

kubectl describe pod <pod name>

```
controlplane:~$ kubectl describe pod guestbook-86cf798756-zfgsk
Name:             guestbook-86cf798756-zfgsk
Namespace:        default
Priority:         0
Service Account:  default
Node:             node01/172.30.2.2
Start Time:       Wed, 03 Dec 2025 10:32:09 +0000
Labels:           app=guestbook
                  pod-template-hash=86cf798756
Annotations:      cni.projectcalico.org/containerID: 64eee9a92062638813da87143e15d8e09f50dc046f89851b0e5758cb30da5135
                  cni.projectcalico.org/podIP: 192.168.1.4/32
                  cni.projectcalico.org/podIPs: 192.168.1.4/32
Status:           Running
IP:               192.168.1.4
IPs:
  IP:           192.168.1.4
Controlled By:  ReplicaSet/guestbook-86cf798756
Containers:
  guestbook:
    Container ID:   containerd://6ceb1fa390b54ab5018db4642167a0cc0aebd5da377a24dc5e50203a63523229
    Image:          ibmcom/guestbook:v1
    Image ID:       docker.io/ibmcom/guestbook@sha256:8e99727133b5c3b40c05bb313dd4591774f0a23415e3d44eac0bc883051de75d
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Wed, 03 Dec 2025 10:32:14 +0000
```

From above you can see the configuration information about the containers and the pod(labels, resource requirements, etc.), and the status information about the containers and pod (state, readiness, restart count, events, etc)

3. To get the events list of your pod, use the command:

kubectl get events [--namespace=default]

```
72s         Normal    Scheduled                pod/guestbook-86cf798756-zfgsk    Successfully assigned default/guestbook-86cf798756-
zfgsk to node01
71s         Normal    Pulling                  pod/guestbook-86cf798756-zfgsk    Pulling image "ibmcom/guestbook:v1"
67s         Normal    Pulled                   pod/guestbook-86cf798756-zfgsk    Successfully pulled image "ibmcom/guestbook:v1" in
3.692s (3.692s including waiting). Image size: 7406761 bytes.
67s         Normal    Created                  pod/guestbook-86cf798756-zfgsk    Created container: guestbook
67s         Normal    Started                  pod/guestbook-86cf798756-zfgsk    Started container guestbook
72s         Normal    SuccessfulCreate         replicaset/guestbook-86cf798756   Created pod: guestbook-86cf798756-zfgsk
72s         Normal    ScalingReplicaSet        deployment/guestbook              Scaled up replica set guestbook-86cf798756 from 0 t
o 1
15d         Normal    NodeHasSufficientMemory  node/node01                       Node node01 status is now: NodeHasSufficientMemory
15d         Normal    NodeHasNoDiskPressure    node/node01                       Node node01 status is now: NodeHasNoDiskPressure
15d         Normal    NodeHasSufficientPID     node/node01                       Node node01 status is now: NodeHasSufficientPID
15d         Normal    NodeAllocatableEnforced  node/node01                       Updated Node Allocatable limit across pods
15d         Normal    NodeReady                node/node01                       Node node01 status is now: NodeReady
15d         Normal    RegisteredNode           node/node01                       Node node01 event: Registered Node node01 in Contro
ller
15d         Normal    Starting                 node/node01
4m44s       Normal    Starting                 node/node01                       Starting kubelet.
```

4. Get Application Logs

Application and system logs can help you gain a better understanding of what happened inside your cluster. You can get logs for a specific pod and if the pod has multiple containers, you can specify which container you want.

Too see your logs, you run this simple command:

kubectl logs <your-pod-name>

```
controlplane:~$ kubectl logs guestbook-86cf798756-zfgsk
[negroni] listening on :3000
controlplane:~$
```

Previous Logs: You can always ask for previous logs with the --previous flag

5. Use a shell inside a running container

You can use kubectl exec to access the shell running in the container and figure out where the process is in your troublesome container, which then gives you a more comfortable way to debug your container.

