Troubleshooting Kubernetes Cluster Node Failures

22m 48s left

100% **Open Cloud Environment**
Setup completed
Average setup time: 2m 27s

100% **Open Code Environment**
Setup completed

Credentials

Account ID
51131186 Copy

Username
student Copy

Password
Ca1_Ey10[Copy

Region
US West 2 Copy

PEM Download PPK Download

Bastion Public Ip
54.212.78 Copy

Cluster SSH
ssh ubunt

Introduction

Nodes can fail in different ways, some may be recoverable while others are not. This lab step focuses on first diagnosing node failures, and then recovering a node that entered into a failure mode but can still be connected to by SSH.

In the case of hardware failures or other unrecoverable errors in a cloud environment, the appropriate remedy is to add a new node and delete the old one. The method for adding a new node varies depending on how the cluster was originally created. If kubeadm is used, then you can use the `join` command with the appropriate token to join the cluster. With the CloudFormation template that this lab uses, you can terminate the failed instance, and a new one will be created automatically when health checks fail. Nodes can be removed from the cluster by using `kubectl delete node <name_of_node>`.

Instructions

1. List all of the nodes in the cluster using `wide` output to include additional columns in the output:

Copy code

```
kubectl get nodes –o wide
```

All of the nodes have a **STATUS** of **Ready**. The **Ready** status means a node is healthy and ready to accept pods. You will force a worker node into a different status to experience a type of failure. Before moving on, notice the **OS-IMAGE** of the nodes is **Ubuntu**. This is the same OS used in Kubernetes certification exams.

2. Connect to one of the worker nodes using SSH:

Support

Copy code

Lab Steps

Copy code

```
sudo systemctl status kubelet
```

```
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
           └─10-kubeadm.conf
   Active: active (running) since 04:54:12 UTC; 6min ago
     Docs: https://kubernetes.io/docs/home/
 Main PID: 2612 (kubelet)
    Tasks: 21 (limit: 4617)
   CGroup: /system.slice/kubelet.service
           └─2612 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf

Nov 01 04:55:08 ip-10-0-0-10 kubelet[2612]: E1101 04:55:08.673251    2612 kuberuntime_manager.go:
Nov 01 04:55:08 ip-10-0-0-10 kubelet[2612]: E1101 04:55:08.673333    2612 pod_workers.go:951] "Er
Nov 01 04:55:13 ip-10-0-0-10 kubelet[2612]: I1101 04:55:13.290045    2612 scope.go:110] "RemoveCo
Nov 01 04:55:13 ip-10-0-0-10 kubelet[2612]: I1101 04:55:13.298732    2612 scope.go:110] "RemoveCo
Nov 01 04:55:13 ip-10-0-0-10 kubelet[2612]: I1101 04:55:13.305526    2612 scope.go:110] "RemoveCo
Nov 01 04:55:13 ip-10-0-0-10 kubelet[2612]: I1101 04:55:13.312668    2612 scope.go:110] "RemoveCo
Nov 01 04:55:13 ip-10-0-0-10 kubelet[2612]: I1101 04:55:13.320846    2612 scope.go:110] "RemoveCo
Nov 01 04:55:22 ip-10-0-0-10 kubelet[2612]: I1101 04:55:22.067449    2612 scope.go:110] "RemoveCo
Nov 01 04:55:25 ip-10-0-0-10 kubelet[2612]: I1101 04:55:25.438921    2612 csi_plugin.go:99] kuber
Nov 01 04:55:25 ip-10-0-0-10 kubelet[2612]: I1101 04:55:25.438960    2612 csi_plugin.go:112] kube
```

Recall that the kubelet is the primary agent running on nodes that watches for pod specs. Ubuntu 18 is a systemd-based Linux operating system. The tool used to manage services in systemd-based systems is `systemctl`. Common systemctl commands include:

- `status`: Show a status summary of the service including its current state (**active (running)**), the location of the service file (**/lib/systemd/system/kubelet.service**), any drop-in files that help to configure the service (**/etc/systemd/system/kubelet.service.d/10-hostname.conf** and **/etc/systemd/system/kubelet.service.d/10-kubeadm.conf**), the command executed to start the service (**/user/bin/kubelet --bootstrap-kubeconfig**...), and the ten most recent log messages at the bottom of the output. These bits of information can be very helpful when debugging a failed node.
- `start`: Starts a service
- `stop`: Stops a service
- `enable`: Enables a service so that it is automatically started on boot. Note that enable does not automatically start a service until the system is restarted. Therefore, this command is usually followed by `start` to start the service without rebooting. (there is also the `––now` option of enable to both enable and start the service in one command)

4. Press *q* to quit the status output view.

5. Enter the following to view all of the log messages associated with kubelet service:

Copy code

you simulate a new error.

6. Stop the kubelet running on the worker node:

Copy code

```
sudo systemctl stop kubelet
```

You can confirm the service stopped by viewing the `status` output again. Another service you could stop to create a failure would be `containerd`, which is the container runtime used by the nodes in this lab. When diagnosing an outage, you should check the status of both the kubelet and the container runtime.

