Hard Drives Monitoring Automation Approach for Kubernetes Container Orchestration System

Shemyakinskaya Anastasia Sergeevna
Peter the Great St.Petersburg
Polytechnic University
Russia, 195251, St.Petersburg,
Polytechnicheskaya, 29
shem98a@gmail.com

Nikiforov Igor Valerevich
Peter the Great St.Petersburg
Polytechnic University
Russia, 195251, St.Petersburg,
Polytechnicheskaya, 29
igor.nikiforovv@gmail.com

Today, a laborious and non-trivial task is to automate monitoring of hard drives in a cluster infrastructure using the Kubernetes container management system.

The paper discusses existing approaches to monitoring hard drives in the Kubernetes container orchestration system and provides a comparative analysis of them. Based on the presented analysis, a conclusion is drawn on the need to improve and automate approaches. The paper proposes an approach to automating the collection of metrics from hard drives by implementing the Kubernetes "operator" for a tool with which you can effectively obtain information about the state of hard drives in the system.

As results, the temporal characteristics of collecting information about disks using existing approaches and the proposed approach are given. Numerical results and graphs showing the gain of the proposed approach are presented.

Key words: Monitoring of disk media, Kubernetes Operator, Container Orchestration System.

I. Introduction

Currently container orchestration systems for the automatic deployment of applications in a cluster infrastructure are gaining popularity. These systems include Kubernetes [1], Docker Swarm [2], Mesos [3].

At the same time a lot of applications need to store their data on hard drives, which sets in turn increasing requirements for the reliability of used hard drives. That's why special attention is paid to monitoring of hard drive state in a cluster infrastructure to prevent their failure and data loss. Many companies such as Dell EMC encounter the problem of disk monitoring.

System administrators and engineers who monitor hard drives in a cluster system use different methods and approaches for obtaining disk information, discussed in Section II.

Section III provides a theoretical basis and defines container orchestration systems using Kubernetes as an example. Section IV describes the operator pattern for Kubernetes.

Section V describes advantages of the proposed approach for automating disk monitoring in a Kubernetes cluster.

Section VI covers the implementation of the proposed approach using the "operator" for disk monitoring tool.

Section Ⅶ presents the results and temporal characteristics of collecting information about disks using existing approaches and the proposed one. Charts show the time gain while using the proposed approach.

II. Approches for evaluation of the hard drives statuses

One way to get the state of hard drive is to use utilities built into the Linux operating system such as lsblk, fdisk, etc. lsblk utility [4] displays information about all available or specified device blocks. It reads information from the sysfs and udev db file systems. The problem with this approach is that the lsblk utility does not provide information about the current state and availability of the disk.

Another disk health evaluation utility is smartctl. This Linux utility allows to retrieve S.M.A.R.T. disk information. S.M.A.R.T. is a technology which allows to analyze and predict the state of hard drives[5]. S.M.A.R.T. monitors the main characteristics of the drive. Each receives estimates and then recounts them into numbers. Depending on the reference value, the state of the disk can be estimated from the result. There are over 150 S.M.A.R.T. indicators with their own reference values, so manual analysis of such information is quite time-consuming for a person. To reduce the complexity of the analysis S.M.A.R.T. indicators, neural networks can be used, which is described in [6, 7].

There is also an intelligent platform management interface (IPMI) designed for remote monitoring and control of a computer system [8]. It is possibly to connect remotely to the server and manage its operation using IPMI. The IPMI specification standardizes an interface. Various companies have the implementations of this interface: IDRAC (DELL), Cisco IMC (Cisco), ILO (HP). This approach is more efficient than using lsblk and smartctl utilities, because specific IPMI implementations (IDRAC, IMC, ILO) provide a graphical interface and a wide range of server functionality in addition to monitoring hard drives. The disadvantage of this method is the lack of free licenses for products that implement the IPMI interface.

The last of the considered approaches is the Linux ipmitool command [9], which implements the interface, but its functionality is limited by the ability to provide information about FRU, LAN configuration, sensor readings and remote power management. The hardware also must have a special BMC port in order to use IPMI, so this approach is not universal.

Considering the approaches, the automating analysis of the state of hard drive through the implementation of Kubernetes "operator" of the disk monitoring tool is proposed. A more detailed definition of the "operator" of Kubernetes is given in Section IV.

The proposed approach is better than others, because of efficient and automatic provision of information on the current state of disks without creating additional load on the Kubernetes cluster;

III. Container orchestration systems

Containerization is an approach in software development which allows an application or service, its dependencies and configuration (abstract deployment manifest files) to be packaged together into a container image [10]. In fact, this is virtualization at the operating system level. Containers greatly simplify and automate application deployment regardless of environment. In turn, the Docker tool [11] helps simplify the creation and launch of containers. As modern applications include more and more containers and distributed servers require complex management and deployment of applications, there is a need for container orchestration systems.

