Automated Object Storage Management Approach with Operator SDK and Custom Resource Definition

Kirill Stonozhenko, Igor Nikiforov, Sergey Ustinov
Peter the Great St. Petersburg Polytechnic University
St. Petersburg, Russia
stonozhenko.km@edu.spbstu.ru, nikiforov_iv@spbstu.ru, usm50@yandex.ru

***Abstract* — The work is devoted to the study of automation tools for managing stateful applications in the Kubernetes environment, particularly object storage systems. A review of existing management tools capable solving the set tasks is made. A comparative description of the considered tools based on review is given and a tool is selected that meets the introduced criteria: popularity, support form Kubernetes, reactivity of developed operator, additional features, and others.**

An approach to automatic object storage management using the Operator SDK and Custom Resource Definition is suggested. As a result of comprehensive comparative analysis of tools Kubebuilder, Juju, Metacontroller, Kudo and Operator SDK, the last one was chosen as a base of approach implementation. The architecture of the system for managing a containerized version of storage systems based on the Kubernetes platform and integrating the operator with a user monitoring system is proposed.

The described approach is implemented in a software tool – an operator of the object data storage system resource. The paper describes the details of software implementation, the structure of the storage custom resource descriptor, and methods for testing the end system.

As a result, an object storage management system based on the Kubernetes platform was created, which made it possible to reduce both labor costs for supporting and maintaining the system, and it's cost by reducing dependence on hardware. Moreover, described approach corresponds to such features of modern object storages as multi-tier, erasure coding support, geo-replication, cluster topology that is quite innovative among existing automated storage management approaches on Kubernetes platforms.

I. Introduction

The amount of information produced by humanity and requiring storage is increasing every day - a study [1] conducted by the consulting firm IDC (International Data Corporation), specializing in information technology [2], showed that approximately 1.7 Mb of data is generated every second by single human, which in absolute terms is approximately 40 zettabytes per second.

Modern personal computers allow storing up to several terabytes of data, which, on the one hand, is a sufficient amount of memory for the average computer user, but on the other hand, a very modest value for large companies that produce and process petabytes and even exabytes of data. Social networks, file hosting, web applications and other programs that communicate via the Internet can exchange even a larger amount of information that require considerable memory resources for storage.

In addition to information carriers, modern data storage complexes should also contain computing devices capable of performing multi-threaded continuous operations of writing, reading, deleting, and modifying data. Along with this, data storage systems are also responsible for the security and integrity of stored information - access to data is provided only to those users who have the appropriate rights, and if one or more hardware storage nodes fail, the data must be restored using the remaining copies.

All these requirements are implemented in object storages [3], a distinctive feature of which is the storage of all data in the form of objects located at the same logical level and defined by a set of metadata. The absence of a hierarchy in such sort of storages makes this solution infinitely scalable, which allows deploying system on a group of independent nodes called a cluster. The object-oriented nature of the storage system indicates lack of data unit structure, so any data format can act as a storage object and makes this type of storage universal.

However, the above specifics of object storage systems as a software solution dictates certain requirements for the hardware, which increases the cost of equipment and, as a result, reduces the potential customer base of the manufactured complex.

Reducing the dependence between the hardware platform and the software of the system could be solved by the well-known virtualization technique [4] – creating an additional software abstraction layer between the user application and the hardware. However, until recently, existing solutions for deploying virtual machines on a cluster were poorly aligned with the idea of horizontal scaling and required a significant pool of resources.

Everything has been changed with the advent of containerization [4-7] - a way to allocate a lightweight set of system resources to programs - and container orchestration systems [8-10] - software that monitors the execution of containers on a provided set of resources and is responsible for maintaining a certain state of the cluster.

One of the most popular orchestration systems - Kubernetes [11-13]. It is designed to run a huge number of programs wrapped in containers on a set of nodes united in one network, which allows scaling the cluster vertically and horizontally without limits. One of the distinguishing features of Kubernetes is the presence of a replication controller [14] responsible for maintaining multiple copies of stateless containers [11, 15]. Stateless programs can be web clients that form user requests based only on input data, and simple web servers that send responses to a request based only on its content.