In order to do so we will need to create new pod with /bin/bash:

kubectl create -f https://kubernetes.io/examples/application/shell-demo.yaml

```
controlplane:~$ kubectl create -f https://kubernetes.io/examples/application/shell-demo.yaml
pod/shell-demo created
controlplane:~$
controlplane:~$
```

Get a shell to the container:

kubectl exec -it shell-demo -- /bin/bash

```
controlplane:~$ kubectl exec -it shell-demo -- /bin/bash
root@node01:/#
```

Now list the root directory:

```
controlplane:~$ kubectl exec -it shell-demo -- /bin/bash
root@node01:/# ls
bin   dev                docker-entrypoint.sh  home  lib64  mnt  proc  run   srv  tmp  var
boot  docker-entrypoint.d  etc                  lib   media  opt  root  sbin  sys  usr
root@node01:/#
```

6. Debug your service

An issue that comes up frequently for new installations of Kubernetes is that the service aren't working properly, so you run your deployment and create a service but still don't get any response. In this section, we'll go over some commands that might help you figure out what's not working.

a. **If the service you are looking for doesn't exist, you can create it by using this command:**

   kubectl expose deployment <deployment-name> --type="NodePort" --port=3000

```
controlplane:~$ kubectl expose deployment guestbook --type="NodePort" --port=3000
service/guestbook exposed
controlplane:~$
controlplane:~$
```

b. **To see all your services, you can use a simple command like this one where we can see all pods:**

   kubectl get svc

```
controlplane:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
guestbook    NodePort    10.98.127.150   <none>        3000:32515/TCP   25s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          15d
controlplane:~$
```

c. **Lets try to get more information about this service:**

kubectl describe service <your service name>

```
controlplane:~$ kubectl describe service guestbook
Name:                     guestbook
Namespace:                default
Labels:                   app=guestbook
Annotations:              <none>
Selector:                 app=guestbook
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.98.127.150
IPs:                      10.98.127.150
Port:                     <unset>  3000/TCP
TargetPort:               3000/TCP
NodePort:                 <unset>  32515/TCP
Endpoints:                192.168.1.4:3000
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
```

Check if you used the right NodePort to access the container and that you have endpoints.

d. **You can also get your information in json format:**

kubectl get service <your-service-name> -o json

```
controlplane:~$ kubectl get svc guestbook -o json
{
    "apiVersion": "v1",
    "kind": "Service",
    "metadata": {
        "creationTimestamp": "2025-12-03T10:37:58Z",
        "labels": {
            "app": "guestbook"
        },
        "name": "guestbook",
        "namespace": "default",
        "resourceVersion": "3462",
        "uid": "f6f19dfd-0cfb-4782-948d-84e44685a0b8"
    },
    "spec": {
        "clusterIP": "10.98.127.150",
        "clusterIPs": [
            "10.98.127.150"
        ],
        "externalTrafficPolicy": "Cluster",
        "internalTrafficPolicy": "Cluster",
        "ipFamilies": [
            "IPv4"
        ],
        "ipFamilyPolicy": "SingleStack",
        "ports": [
            {
                "nodePort": 32515,
                "port": 3000,
                "protocol": "TCP",
                "targetPort": 3000
            }
        ],
        "selector": {
            "app": "guestbook"
        },
        "sessionAffinity": "None",
        "type": "NodePort"
    },
    "status": {
        "loadBalancer": {}
    }
}
```

7. Check DNS

Some of the network problems could be caused by DNS configurations or errors. So first you'll need to check if the DNS works correctly.

1. **Use the get pods command to get your pod name:**

   kubectl get pods

```
controlplane:~$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
guestbook-86cf798756-zfgsk    1/1     Running   0          8m13s
shell-demo                    1/1     Running   0          4m55s
controlplane:~$
```

2. Check the pod DNS with the following command:

kubectl exec -it <your-pod-name> -- nslookup kubernetes.default

```
controlplane:~$ kubectl exec -it guestbook-86cf798756-zfgsk -- nslookup kubernetes.default
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      kubernetes.default
Address 1: 10.96.0.1 kubernetes.default.svc.cluster.local
controlplane:~$
```

3. Go inside the resolve.conf file to see if the parameters are ok:

kubectl exec <your-pod-name> cat /etc/resolv.conf

```
controlplane:~$ kubectl exec guestbook-86cf798756-zfgsk -- cat /etc/resolv.conf
search default.svc.cluster.local svc.cluster.local cluster.local
nameserver 10.96.0.10
options ndots:5
controlplane:~$
```

8. Network Policies

The NetworkPolicy defines how pods are allowed to communicate with each other and with other network endpoints. NetworkPolicy uses the labels to manage the traffic between pods.

If you are not able to communicate with pods, you might want check your network policies to see if this pod doesnt allowed to get any requests.

By default, pods are not isolated and they accept traffic. But once you have NetworkPolicy selected a specific pod, you'll reject any communication with unauthorized connections.