Container orchestration allows to determine how to select, deploy, track and dynamically manage the configuration of multi-container packaged applications [12].

Container orchestration concerns not only the initial deployment of multi-container applications, but also includes management, for example, scaling a multi-container application as a single object.

The most popular is the Kubernetes container orchestration system [1]. Kubernetes is the open source software needed to manage and deploy a Docker container cluster.

A Kubernetes deployment is known as a cluster. Kubernetes cluster can be imagined as two parts: a management layer, which consists of the main node(s) and worker nodes. Pods consisting of containers are made on working nodes. Each node represents its own Linux environment and can be either a physical or virtual machine.

IV. Pattern «operator»

"Operator" is a Kubernetes plug-in that uses user resources to manage applications and their components [13].

A resource in Kubernetes stores a collection of specific types of API objects. Kubernetes objects are persistent resources in the Kubernetes system. Each resource has HTTP request at a unique address, which is processed by the server with the Kubernetes API and returns information about this resource. Kubernetes uses these entities to represent the state of a cluster. For example, Kubernetes Deployment is an object that can represent an application running in a cluster. While deploying applications, there should be a specification for the final configuration. For example, setting up three replicas of an application makes Kubernetes system reads the specification and starts three instances of the desired application, updating the state according to the configuration.

Pods can be example of resources. It is embedded resource and it contains a collection of pod objects. Custom Resource is a Kubernetes API extension that is not necessarily available when Kubernetes is installed by default.

"Operator" combines user resources and custom controllers. They allow to monitor the specified resources and maintain their state in accordance with the value the user set. When a corresponding event occurs with a resource, the "operator" reacts and performs a specific action.

V. Disk information gathering automating approach using «operator »

Figure 1 shows the scheme of the disk information gathering automating approach in the Kubernetes cluster and user interaction.

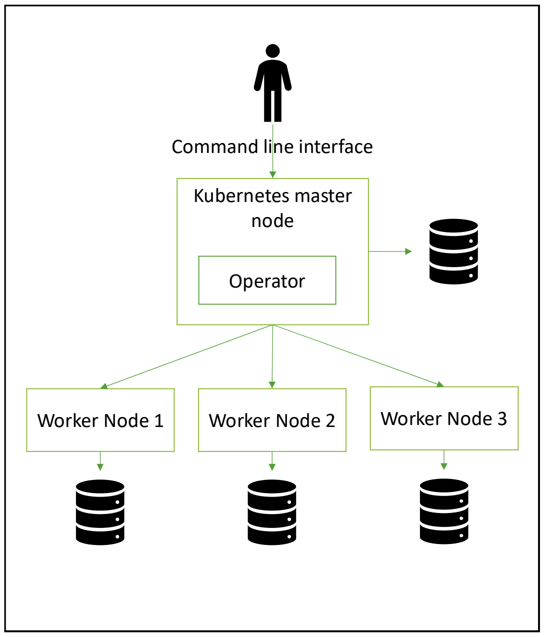

Fig. 1 Scheme of the "the disk information gathering automating approach " in the Kubernetes cluster

The approach proposes to introduce an "operator" into the Kubernetes container orchestration system to automate monitoring of hard drives on nodes. "Operator" is an application deployed on a Kubernetes cluster. It works with its Kubernetes API and monitors events within the cluster. Its task is to collect information from cluster nodes about hard drives, create disk objects on the cluster and monitor them.

Once objects are created, Kubernetes can provide information about them through a client application. Using the command line utility, the user can access the Kubernetes cluster to obtain information about the specified resources. The use of the "operator" has several advantages:

1. "Operator" allows to take advantage of Kubernetes, e.g. work with custom resources.

2. One of the advantages of the "operator" is the ability to process the network events of the Kubernetes cluster (GET, CREATE, DELETE, PATCH, etc.), that is, the programmer decides what should happen to the custom resource during an event.

3. Since a resource is created in the system, this opens up the possibility of using the Kubernetes command-line interface. The user can get information about all resources with a single command (Fig. 3).

4. The information that Kubernetes can show about resources also might be regulated. For example, if you need to know the status of disks, then the command line can show only the requested information. This filtering approach reduces human analysis time.

5. Kubernetes provides data in the form of a table, which simplifies the analysis of information by a person.

6. Kubernetes administrators can create volumes, knowing which drives can be used for this.

Thus, the use of the "operator" allows not only to automate monitoring of hard drives, but also to reduce the time of analyzing disk information. Working with the Kubernetes API allows to take advantage of Kubernetes described above.

