Date of publication SEP-30, 2023, date of current version JUN-05, 2023.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.22.3.3231

Organization of FPGA-based Devices in Distributed Systems

MYKHAILO MAIDAN[1], ANATOLIY MELNYK[1,2]
[1]Department of Computer Engineering, Lviv Polytechnic National University, Lviv, 79013, Ukraine
[2]The John Paul II Catholic University of Lublin, Lublin, Poland

Corresponding author: Mykhailo Maidan (e-mail: mykhailo.maidan@gmail.com).

ABSTRACT The article proposes using Kubernetes (k8s) as a tool for managing FPGA-based devices in a distributed system. This can help automate programming, monitoring, and controlling the state of devices, and also optimize resource usage, ensure high availability and reliability, and provide security and privacy for data processed by specialized processors. The article provides a practical example of integrating an FPGA-based device into a Kubernetes cluster. It will help to scale, maintain and monitor distributed systems with millions of devices and manage such big systems from one place by using Kubernetes API. Also, it will help to integrate other third-party tools into the system, which makes it to possible to extend the systems. As a future work, the proposed approach can help integrate FPGA and its real-time reconfiguration tool into a distributed system, making it possible to control FPGA on different IoT devices. Overall, using k8s to manage FPGA-based devices can provide significant advantages in such fields as telecommunications, information technology, automation, navigation, and energy. However, the implementation may require specialized skills and experience.

KEYWORDS FPGA; Kubernetes; k8s; CRD; custom resource definition; distributed systems.

I. INTRODUCTION

FPGAs have been proven to be effective in systems that require specialized computing, or systems where computing algorithms require specialized architectural solutions. Such algorithms enable the use of multi-level parallelism, which cannot be used in general-purpose processors (CPUs). For such algorithms, it is better to use specialized architectures that take into account the peculiarities of the algorithm's structure [1-3]. Using high-level programming languages such as C++, C, and others, and specialized compilers, it is possible to convert an algorithm written in a high-level programming language into architecture-level hardware description code in the VHDL or Verilog programming language. This will make it possible to create a specialized processor that efficiently executes this algorithm [4-6].

Increasingly, FPGAs are being used in Internet of Things (IoT) devices for effective data processing at the locations where these devices are installed. For example, in places that are difficult for people to reach or where people do not stay long, such as deserts or glaciers [7-8]. Such devices are part of a larger system that is combined into one large distributed system with many components that communicate with each other. In such a system, there is often a component that deals with the organization of this system since decentralization is not always possible. However, managing FPGAs in a distributed system can be challenging. Each FPGA has its own programming and configuration requirements, and coordinating multiple FPGAs in a cluster can be a complex task. To address these challenges, Kubernetes (k8s) has emerged as a powerful tool for managing distributed systems. Kubernetes is an open system for the automatic deployment, scaling, and management of applications in containers [9-10]. This system deals with the deployment and control of resources in the system and it is also responsible for their reliability and fault tolerance [11-13].

Organizing FPGA-based devices based on k8s means using the k8s container management system to manage the resources of FPGA-based devices. This management may include the distribution of tasks between different devices, or the optimization of the resources of specialized processors.

Using k8s allows you to easily manage and automate the processes of organizing FPGA-based devices, which helps ensure efficiency and reliability when working with such devices. Also, the use of k8s allows you to automate the process of programming, monitoring, and controlling the state of devices. Automation of the programming process can include:

- Distribution of tasks between different devices depending on the availability of resources and the needs of the task.

- Automation of the FPGA configuration process on such devices using containers ensures stability and convenience.
- FPGA status monitoring using k8s tools such as Prometheus, Grafana, which allows you to identify and diagnose problems with FPGA operation.

In addition, using k8s allows easy scaling of FPGA-based device resources depending on the needs of tasks, which allows you to optimize the use of resources and maintain high performance, and it also allows the real-time integration of FPGA-based devices into existing systems, for example, to process large volumes of data in real-time, or integrate FPGA-based devices into the architecture of distributed systems and use the FPGA-based device to optimize the data processing process [14].

Integrating FPGA-based devices into a distributed system using k8s allows you to ensure efficiency and reliability when working with them, as well as make them more accessible and easily integrated into existing systems.