Along with stateless applications, Kubernetes terminology also defines stateful applications that are forced to save their state as they execute. Their replication can no longer be performed by sequentially deleting a copy of the program and recreating it in a new location. Work features of each individual stateful application require the creation of a separate custom replication controller, also called an operator [16,17], which monitors the state of its replica set.

Now there are many tools for creating stateful application operators in the Kubernetes environment. Some of them, considered in this paper, have been used by stateful software developers to create their own Kubernetes operators. Such stateful software include databases, metric collectors, machine learning programs, logging systems, and many others.

Transferring an object storage system to the Kubernetes container platform and creating a separate operator for monitoring the state of stateful components is an advanced and relevant direction in the development of object storage, which reduces the cost of storage development, testing, implementation, and maintenance, and also provides the ability to deploy the system on almost any set of hardware resources.

In this paper, we consider an approach to automatic management of object storage based on the use of the Operator SDK framework and Custom Resource Definition.

II. Related works

One of the main components of the developed approach for managing object storage in the Kubernetes environment is an operator - a kind of user resource controller. Developing an operator from a scratch, just like its further maintenance as a product, is a time-consuming and expensive process, which is a consequence of both the complexity of the internal structure of Kubernetes controllers and the rather frequent change of Kubernetes API versions.

In this regard, many tools have been created for your own Kubernetes custom resource operators' implementation. Some of these tools are described below.

Kubebuilder is a command line utility, providing functions for extending the Kubernetes cluster API. The main purpose of the program is to create custom resources [17] and controllers for their monitoring. The utility is implemented and builds controller code in Golang, the native language of the Kubernetes infrastructure. The distinctive features of the Kubebuilder project include the ability to include the configuration and the versioning of the resource being defined.

An alternative to Kubebuilder is the Operator SDK, an open-source toolkit whose main purpose is to build Kubernetes operators. The set includes the operator-sdk utility, which provides a list of commands for generating an operator template for any type of custom resource. The operator itself can be implemented both in Golang (using scripts from the Kubebuilder library) and using Ansible playbooks - configuration management scripts - or Helm charts. In addition, the Operator SDK allows you to connect the developed operator to the Operator Lifecycle Manager - a separate operator lifecycle management program, whose tasks include monitoring the use of system resources, saving metadata about a specific version of the operator, simplifying the deployment, deletion and testing of the custom operator.

Another tool for creating Kubernetes operators is Juju [18]. In Juju terminology, user resource controllers are called "charms" [19]. Python is the preferred language for implementing charms, but other scripting languages can be used that include libraries for interacting with Juju. Along with the "juju" and "charm" command line utilities, the Juju package also downloads the Charmed Operator Lifecycle Manager, similar to the Operator Lifecycle Manager from the Operator Framework. But unlike the second program, the controller supplied by Juju is the software required to run on the cluster in parallel with user statements, which negatively affects the performance and complexity of system developed.

You can also use Metacontroller [20], a Kubernetes API extension developed by Google, to create custom operators. The tool provides the ability to create bundles of controllers and stateful web applications through a web-hook. These applications implement custom resource management logic and can be written in any language capable of handling JSON-like objects, such as Jsonnet or Python. Google's suggestion is a declarative way to create a Kubernetes operator, which simplifies the development process, but increases the load on the cluster due to the deployment of additional entities.

For a completely declarative way to create Kubernetes operators, you can apply KUDO (Kubernetes Universal Declarative Operator) – kubectl Kubernetes CLI plugin. Statements developed with KUDO manage user resources by defining tasks and plans in statement descriptors in YAML format. The undoubted advantage is the ease of creating controllers, however, the declarative method of setting operators reduces the ability to manage resources - KUDO allows you to deploy complex stateful applications but is not able to respond to changes in the cluster.

A full comparison of the considered tools is presented in Table. 1.

During the comparison, the Operator SDK was chosen as the tool for creating the Kubernetes object storage operator, as it meets our requirements more than others.

