Guided Exercise: Troubleshooting Using the Cluster Monitoring Stack

In this exercise you will identify a potential application problem using the cluster monitoring stack.

**Outcomes**

You should be able to use Grafana to identify a deployment that consumes excessive amounts of CPU and memory.

To perform this exercise, ensure you have access to:

* A running OpenShift cluster.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and deploys an application with a potential problem.

**[student@workstation ~]$ lab monitor-troubleshoot start**

**Procedure 9.2. Instructions**

1. Use Grafana to display cluster resource usage information within the OpenShift web console.
   1. Access the web console URL at https://console-openshift-console.apps.ocp4.example.com using Firefox. Accept the TLS certificates, if prompted, and then log in with htpasswd\_provider using admin as the username and redhat as the password.
   2. Navigate to **Monitoring** → **Dashboards**.
   3. Select the Kubernetes/Compute Resources/Cluster dashboard.
   4. Scroll to the **CPU Quota** section, and then click the **CPU Usage** column header. You might have to click the column header more than once. Click until it turns blue with an arrow pointing down to indicate that rows are sorted by namespaces using the highest to lowest amount of CPU.

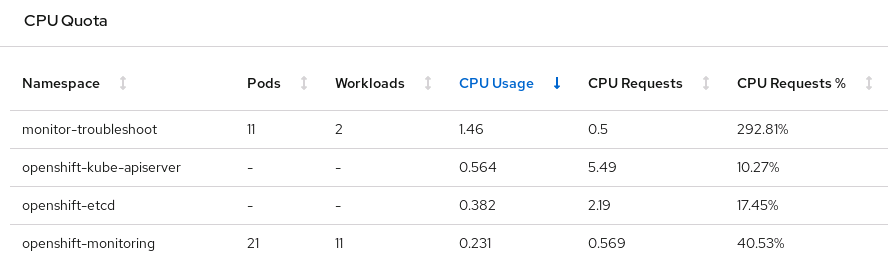


Figure 9.9: Namespaces with the Highest CPU Usage

Expect that the monitor-troubleshoot namespace is listed as one of the top five namespaces using the most amount of CPU resources. Although the monitor-troubleshoot namespace only requests 0.5 CPU resources, it uses considerably more. The result is that the **CPU Requests %** column reports between 100% to 500%. If you do not see the monitor-troubleshoot namespace consuming an elevated amount of resources, then reload the page in your browser.

* 1. Scroll to the **Requests by Namespace** section and then click the **Memory Usage** column header. You might have to click the column header more than once. Click until it turns blue with an arrow pointing down to indicate that rows are sorted by namespaces using the highest to lowest amount of memory.

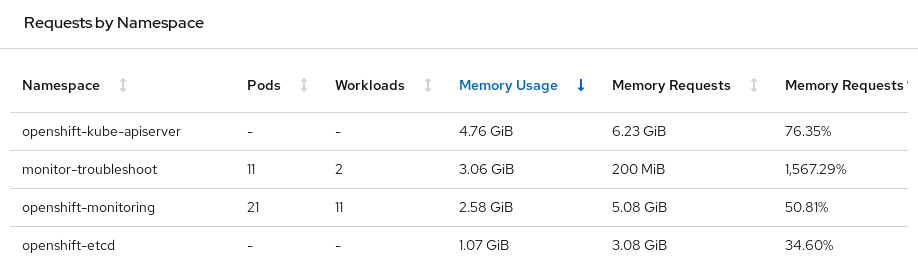


Figure 9.10: Namespaces with the Highest Memory Usage

Expect that the monitor-troubleshoot namespace is listed as one of the top five namespaces using the most memory resources. As with CPU usage, the monitor-troubleshoot namespace uses considerably more memory than the 200 MiB requested by the pods in the namespace.

1. Use Grafana to identify the workload and pod that uses the most cluster resources in the monitor-troubleshoot namespace.
   1. Select the Kubernetes/Compute Resources/Namespace (Workloads) dashboard. Select the monitor-troubleshoot namespace and leave deployment as the workload type.
   2. The **CPU Quota** section shows that the hello deployment requests a total of 0.5 CPU resources and currently uses almost no CPU resources. The python-load deployment does not request or limit CPU resources, but the one pod in the deployment consumes more than three CPU resources.

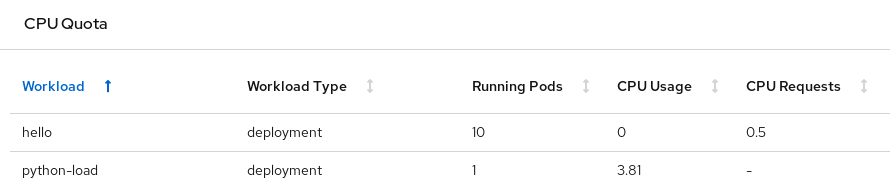


Figure 9.11: Deployment CPU Usage

Although the actual CPU usage in your cluster might be different, it is clear that the python-load deployment uses the most CPU resources.

* 1. The **Memory Usage** graph shows that the python-load deployment has dramatic increases and decreases in memory usage. However, the ten pods in the hello deployment consistently use a total of about 122 MiB of memory.

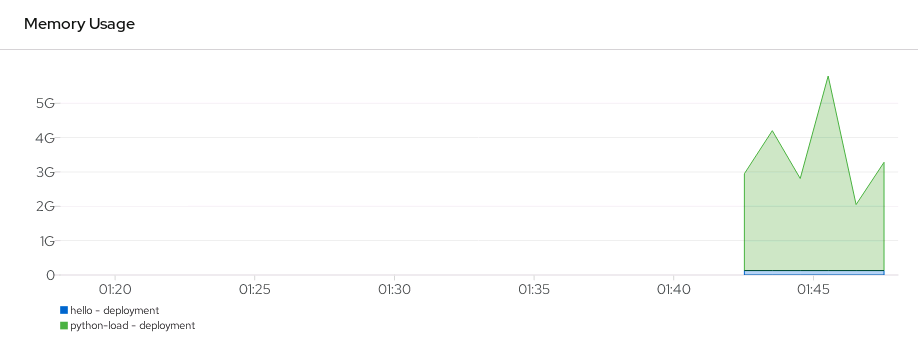


Figure 9.12: Deployment Memory Usage

1. You have identified a deployment that appears to be behaving erratically and might be using more resources than intended. The next step might be to identify the owner of the python-load deployment so that the owner can verify the application.

Minimally, the owner should add either resource requests or limits for both CPU and memory to the python-load deployment. Although a cluster administrator can do this, setting incorrect limits might result in the pod entering a CrashLoopBackOff state.

Similarly, a cluster administrator can implement a quota for the monitor-troubleshoot namespace to limit the total amount of CPU and memory resources used by the project. Because the python-load deployment does not currently specify requests or limits for either CPU or memory, restricting CPU and memory resources with a quota would prevent the python-load pod from running.

**Finish**

On the workstation machine, use the lab command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

**[student@workstation ~]$ lab monitor-troubleshoot finish**