Using k8s also allows for easy lifecycle management of FPGA-based devices. Including automated preparation and connection of FPGAs, as well as updating and maintenance of FPGAs, the use of k8s allows you to ensure the high availability and reliability of a system that includes FPGA-based devices [15].

Also, using k8s to organize FPGA-based devices can provide security and privacy for data processed by specialized processors. This may include control of access to FPGA resources, as well as cryptographic protection of data that is processed on the FPGA [16].

Using k8s to organize the operation of FPGA-based devices can help in such fields as telecommunications, information technology, automation, navigation, and energy.

For more complex tasks, the use of devices based on FPGA and k8s can give a significant advantage compared to traditional solutions, as it allows for the achievement of high performance and speed of data processing [17].

It is important to note that implementing FPGA-based device organization with k8s can be complex and require specialized skills and experience, especially if your task is complex.

In this article, we will explore how organizations can leverage Kubernetes to manage devices based on FPGA in a distributed system. We will discuss the benefits and challenges of using FPGAs in distributed systems, and how Kubernetes can help overcome these challenges. We will also provide a practical example of deploying an FPGA-based application on a Kubernetes cluster.

II. RELATED WORKS

Implementation of devices based on FPGA (Field Programmable Gate Array) plays an important role in the development and implementation of modern systems. FPGA-based hardware organization is a critical aspect that determines overall system performance, scalability, and reliability.

Considering the latest research, devices based on FPGA can effectively perform tasks in the data centers of large corporations because there is an opportunity to adapt them to system requirements by improving performance, energy efficiency, etc. [18]. Also, such devices can be used in big data analysis because they have a flexible architecture that can be changed and trained [19]. To create such devices, you need to use a systematic approach to the design and development of reconfigurable systems that use FPGA, it is necessary to implement the design, development, and testing methodologies of the system [20].

When such devices are developed, the task is to create effective algorithms that can be parallelized and thus, ensure the achievement of the best speed in data processing. The development of such algorithms requires a lot of time and research to determine the various characteristics of the algorithm, but the implementation of such algorithms gives the system an advantage in the speed of data processing [21-23].

Besides speed, reconfigurable hardware has many other advantages over the microprocessor. Reconfigurable hardware helps to reduce energy and power consumption. In a reconfigurable system, the circuitry is optimized for the application, such that the power consumption will tend to be much lower than that for a general-purpose processor. A recent study reports that moving critical software loops to reconfigurable hardware results in average energy savings of 35% to 70% with an average acceleration of 3 to 7 times, depending on the device used [24].

There is also an open platform to control IoT devices made by Amazon company called AWS IoT Service. The AWS IoT service provides cloud services that connect IoT devices to other devices and AWS cloud services. It is also possible to use this solution for the FPGA-based device. The AWS IoT service provides an SDK that allows any type of device to be integrated into its ecosystem and manipulated by that device using this service. [25]

In summary, the organization of FPGA devices in distributed systems is a key aspect to which researchers have paid considerable attention in recent years. Using devices based on FPGA in data centers and big data processing systems has proven to be beneficial due to their ability to increase productivity and energy efficiency. However, the implementation of these devices in distributed systems also arises some problems that must be solved to fully realize their potential.

III. KUBERNETES CUSTOM RESOURCE ARCHITECTURE

Kubernetes (k8s) is one of the most popular container management systems that allows you to automate the deployment, scaling, and maintenance of containers. One of the key features of k8s is the ability to create your own resources, or "custom resources" (CRs).

Custom resources are special objects that can be created, updated, and deleted using k8s. They can extend the Kubernetes API by creating new types of resources that are not part of the standard Kubernetes system resources, such as databases, message queues, or even specialized FPGA-based processors [26].

To create custom resources, you must first define a custom resource (CRD - custom resource definition) in YAML format. A custom resource definition defines the fields of the resource as well as the constraints for that resource, as well as the rules for checking constraints for that resource. Once the resource has been defined, it needs to be registered in the system using the Kubernetes API [27].

Other advantages of custom resources may be that they allow a third party to integrate services into their own system very quickly. For example, you can use a custom resource to work with the database, and then use a tool like ReplicaSet or

Services to automatically scale and distribute the database between clusters.