Table. I. Comparison of Operator development tools

Criteria	**Kubebuilder**	**Operator SDK**	**Juju**	**Metacontroller**	**KUDO**
Popularity (thousands of Google search results)	92	18 100	1 610	6 360	611
Requires additional controller level	-	-	+	+	-
Support from k8s	+	+	-	+	-
Additional opportunities	+	+	-	-	-
Operator reactivity	+	+	+	+	-

III. Proposed Approach description

A. Approach features

The software of modern object data storage systems is a combination of several services that implement various functions. Among them are software components for accounting for user roles of the storage system, providing access to one or more interfaces of various levels, distributing data between several geographically separated clusters, and many others.

The main service of the system is the storage service, a component responsible directly for encoding and saving user data. Encoding is understood as the splitting of the input stored information into several sets of transformed data, from which it is possible to restore the original information. The number of sets and the type of transformation depend on the selected data encoding scheme. The key feature of this procedure is the guarantee that when the received data sets are placed on different nodes and one or more of them fails, the system is able to restore all information on the remaining sets, after some time recode the data and save them on the working nodes.

The difference in data encoding schemes complicates the process of managing several storage service replicas, therefore, the main difficulty in designing an automated object storage management system is focused on the replication of this component.

Furthermore, none of modern approaches of storage system management on virtualized deployments included extension of Kubernetes replication controlling system in order to maintain erasure coding schemes and geo-replication of stored data. Storage service replication fully demands on orchestrator or simple custom controller reconciliation logic [21, 22].

The main features of the developed approach include:

- Reducing labor costs in the implementation, maintenance and operation of the system;
- Transition to implementation of storage on any provided set of computing resources;
- Abstracting from the hardware component of the system and shifting the responsibility for scaling and replicating storage systems to the Kubernetes platform;
- Use built-in Kubernetes resources to create our own event monitoring system.

The first three features of the approach are related to the transfer of the system to a virtual Kubernetes platform. The system requirements for the hardware are dictated by the very platform, that takes responsibility for connecting the nodes into a single network and deploying Kubernetes on the resulting cluster. Also, the platform can provide a graphical user interface of various levels for managing the network, nodes, Kubernetes resources and interacting with executable programs.

Elastic Cloud Storage (ECS) [23] from Dell Technologies was chosen as the object storage to which the developed automation approach is applied. While vSphere from VMware and OpenShift from RedHat were the Kubernetes platforms that should support deployment of containerized version of object storage.

The fourth specified feature of the approach involves the use of the event resource to monitor the state of storage systems during service procedures - cluster maintenance operations or changing storage system parameters. Examples of service procedures include the following operations:

- Horizontal storage server expansion – increase of service replicas number;
- Vertical storage server expansion – increase of volumes number that bound to each replica;
- Object storage version upgrade;
- Volume or disk replacement;
- Entering maintenance mode of cluster node, etc.

During each of the presented service procedures processing, the system must ensure the correct replication of all components of the object storage and notify the user of all changes in the state of the storage system.

B. Architecture

A Kubernetes operator is required [24] to organize automatic management of object storage components when deploying a system or performing service procedures. Its principle of work is shown in Figure 1:

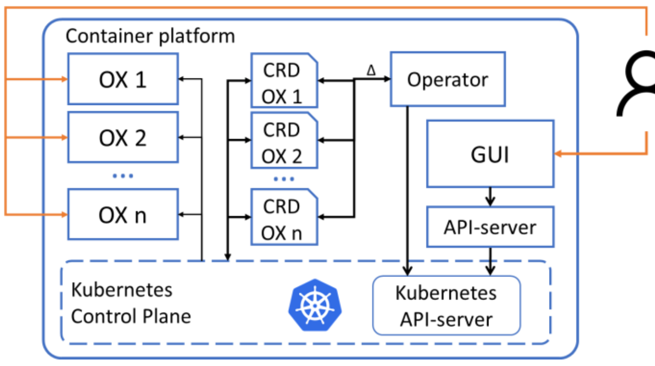

Fig. 1. System architecture

All software components of the system are executed on the Kubernetes platform, that provides a graphical user interface and a deployed Kubernetes orchestration system. The central element of the orchestration system is the Control Plane. Interaction with it is carried out through the Kubernetes API-server, which accepts requests from any client - the native CLI kubectl [13] or a program using the k8s-client library.

