Kubernetes Introspection Tool for NVIDIA GPU Clusters

# Introduction

This document outlines the concept, design, and usage of an application that gathers a wide range of information on a Kubernetes cluster with NVIDIA GPUs. The purpose of this tool is to capture a detailed cluster inventory, including GPU data, package it into a transportable file/data structure, and make this information easily accessible. This tool is compatible with various Kubernetes deployment options (on prem, cloud, OpenShift, Tanzu, etc.), as well as many GPU configurations (vGPU, MIG, etc.).The github repository for this deployment can be found at: [MattFeinberg/K8s-Introspection-Tool (github.com)](https://github.com/MattFeinberg/K8s-Introspection-Tool).

## Assumptions, Constraints, Dependencies

* This deployment assumes that the NVIDIA GPU Operator has been deployed on the cluster.
  + In particular, the device-plugin-daemonset pod must be up and running.
* Certain data pieces are not always reachable, see [Data Gathering Techniques](#_69zo6r20k5vc) for more information.

# Information Gathered Preview

Below is a list of **some** of the information gathered by the introspection tool. Note that this is not a complete list of all the data points collected.

Cluster Level Information:

* Number of Nodes
* Number of Unhealthy Nodes
* Number of GPUs Across All Nodes
* GPU distribution (example: 5 T4s, 2 A30s)
* Kubernetes Distribution

Node Level Information:

* Operating System
* Container Runtime & Version
* Number of GPUs
* CUDA Driver Version

GPU Level Information:

* GPU Name
* GPU Serial Number
* Framebuffer Memory Total Size
* Physical vs Virtual GPU
* MIG Enabled/Disabled (or N/A)

# Design Details

This application gathers data in two main ways:

1. Through the nvidia-smi tool, for most of the GPU related information
2. Through the various existing Golang packages for Kubernetes, for other information

## Program Flow

On some specified time interval (default is 24 hours), the application performs these actions in the following order:

1. Updates Node Level Information
   1. For GPU related information, the application finds the driver-plugin-daemonset pod associated with each node, then uses the programmatic equivalent of kubectl exec to run nvidia-smi once on each node.
2. Update Cluster Level Information

## Data Gathering Techniques

In the case where information is not **directly** accessible from the Kubernetes API or nvidia-smi, some of the data gathering methods performed by this application are not intuitive. It is useful to understand how certain data points are generated to give more context into a cluster inventory, especially in the case of missing or unknown information. Below are selected data points and an explanation of how exactly the application collects them.

* Unhealthy Nodes
  + The application identifies a “healthy” node as a node where nvidia-smi runs successfully and returns the value 0. Any nonzero return value is reported in the cluster inventory, and the node is considered unhealthy.
* Kubernetes Distribution
  + The application determines distribution by first selecting a single node to use for searching, and then using a grep-style on the node’s labels for a predefined list of keywords (distributions). For example, if the word “OpenShift” occurs in any node label, the application identifies the cluster as an OpenShift cluster. If multiple distributions are somehow detected, that is, at least two keywords were found in the node labels, then the application reports that it cannot confirm a distribution, and gives both of the detected distributions as possibilities. If no keywords are found, the application reports “Standard” for a vanilla Kubernetes cluster.
  + As of August 23, 2022, the list of distributions to search for is:
    - Tanzu, OpenShift, K3s, RKE
* Cloud vs. On-Prem
  + In a similar style to the distribution detection method, the application distinguishes cloud clusters from on prem clusters by searching through node labels and annotations. If the word “cloud” appears anywhere in the labels or annotations, the application identifies the cluster as a cloud cluster. Otherwise, it reports it as an on prem cluster.
* VM vs. Bare Metal
  + To distinguish between VM and bare metal nodes, the application looks for the feature.node.kubernetes.io/cpu-cpuid.HYPERVISOR label. If a node has this label as true, it is identified as a VM node, and if false, as a bare metal node.
  + **NOTE:** There have been occurrences where this node label is not present on a node. In this case, unfortunately, we currently have no alternate way to report whether the node is a VM or bare metal. Thus, we cannot always gather this data point.
* Mellanox NICs
  + The application identifies Mellanox NICs on the cluster by first checking if the NVIDIA Network Operator is installed. If so, it runs lspci on the mofed pod and filters the output for devices containing the word “Mellanox.”