The pattern operator is a pattern design that is often used in Kubernetes to automate and manage complex applications. It allows engineers to delimit specific application logic into a specialized controller that is part of a Kubernetes cluster. The operator monitors clusters for user resources and uses the Kubernetes API to create, update, and delete the required resources to run the application [28].

The combination of the "operator" pattern and custom resources allows engineers to create a custom controller that automatically manages the application's life cycle, and also provides a simple and convenient mechanism for interacting with the application using the Kubernetes API.

Controllers are used for monitoring the state of system resources. The controller works in the eternal cycle mode and constantly monitors the state of the resource attached to it. Since the reference state of the resource is described in the resource, the controller constantly monitors the current state of the resource, and if the current state of the resource does not correspond to the reference state, the controller tries to bring the current state of the resource to the reference state. In this way, the system controls its reliability and fault tolerance of itself. This process is fully automated and is shown in Fig. 1.

In general, custom controllers allow engineers to describe algorithms for the administration and monitoring of individual system components that are not typical for the Kubernetes API.

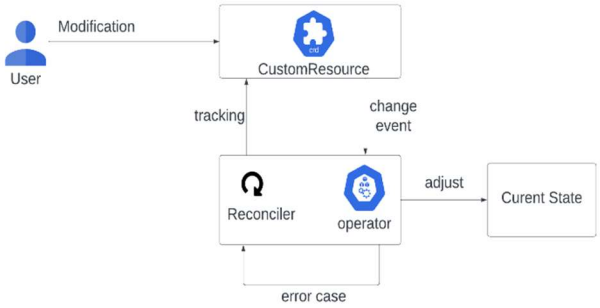

Figure 1. Kubernetes Operator Workflow Diagram

IV. METHODS

A. CONCEPTUAL MODEL DEVELOPMENT

To develop a conceptual model that will confirm the hypothesis about the emergence of user resources, as well as the "operator" pattern, it is necessary to divide and describe each component of the system, as well as describe the role of this component in the system.

Let us first divide the system into the following components:

- User resource (CRD) - a component that describes an FPGA-based device in a distributed system. Without this FPGA-based decent module, it is impossible to play because k8s will not understand the interface for interacting with such resources. This element is a resource object but only describes the structure of objects that can be described in the system.
- The FPGA operator is a user controller whose function is to monitor changes in the user resource (CRD) that describes the FPGA-based device. The operator is responsible for the connection, monitoring, and control of the FPGA-based system in the distributed system.
- FPGA Controller - a module that is responsible for working with FPGA. This module is not part of k8s, but is responsible for working with the FPGA on each device, according to which the implementation may depend on the FPGA manufacturer.
- The central control module (Contol Plane) is responsible for the integration of user resources with other system components using appropriate interfaces: APIs or the message bus, thus ensuring effective communication.

Fig. 2 shows the structural diagram of the system component.

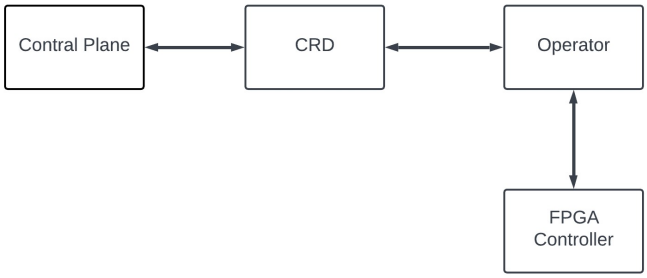

Figure 2. Structural diagram of the system component

Having described the structural diagram of the system, it is possible to divide it into functional blocks and describe the interaction of these functional blocks with each other, as well as describe the structure of the Kubernetes cluster, which allows working with FPGA-based devices. Fig. 3 shows the scheme of the interaction of system components. Fig. 4 shows the structure of the Kubernetes cluster, which includes all functional elements.

The cluster includes the following functional elements:

- CRD is a description of a user resource that allows the integration of FPGA-based devices into a distributed system. This description contains an interface through which it will be possible to interact with resources.
- Node - microcontrollers based on ARM processors and with a set of peripherals for communication with the outside world: Ethernet, programmable I/O interface (PIO) between the microcontroller and a specialized processor based on FPGA, RAM.
- FPGA Device - synthesized processor for specialized calculations.
- Node Custom Resource - a custom resource in the k8s system that describes Node as objects in a distributed system.
- 8s API Server is a distributed system server, which is an interface through which other services can communicate with the cluster.
- ETCD is a distributed database in which k8s stores information about the state of the cluster.
- Operator - a module for monitoring and managing the state of the Node object in a distributed system.