The object storages deployed on the platform, on the one hand, are a set of Kubernetes services, each of which requires the creation of a collection of Kubernetes resources, such as volumes, pods, cluster-roles, replica sets [11] and others. Control plane manages all these resources according to YAML descriptors containing a description of the desired state of the resource.

On the other hand, object stores are resources themselves that are a user-defined extension of the Kubernetes API. Each storage instance, like any resource, is described by its own descriptor. But since the resource is user-defined, its descriptor is called Custom Resource Definition [17], and its management is carried out by a custom controller named operator.

Object storage is managed by the operator upon the occurrence of certain events associated with a change in the state of nodes, volumes, or storage descriptors. These events are triggered by the launch of service procedures that are initiated by the user using the platform's graphical interface.

The operator manages the Kubernetes resources of the storage system components by generating requests to the Kubernetes API server [11].

The collaboration of the operator and the Kubernetes control plane maintains the correct operation of the storage system, which is accessed by the user through any provided storage interface (HTTP, S3, etc.)

C. Operator and monitoring system interaction

Event Kubernetes-resource [11] allows you to monitor the status of other cluster resources. Generation of events during the operator's work can be useful both for tracking the current state of the storage system by the end user, and for implementing a reaction to some errors that occur during object storage management.

If some error occurs during the execution of any service procedure, it is not enough to inform the user about it. Depending on its severity, the system should give recommendations for its elimination, ask for help from the system administrator or even from the support engineer responsible for the implementation and maintenance of the software product.

To implement the described logic, a Kubernetes event monitoring system is required - an intermediate layer between the events generated by the system components and all the specified actors.

The general scheme of interaction between the operator and the monitoring system is shown in Figure 2.

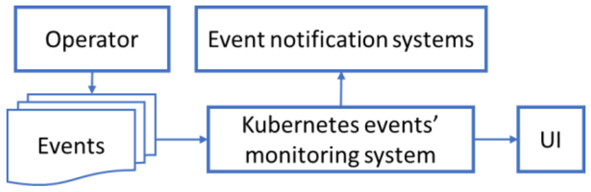
Operator
Event notification systems
Events
Kubernetes events' monitoring system
UI

Fig. 2. Operator and monitoring system integration

IV. Implementation

A. Object storage descriptor design

To organize the processing of all components of the object data storage system by the operator as a single whole, it is necessary to implement a custom resource definition (CRD) containing the settings of all services presented in a containerized form.

Like all Kubernetes descriptors, the storage CRD contains the following sections:

- Used Kubernetes API version (apiVersion);
- Kubernetes resource kind;
- Resource metadata;
- Resource specification (spec);
- Its current status.

Since the storage descriptor is a custom resource definition, an extension of the standard Kubernetes API, then apiextensions.k8s.io/v1 acts as the CRD version, and CustomResourceDefinition as the resource type. Metadata includes resource name, entity creation time, various labels and annotations.

The specification of a resource is its main characteristic since it contains all the desired settings for both the resource itself and all its components. For instance, for the minimal executable unit of a Kubernetes cluster - a pod - the specification contains the launch parameters of all containers, associated volumes, the name of the node on which the pod want to be running, and other information. Object storage, on the other hand, consists of dozens of components, for each of which it is necessary to define a service, a replica set, a set of stateful pods (statefulset) [11], or some other resources and their characteristics. Therefore, the storage CRD specification is the largest part of the descriptor.

CRD status contains current information about the state of the resource, that allows you to compare it with the desired state. The distinctive parameters stored in this section of the descriptor of our storage system are:

- Phase that reflects storage current state;
- Components' subsection containing brief information about the current state of all storage components;
- Conditions [25] - a special extension of the resource status that allows you to track the time, cause and status of transitions between resource phases;
- Additional information used by the operator in the course of managing the storage resource, such as information about the nodes that are in maintenance state.

B. Reconcile loop implementation

Custom resource processing is implemented in software using the Operator SDK. As mentioned earlier, the package includes the operator-sdk utility, which is used to create, build, run and test the custom resource operator.

The basis of the program is the reconcile loop [24], during which the operator reads the descriptor of the controlled resource and performs actions that bring it to a state corresponding to the specification. In fact, this is the main task of the controller. The cycle runs at a certain interval (for example, once a minute) and checks whether the state of the storage components corresponds to the status of the cluster or the ongoing service procedure. Figure 3 shows the general scheme for processing a user resource in a loop.