7. Close the ssh connection to the worker to return to the bastion host's shell:

Copy code

```
exit
```

8. List and watch the status of all the nodes in the cluster:

Copy code

```
watch kubectl get nodes
```

```
NAME                                         STATUS     ROLES
ip-10-0-0-10.us-west-2.compute.internal      NotReady   <none>
ip-10-0-0-100.us-west-2.compute.internal     Ready      control-plane
ip-10-0-0-11.us-west-2.compute.internal      Ready      <none>
```

Notice that the first worker node has a **STATUS** of **NotReady**. Because you created the failure, you know exactly what is wrong. But if it was an unexpected failure, you would issue the `systemctl status` and `journalctl` commands to diagnose the problem.

Press *ctrl+c* to stop watching the nodes.

9. Start the kubelet service on the failed worker node:

Copy code

```
ssh worker1 'sudo systemctl start kubelet'
```

```
watch kubectl get nodes
```

All of the nodes should be **Ready** within 40 seconds of the kubelet starting. Nodes can also fail to accept pod requests due to resource pressure. For example, if a node is out of disk space or running low on memory. You will simulate this situation in the following instructions.

Press *ctrl+c* to stop watching the nodes.

11. Restart the worker node's kubelet with an additional option to make it detect memory pressure:

Copy code

```
ssh worker1 'sudo sed -i "s%\(/usr/bin/kubelet\) %\1 --eviction-
```

The commands add the `--eviction-hard` option to the command that starts the kubelet in `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf`. The `--eviction-hard` sets the threshold for when Kubernetes starts to detect memory pressure and is forced to start evicting pods to try to reclaim memory. Its value is 100 mebibytes (`Mi`) by default. The node only has 3.8Gi of memory in total. Setting the threshold to 3.75Gi means Kubernetes will always detect memory pressure.

12. List the status of all the nodes in the cluster:

Copy code

```
kubectl get nodes
```

All of the nodes still have a **STATUS** of **Ready**. However, if you were scheduling pods, you may notice that one worker is not accepting any new pods and is evicting existing pods. This requires more details to diagnose than the `get` command provides.

13. Describe the worker node with the increased hard eviction threshold:

Copy code

```
kubectl describe node ip-10-0-0-10 | more
```

Focus on the **Conditions** section near the top of the output:

```
NetworkUnavailable   False
MemoryPressure       True
DiskPressure         False
PIDPressure          False
Ready                True
```

You will notice that the node is reporting memory pressure (**MemoryPressure**). The node is still **Ready** because it is listening for pod requests, although it typically would not accept any new requests because it does not have enough memory available. If the memory shortage was due to pods running on the node, Kubernetes could remedy the situation itself by evicting the pods. However, if the node had processes running outside of Kubernetes that were consuming the memory, you would need to connect to the node and stop processes to free up memory for Kubernetes. You would take similar action if the node was reporting disk space is running low (**DiskPressure**), or there are too many processes running on the node (**PIDPressure**). You may want to `drain` or `cordon` the node to prevent pods from being scheduled on the node until you are sure the issue is resolved. When you are ready, you would `uncordon` the node to allow pods to be scheduled on the node again. The **NetworkUnavailable** condition indicates if there is an issue with the node's network connection.

14. Press *spacebar* until you have paged through all of the `describe` output, and the shell prompt is returned to you.

At the end of the output in the **Events** section, which includes the most recent events at the end, you will see the following two events related to the memory pressure:

```
Normal   NodeHasInsufficientMemory
Warning  EvictionThresholdMet
```

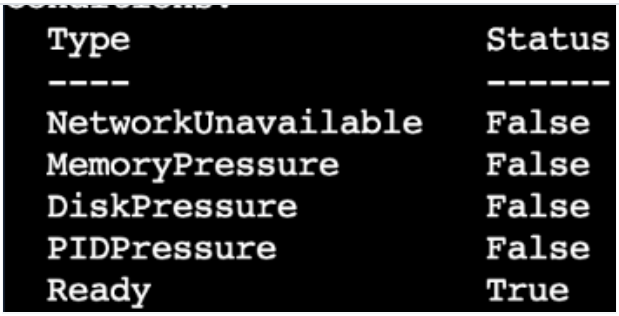
15. Restart the worker node's kubelet without the hard eviction limit option to repair the node:

Copy code

```
1 ssh worker1 'sudo sed -i "s%--eviction-hard=memory.available<3.7
```

16. Confirm that the node no longer reports memory pressure:

Copy code



Summary

In this lab step, you learned how to detect and resolve several node-level issues that can cause failures or have an undesirable impact on the performance of your Kubernetes cluster.

VALIDATION CHECKS

1 **Checks** Start check

Triggered MemoryPressure Condition on Node

Triggered the MemoryPressure condition on worker node

Kubernetes

Did you like this step?

End Lab Back Start check

COMMUNITY

Join Discord Channel

HELP

Help Center