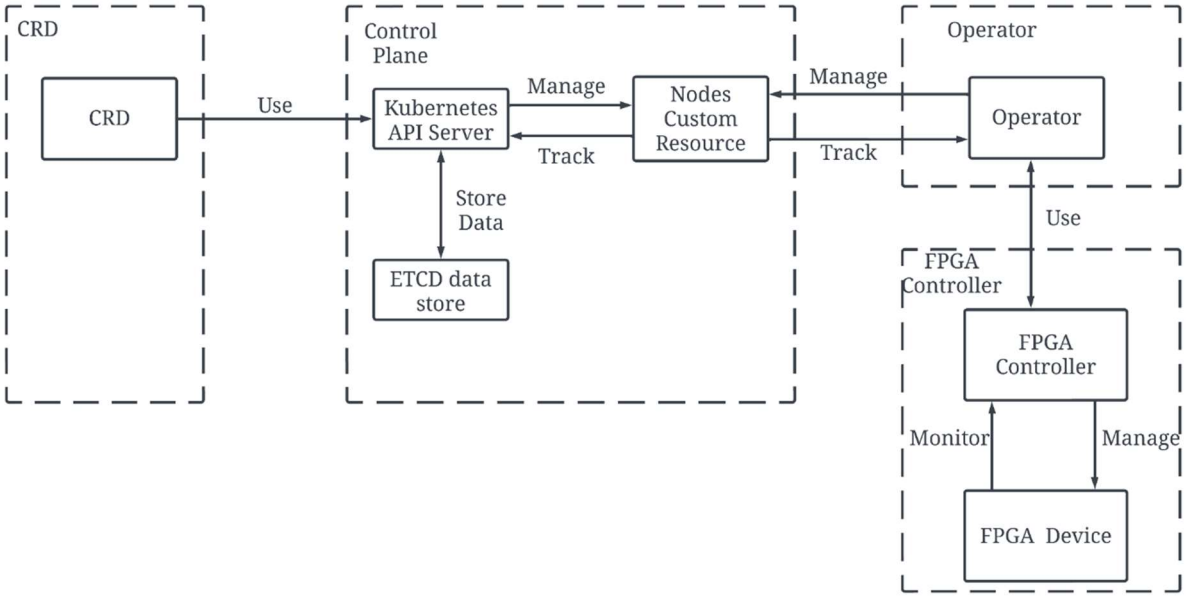

Figure 3. Schema of the interaction of system components

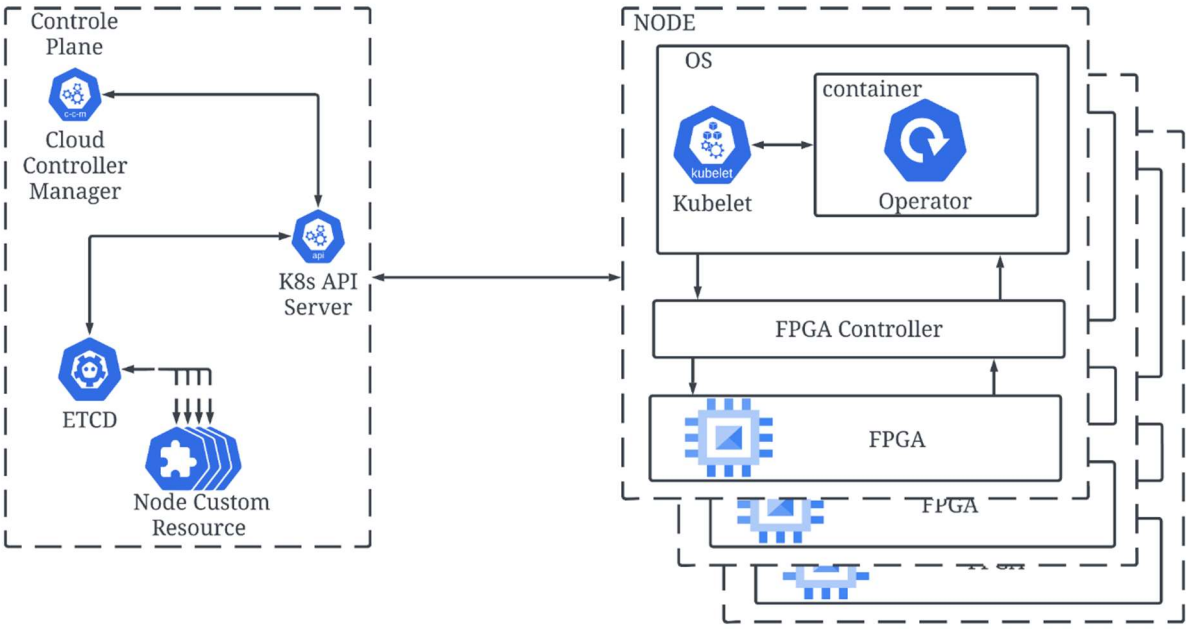

Figure 4. Structural diagram of a cluster with FPGA-based devices

B. OPERATOR STRUCTURE

To create an operator that describes and controls an FPGA-based device, it is needed to include the following components:

- Device Controller presents objects that control the logic of the controller's operation. The controller monitors and changes the state of the device, and also responds to requests from the outside world.
- DeviceCR is a user resource object that describes the idle state.
- DeviceSpec is an object that describes the reference state of the device, can change both externally and internally, depending on the implementation of the controller.
- DeviceStatus is an object that describes the current state of the device. It can be changed by device monitoring systems.

The operator is a design pattern and is only an abstraction that describes the principle of working with a user resource in the k8s system, and also contains the logic of responding to a change in the state of this resource. Fig. 5 shows the structural diagram of the operator that describes and controls the device based on the FPGA.

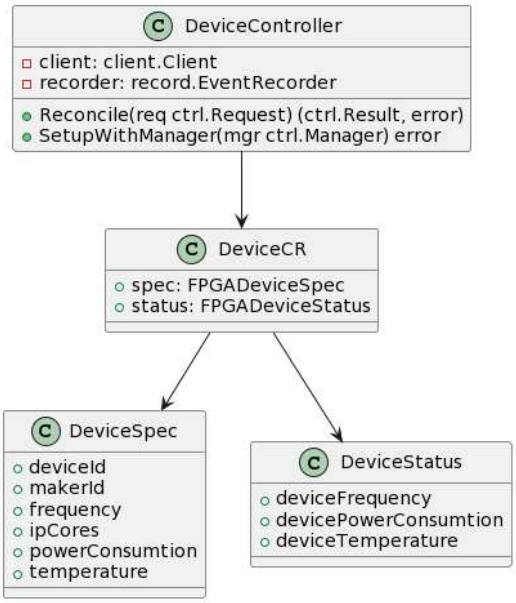

Figure 5. Structural Diagram of the Operator

C. CRD STRUCTURE

To create and register a resource in the system, it must first be described using the YAML language. Kubectl, which is used as a tool for registering new resources in the system, understands only this version of the description of resources in the system. Such resources are described and registered in the system once when the system starts its work.

Resources usually consist of two key characteristics: spec and status:

- spec is a set of system attributes that define the reference state of the device for normal operation.
- status is a set of device attributes that defines the current state of the latter.

To create a resource, it is necessary to determine which spec and status attributes this resource will have. To do this, it is necessary to determine which characteristics of the FPGA device system need to be monitored in the system.

Spec device attributes:

- deviceId is a unique device identifier in the system, most modules that will communicate with this device will use this identifier to designate the device as the end device in the message chain;
- makerId is the identifier of the manufacturer of the FPGA device, it can be Intel, Xilinx, Altera;
- frequency is reference clock frequency of the device, the unit of measurement is hertz (Hz);
- ipCores - a reference number of logical blocks that can be used to design a specialized processor using an FPGA device;
- powerConsumtion - reference amount of electricity consumed by the device for normal operation, the unit of measurement is W;
- temperature - reference temperature of the core of the device, unit of measurement C.

Status attributes of the device:

- deviceFrequency - the current clock frequency of the device;
- devicePowerConsumtion - the current level of device consumption;
- deviceTemparature - the current temperature level of the device.

Table 1 shows an example of defining custom resources for FPGA devices.