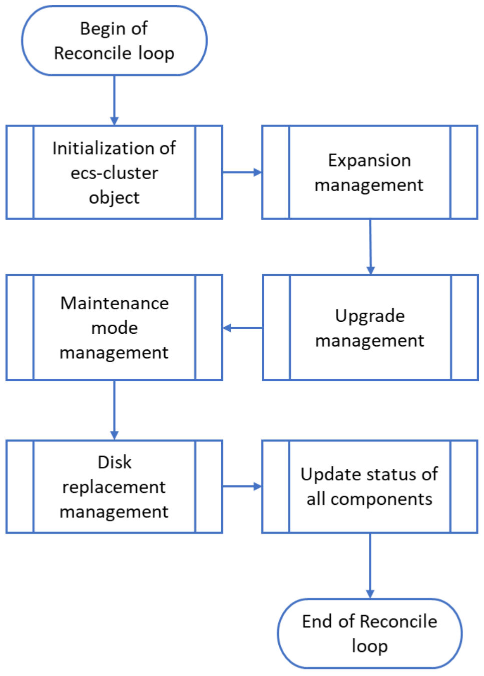
Begin of Reconcile loop
Initialization of ecs-cluster object
Expansion management
Upgrade management
Maintenance mode management
Disk replacement management
Update status of all components
End of Reconcile loop

Fig. 3. Storage resource Reconcile loop

The storage processing logic encapsulated in Operator allows us to track the state of the storage system using a phase - a capacious description of the current status of the storage system. The behavior of our user resource can be described by the storage resource state machine, presented on Figure 4.

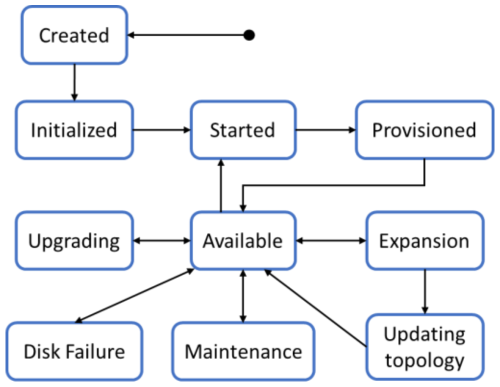

Fig. 4. Storage resource state machine

C. Monitoring system integration

In addition to the current storage phase, the end user can monitor the flow of service procedures and their impact on the operation of object storage components using events. Developed separately from the operator, the event monitoring system allows you to define the severity, the resource and application involved, as well as the cause and message of the event.

If the severity of the event is higher than normal (warning or error), the system generates an issue, which, depending on the settings, can be resolved by the user himself or sent for investigation to the software product support service.

V. Results

A. Testing methods

Testing of the implemented system is carried out at different levels - at the level of the operator code and at the level of the entire system.

To test the functional changes of the operator unit and integration tests were created using the Gomega and Ginkgo frameworks, the main scenarios of which include checking the correct processing of the storage descriptor during various service procedures.

Operator testing as part of the operation of the entire object storage management system at the Kubernetes cluster level is performed using the Jenkins automatic CI/CD pipeline, which downloads software dependencies and the Operator SDK, code syntax analysis, executes tests, builds the image and uploads it to the registry. A pipeline is also implemented that allows you to deploy an entire storage management system on a provided virtual or bare-metal cluster. The resulting system is ready for all kinds of end-2-end tests.

To monitor the absence of a decrease in the efficiency of the object storage during system development, a cluster is regularly allocated for load testing: various service procedures are alternately performed for several days. At the same time, object storages constantly take on the load - the main operations are:

- creation and deletion of user with different roles;
- creation and deletion of namespaces;
- creation and deletion of buckets;
- write, read, modify and delete files to/from buckets;
- File metadata search.

To monitor the state of the storage during testing, the Grafana metrics collection and visualization tool is used [26]. The main metrics collected include the execution time of REST requests, the percentage of unsuccessful requests, network bandwidth, etc.