VI. «OPERATOR» IMPLEMENTATION

A. *Structure of «operator» units*

Figure 2 shows the architecture of the "operator" for disk monitoring.

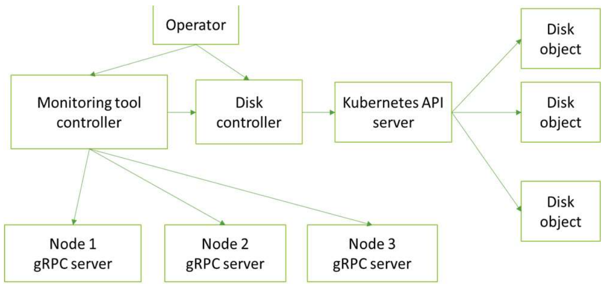

Fig 2 The architecture of the "operator"

"Operator" is a software tool for Kubernetes, which implements two controllers: monitoring tool controller and disk resource controller.

The controller of the monitoring tool deploys a server on each node of the cluster, which collects information about the hard drives of this node on request.

The disk resource controller requests this information and sends a request to create a resource in Kubernetes. Kubernetes API server based on the received information creates custom resources (CR) for disks. Disks CR store information about a specific drive from a node, for example, serial number, size, etc.

B. *Technical implementation features*

Since all Kubernetes "operators" have the same structure, it is possible to generate common code, filling it with own logic. Usually such tools are used for generation as Operator-SDK [14], Kubebuilder [15], Code-Generator [16] and written. Usually Golang programming language is used for «operator» development

The Operator-SDK tool was used in this work. It allows to generate a resource structure template for Kubernetes, in the case of an "operator" for disk monitoring - Disk Custom Resource Definition (CRD) and a resource represented disk monitoring tool on each machine in the Kubernetes cluster. It runs as a gRPC server. To generate code template Operator-sdk has a command (fig. 3). Controllers have also been created to manage these resources. Each controller has a synchronization function (Reconcile loop). It is called by the Kubernetes system every time during any events happening to a custom resource.

```
$ operator-sdk new monitoring-operator --repo
github.com/example-inc/monitoring-operator
$ cd monitoring-operator

$ operator-sdk add api --api-
version=app.example.com/v1alpha1 --kind=Disk

$ operator-sdk add controller --api-
version=app.example.com/v1alpha1 --kind=Disk
```

Fig. 3 Command for generating code template

- First command generates Golang project for «operator».
- Second command generate Golang structure represented Kubernetes objects (fig. 4).
- Third command generate code template for object controller with Reconcile loop.

```
type DiskSpec struct {
        DiskID          string
`json:"disk_id"`
        NodeID          string
`json:"node_id"`
        Path            string
`json:"path"`
        Capacity        int
`json:"capacity"`
        SerialNumber    string
`json:"serialNumber"`
        DriverHealth    string
`json:"driveHealth"`
}
```

Fig 4 Disk structure in Golang

In the operator implementation, the controller of the disk monitoring tool in the Reconcile loop ensures that the gRPC server is running on each node of the cluster. In turn, the Disk CRD controller in the synchronization function creates a connection with the monitoring servers and makes a request for information about the disks. Then using a REST client the request for creating resources is sent. Based on the information received, Kubernetes creates disk object. If such object already exists, then Kubernetes updates it. The synchronization cycle is called every 2 seconds to keep disk information up to date.

To build docker image the command on fig. 5 can be used.

```
operator-sdk build operator:v0.0.1
```

Fig 5 Command for building docker image

To deploy "operator" in Kubernetes cluster Helm tool was used [17]. This tool allows to deploy application using one command (fig. 6). We can develop special manifests (charts) for Helm to represent "operator" application in Kubernetes. These manifests contain all necessary information about the application, so Kubernetes deploy it correctly. For example, we can specify how many replicas of application should be deployed on Kubernetes cluster.

```
helm install operator charts/operator
```

Fig 6 Command for deploying "operator" in Kubernetes cluster

Thus, Kubernetes has a resource that can be accessed using the command line or an HTTP request.

Output is shown in figure 3.

```
~# kubectl get disks

NAME                  NODE      HEALTH
disk-node1-dev-sda    node1   HEALTH_GOOD
disk-node2 -dev-sdb   node2   HEALTH_GOOD
disk- node3-dev-sdc   node3   HEALTH_UNKNOWN
disk- node4 -dev-sdb  node4   HEALTH_FAILED
```

Fig 3 Command line output

VII. Results

Table 1 shows the results of assessing the status of the Kubernetes cluster hard drives in three ways:

- using the smartclt utility;
- using the IDRAC program (implementation of the ipmi interface);
- the proposed approach with the “operator” Kubernetes;

The characteristics of the tested cluster are:

- the cluster consists of 4 nodes;
- each node has 4 hard drives (system drive: 112 GB SSD, 3 drives: 8 TB HDD)
- each node has a Linux operating system deployed;

The necessary command was entered on each node in order to obtain information.