Table 1. Custom Resource Definition for FPGA device

Configuration Options	Meaning
apiVersion: apiextensions.k8s.io/v1	Specifies the version of the Kubernetes API
kind: CustomResourceDefinition	Specifies the type of Kubernetes object we are trying to create
metadata:	Contains information that helps us to uniquely identify our Kubernetes object
name: fpga-devices.device.com	Name of Resource
spec:	Describes the specifications or configuration details of the CustomResourceDefinition
group: device.com	Indicates the group to which the custom resource belongs
versions:	Specifies the supported versions of the custom resource
- name: v1	Defines the name of the version. In this case, it is v1
schema:	Specifies the schema definition of the custom resource
openAPIV3Schema:	Contains the OpenAPI v3 schema definition.
type: object	Defines the type of the schema, which is an object in this case
properties:	Specifies the properties or fields of the object
spec:	Specifies the desired state of our object
type: object	
properties:	Specifies the properties or fields of the object.
deviceId:	Specifies the deviceId field as a string
type: string	
makerId:	Specifies the makerId field as a string
type: string	
frequency:	Specifies the frequency field as an integer
type: integer	
ipCores:	Specifies the ipCores field as an integer.
type: integer	
powerConsumption:	Specifies the powerConsumption field as an integer.
type: integer	
temperature:	Specifies the temperature field as a number.
type: number	
replicas:	Specifies the replicas field as an integer.
type: integer	
status:	Defines the status field of the custom resource, which has its own set of properties
type: object	
properties:	Specifies the properties or fields of the object.
deviceFrequency:	Specifies the deviceFrequency field as an integer. Unit of measurement: hertz (Hz)
type: integer	
devicePowerConsumtion:	Specifies the devicePowerConsumtion field as an integer. Unit of measurement: watt (W)
type: integer	
deviceTemperature:	Specifies the deviceTemperature field as a number. Unit of measurement: Celsius (°C)
type: integer	

D. CRD INSTANCE CREATION

In order for resource objects to be registered in the system, an object of this resource must be created for them. The controller constantly monitors the state of the object and, using the reference model of this resource tries to bring the current state to the reference state. Table 2 describes how an object of the user type should be declared using markup in the YAML style.

Table 2. FPGA device resource declaration configuration

Configuration Options	Meaning
apiVersion: "device.com/v1"	Specifies the API version being used for this resource. In this case, it indicates that the resource follows the "device.com" group's API version 1.
kind: FpgaDevice	Defines the kind or type of the resource being defined. Here, it specifies that the resource is of kind "FpgaDevice".
metadata:	Contains metadata about the FpgaDevice resource.
name: xilinx-fpga	Specifies the name of the FpgaDevice resource. In this case, the name is set to "xilinx-fpga".
spec:	Describes the specification or configuration details of the FpgaDevice resource.
deviceId: "123123-452345-123-345-123"	Represents the deviceId property of the FpgaDevice resource.
makerId: "Xilinx"	Represents the makerId property of the FpgaDevice resource.
deviceFrequency: 340	Represents the working frequency property of the FpgaDevice resource. Unit of measurement: hertz (Hz)
ipCores: 190800	Represents the ipCores property of the FpgaDevice resource.
powerConsumption: 10	Represents the working powerConsumption property of the FpgaDevice resource. Unit of measurement: watt (W)
temperature: 65	Represents the temperature property of the FpgaDevice resource. Unit of measurement: Celsius (°C)

E. CONTROLLER CREATION

For the system to be able to respond to state changes, you need to register a controller that will monitor the change and perform the necessary actions to change the state of the devices. For example, let us take a change in the clock frequency of the device. If the clock frequency of the device is changed externally, then for these changes to be implemented on the device that uses the FPGA, the controller must send the necessary instructions to this device. In Listing 1, we can see an example of code that reacts to a change in the clock frequency and changes it for the device.

Listing 1

```
async fn reconcile(api: Api<FPGADevice>, name: &str) {
    let fpga_device = match api.get(name).await {
        Ok(fpga_device) => fpga_device,
        Err(e) => {
            println!("Failed to get FpgaDevice {}: {:?}",
                    name, e);
            return;
        }
    };
    let patch = serde_json::to_vec(&Patch
{
        op: "replace".to_owned(),
        path: "/status/device_frequency".to_owned(),
        value:
serde_json::to_value(fpga_device.spec.frequency).unwrap(),
    })
    .unwrap();
    let res = api
        .patch(name,&PatchParams::default(),
            serde_json::to_vec(&patch).unwrap(),
        )
        .await;

    if let Err(e) = res
{
        println!("Failed to patch FpgaDevice {}: {:?}", name,
e);
    }
}
```

F. RESULT

This section describes the results of building a conceptual model of the integration of FPGA-based devices with distribution using a system of modeled resources based on Kubernetes.