B. Comparison to classic approach: automation

The main purpose of the developed object storage management system on the Kubernetes platform is to increase the degree of automation of storage management when changing the configuration of both the physical cluster and the storage system itself. To demonstrate the reduction in labor intensity of storage management, a comparison was made of the time to remove a disk from a cluster between a manual approach and the proposed automated approach.

The list of actions for an engineer when removing a disk as part of a classic deployment of object storage without the use of container technologies includes points, presented below.

1. Simulation of disk outage on HAL level.
2. Waiting for data recovery with rest storage replicas.
3. Logic disk replacement from cluster with fabric CLI.
4. File system deletion.
5. Partition table formatting.
6. Disk wipe.

To remove a disk in the developed system, you must perform the following operations in the UI of the container platform:

1. Annotate unhealthy disk or volume.
2. Wait Service procedure to finish.

The average disk replacement time for the classical approach is ~3 hours, and for the automated one it is approximately 1 hour. In total, we have a gain in time of about three times.

VI. Conclusion

Work on operator of the object data storage system helped us to reach following important results.

Firstly, an approach of automated object storage management system was introduced. Distinctive feature of the approach is Kubernetes platform. Operator is used as a crucial element of the system as it performs CRUD operations on Kubernetes resources to manage components of storage that is presented by its own Custom Resource Definition.

Secondly, the operator of the object data storage system developed within the framework of this project was successfully implemented into the storage management system on the container platform. The operator not only manages the storage components during service procedures, but also generates a set of Kubernetes events displayed in the platform UI and informing the user about the general state of the storage.

Operator was implemented based on comprehensive analysis of existing solutions: different tools to develop operator of custom Kubernetes resources were studied that helped us to form knowledge about different ways to build custom controllers and to choose Operator SDK as the base of operator development framework according to its benefits.

Finally, resulting system was compared with storage build using classic approach, that let us to see such qualitative changes as possibility to run new solution on wide specter of container platforms, use different tools made in Kubernetes infrastructure to improve storage system and reduce of cost of outcome product as the responsibility of deployment, replication and low-level system management lies on orchestrator.

Manufactured software solution has shown increase in performance and more wide set of functionalities in comparison to classic object storage system, and was applied to real enterprise object storage system.

On the basis of results further directions of project development were suggested:

- new automation pipelines;
- use of new Operator Framework features;
- multiple Service Procedure support;
- introducing new CRD for service procedures.

One of them is the development of new automation pipelines for configurable regression testing. The Robot Framework and Jenkins capabilities allow us to add settings to run a specific set of end-2-end tests for each service procedure. On the one hand, this would reduce the load on the clusters allocated for testing, and on the other hand, it would cover many previously untested use cases.

Operator Framework was also improved and issued several versions, which now allows us to add to the project such tools as collecting metrics for the operator and all resource controllers included in it. Analysis of the measured parameters will help us to find solutions to improve the performance and speed of the software and reduce the consumption of cluster resources.

Moreover, one of the directions of future work on the operator is the support of several parallel processed service procedures. Modifications in the operator code and specification of the storage resource would allow, for example, updating the storage system version and simultaneously deleting one of the cluster disks.

Furthermore, development of additional components of storage management system showed us that they also need some operator-side management during service procedure handling. Introducing new service procedure CRD would simplify monitoring of different service procedures that affect not only storage services, but also components of the management system.

References

[1] J. Gantz, D. Reinsel. The Digital Universe in 2020: Big Data, Bigger Digital Shadow s, and Biggest Growth in the Far East. IDC IVIEW [Online]. Available: https://www.cs.princeton.edu/courses/archive/spring13/cos598C/idc-the-digital-universe-in-2020.pdf [Accessed Mar. 24, 2022].

[2] International Data Corporation (IDC), “Changing the way the world thinks about the impact of technology on business and society”. [Online]. Available: https://www.idc.com/ [Accessed Mar. 24, 2022].

[3] V. Yu. Shevtsov, E. S. Abramov, “The Analysis of Modern Data Storage Systems”, NBI tehnologii, 2019, vol. 13, no. 1, pp. 25-30 (in Russian).

[4] R. Dua, A. R. Raja, D. Kakadia. Virtualization vs containerization to support PaaS, in: Proc. of 2014 IEEE Int'l Conf. on Cloud Engineering, IC2E'14, 2014, pp. 610 - 614.