# Installation

Before installing, ensure that the NVIDIA GPU Operator has been installed onto your cluster. Additionally, privileges to create and deploy in a new namespace on the cluster must be set. Some distributions may require extra steps to ensure this; for example, tanzu environments require a security policy for new namespaces, like this:

| apiVersion: rbac.authorization.k8s.io/v1  kind: RoleBinding  metadata:  name: psp:vmware-system-privileged:monitoring  namespace: monitoring  roleRef:  apiGroup: rbac.authorization.k8s.io  kind: ClusterRole  name: psp:vmware-system-privileged  subjects:  - kind: Group  apiGroup: rbac.authorization.k8s.io  name: system:serviceaccounts |
| --- |

## Installation Options

The helm chart for this application offers various installation options:

* Data Scrape Rate (Default: 24 hours)
  + Helm chart variable name: rate (integer)
  + Measured in hours
* Deploy Web Interface (Default: true)
  + Helm chart variable name: web (boolean)
* NodePort Value (Default: 30069)
  + Helm chart variable name: nodePort (integer)
* Path To Output File (Default: “inventory.csv”)
  + Helm chart variable name: path (string)
  + This variable must specify a **file**, not a directory. If the specified file does not exist, it will be created. If it already exists, the inventory data will be appended to it.

These options are specified through the helm chart’s values.yaml file:

| web: true  rate: 24  nodePort: 30069  path: "inventory.csv" |
| --- |

To override a default value and specify your own, use --set <variable>=<value> when installing. Be sure to use the correct data type when setting a variable. For example, set path to a string value and web to a boolean value.

## Add Helm Repo

First, retrieve the most recent version of the helm repo:

| helm repo add introspection \  https://mattfeinberg.github.io/K8s-Introspection-Tool/helm/  helm repo update |
| --- |

## Install Helm Chart

To install the helm chart, run:

| helm install introspec-tool introspection/introspection-chart \  --create-namespace --namespace monitoring |
| --- |

This will use the default installation settings, but you can customize your installation by adding --set <variable>=<value> to the end of the installation command. To view the default settings and customizable variables, see [Installation Options](#_mbnnhmh6ut4q). This command will also deploy the introspection application in the monitoring namespace, and will create the namespace if it does not already exist. There are no restrictions on this, so feel free to deploy in another namespace if desired.

# Usage

The application will produce a JSON file capturing the cluster inventory, and (if enabled) display the information on a webpage. Below are directions for accessing the data file, web page, and reading any error messages produced by the application.

## Accessing Data File

By default, the application will write to the container’s home directory in a file named inventory.csv. As of August 23, 2022, there is no mechanism to automatically transfer this file anywhere, so you must retrieve it manually. To copy the data file to your local machine, use kubectl cp:

| kubectl cp <namespace>/<pod>:<path to file> <destination path> |
| --- |

The namespace is monitoring by default, and the default path to file is /root/inventory.csv. To find the name of the application’s pod, run:

| kubectl get pods -A | sed -n -e '1p' -e '/introspection/p' |
| --- |

Here is an example output, with the pod name highlighted:

| NAMESPACE NAME READY STATUS RESTARTS AGE  monitoring introspection-tool-7498988895-d8dtg 1/1 Running 0 31s |
| --- |

## Accessing Web Page

To access the webpage, simply navigate to <machine-IP>:<NodePort> on a web browser. If you did not specify a NodePort value at installation, the default is 30069. Since we are using NodePort, you can use the IP address of any of your nodes. Run:

| kubectl get nodes -o wide |
| --- |

And choose a node’s external IP address to use. If a node doesn’t have an external IP, then the web page will work through its internal IP instead.

**Note**: Certain deployment platforms may require additional steps to access the web page (security permissions, firewalls, etc.)

## Error Handling

Various things can go wrong during the data gathering process. To check for and read any error messages, run:

| kubectl -n <namespace> logs <pod> |
| --- |

There are directions to retrieve the pod name in: [Accessing Data File](#_osgchve2etwg).