Measurements were performed three times on the cluster. It was made of the time to enter the necessary tool commands to obtain information about the disks. The time of manual analysis of the output of commands is also measured. The average value of each measurement was taken. In the future, it is planned to automate this process and collect more metrics.

TABLE 1 COMPARATIVE ANALYSIS OF DISK MONITORING APPROACHES

Time\Approach	Smartctl utility	ipmi (IDRAC)	Using operator
T_c (seconds)	5	7	5
T_d (seconds per disk)	10	2	2
T_a (second) in tested system	60	36	28

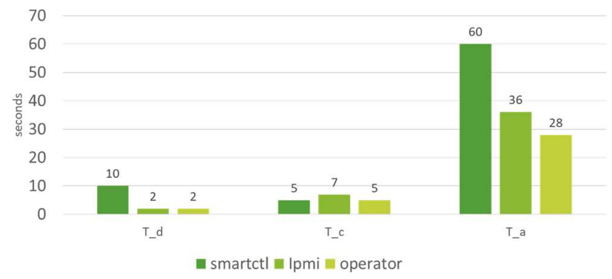

Fig. 5 Chart of spending time on analysis for various instruments

T_d expresses the time for manual analysis of the received logs from the specified methods.

T_c is the time to enter commands into the terminal.

T_a – the total time spent in the tested system. This is the time to enter the command and the time to analyze the logs.

We can conclude from Table 1 and the graph in Figure 5, the gain of the “operator” approach compared to IDRAC was 8 seconds, and for smartctl utility the total analysis time decreased by 2 times.

VIII. Conclusion

Thus an “operator” approach was developed for Kubernetes, which provides information about hard drives in a cluster infrastructure.

This approach requires 2 times less time than using built-in utilities in the Linux operating system. The proposed approach is suitable for different platforms, as it runs in a Kubernetes cluster, which can be run on different platforms.

However, the approach has several disadvantages compared to the existing ones. For example, the smartctl utility and the ipmi IDRAC implementation provide more information about hard drives, while the “operator” provides only status.

In the future, it is planned to expand the list of indicators that can be analyzed through the kubectl utility using the "operator" and to create a graphical interface and scripts to automate command input to reduce the amount of time analyzing disk information.

References

[1] Kubernetes, [online] Available: https://kubernetes.io/

[2] Docker Swarm, [online] Available: https://docs.docker.com/engine/swarm/

[3] Mesos, [online] Available: http://mesos.apache.org/

[4] Adam K. D. (2016), "Linux Administration Cookbook", Birmingham, UK: Packt Publishing.

[5] A. Chatzidimitriou, G. Papadimitriou and D. Gizopoulos, "HealthLog Monitor: Errors, Symptoms and Reactions Consolidated," in IEEE Transactions on Device and Materials Reliability, vol. 19, no. 1, pp. 46-54, March 2019.

[6] S. Ganguly, A. Consul, A. Khan, B. Bussone, J. Richards and A. Miguel, "A Practical Approach to Hard Disk Failure Prediction in Cloud Platforms: Big Data Model for Failure Management in Datacenters," 2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService), Oxford, 2016, pp. 105-116.

[7] X. Sun et al., "System-level hardware failure prediction using deep learning," 2019 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, USA, 2019, pp. 1-6.

[8] IPMI Specification, [online] Available: https://www.intel.ru/content/www/ru/ru/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html

[9] ipmitool man page, [online] Available: https://linux.die.net/man/1/ipmitool

[10] M. Abdelbaky et al., (2015) Docker Containers across Multiple Clouds and Data Centers, 2015 IEEE/ACM 8th Int. Conf. on Utility and Cloud Computing, Limassol, 2015.

[11] Docker, [online] Available: https://www.docker.com/

[12] I. M. A. Jawarneh et al., "Container Orchestration Engines: A Thorough Functional and Performance Comparison," ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 2019, pp. 1-6.

[13] Operator Pattern [online] Available: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

[14] Operator SDK [online] Available: https://github.com/operator-framework/operator-sdk

[15] Kubebuilder [online] Available: https://github.com/kubernetes-sigs/kubebuilder

[16] Code-generator [online] Available: https://github.com/kubernetes/code-generator

[17] Helm [online] Available: https://helm.sh/