[5] S. M. Nanyan, T. N. Nichushkina, “Virtual Docker containers: purpose and application features”, Inzhenernyj Vestnik, 2015, no. 2, p. 2 (in Russian).

[6] J. Turnbull. The Docker Book, 2014, p. 338. [E-book] Available: https://dockerbook.com/.

[7] D. Silakov, “Docker project. Manage virtual environments”, Sistemnyj administrator, 2015, no. 3, pp. 10-14 (in Russian).

[8] D. Silakov, “Tools to manage multiple Docker containers”, Sistemnyj administrator, 2015, no. 5 (150), pp. 11-15 (in Russian).

[9] T. Uphill, J. Arundel, N. Khare, H. Saito, H.-C. Chloe Lee, Ke-J. Carol Hsu, DevOps: Puppet, Docker, and Kubernetes. Packt Publishing, 2017.

[10] Z. A. Orlov, “Study of scaling tools for container virtualization systems”, Alleya nauki, 2017, vol. 2, no. 9, pp. 867-871 (in Russian).

[11] M. Luksha, Kubernetes in action. DMK Press, 2018 (in Russian).

[12] S. Yaremchuk, “Get to know Kubernetes”, Sistemnyj administrator, 2017, no. 1-2 (170-171), pp. 41-45 (in Russian).

[13] J. Shah, D. Dubaria, “Building modern clouds: Using docker, kubernetes google cloud platform”, in: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference, 2019, pp. 184-189.

[14] D. Safronov, K. M. Stonozhenko, I. V. Nikiforov, “Automatic load balancing between streaming data processing and cluster internal tasks using Kubernetes”, in: Modern technologies in programming theory and practice, Peter the Great St. Petersburg Polytechnic University; Dell Technologies; EPAM Systems, Polytech-Press, 2020, pp. 165-167 (in Russian).

[15] Kubernetes Deployment vs. StatefulSets. [Online] Available: https://www.baeldung.com/ops/kubernetes-deployment-vs-statefulsets [Accessed Mar. 25, 2022].

[16] Operator pattern. [Online] Available: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ [Accessed Mar. 25, 2022].

[17] P. S. P. Shenoy, S. S. Vishnu, R. P. Kumar, S. Bailuguttu, “Enhancement of observability using Kubernetes operator”, Indonesian Journal of Electrical Engineering and Computer Science, 2022, vol. 25, no. 1, pp. 496-503.

[18] D. Silakov, “JUJU project. Deploy complex applications with one click”, Sistemnyj administrator, 2015, no. 10 (155), pp. 4-8 (in Russian).

[19] Juju SDK Documentation. [Online] Available: https://juju.is/docs/sdk [Accessed Mar. 25, 2022].

[20] Introduction to Metacontroller. [Online] Available: https://metacontroller.github.io/metacontroller/intro.html [Accessed Mar. 25, 2022].

[21] B. Shrishail, S. Ashish, G. Sagar, J. Chinmay. “A QOS-aware secure personal cloud storage with ubiquitous access and smart home extension”, in: 2015 International Conference on Computer, Communication and Control (IC4), 2015.

[22] C. Wu, V. Sreekanti, J. M. Hellerstein. “Eliminating Boundaries in Cloud Storage with Anna”, CoRR, 2018, vol. 1809.00089.

[23] Dell EMC ECS: powering a data-driven future. [Online] Available: https://www.dell.com/en-us/dt/learn/data-storage/ecs.htm [Accessed Mar. 25, 2022].

[24] A. S. Shemyakinskaya, I. V. Nikiforov, Hard drives monitoring automation approach for Kubernetes container orchestration system // Proceedings of the Institute for System Programming of the RAS. – 2020. – Vol. 32. – No 2. – P. 99-106. – DOI 10.15514/ISPRAS-2020-32(2)-8.

[25] Extend the Kubernetes API with CustomResourceDefinitions. [Online] Available: https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/ [Accessed Mar. 29, 2022].

[26] S. Yaremchuk, “Deploy monitoring Prometheus + Grafana”, Sistemnyj administrator, 2017, no. 5 (174), pp. 36-44 (in Russian).