To understand the number of devices registered in the system, as well as their current statuses, you can use an integrated environment that displays the status of the entire cluster. Fig.6 shows an integrated environment that allows you to monitor and control the state of resources of FPGA-based devices in a distributed system.

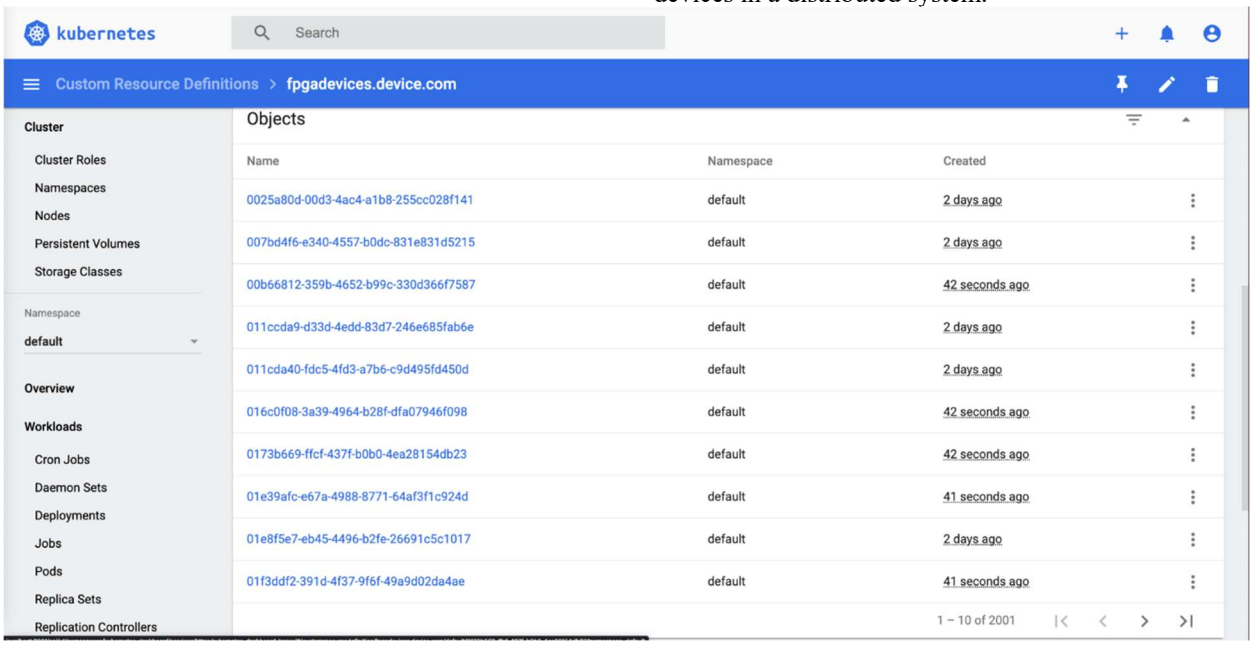

Figure 6. An integrated environment for monitoring and controlling the resources of FPGA-based devices

These resources can be used to configure and orchestrate non-Kubernetes components. The use of custom resources allows you to build a resource model of component management, where each component acts as a resource in the system, the characteristics of which are constantly changing.

V. CONCLUSION

A system for automatic control of distributed systems is considered, as well as the integration of FPGA-based devices into such a system. The expediency of using k8s system user resources for describing FPGA-based devices as k8s resources is analyzed.

In Section IV, a conceptual model for building a distributed system is proposed, which allows for the integration of FPGA-based devices into the system.

This model makes it possible to ensure high fault tolerance of the system due to the distribution of nodes. The most critical node is the Control Plane because it performs the function of an administrator. Such a node is usually maintained with several copies in the system to ensure system fault tolerance. If nodes stop working, it will not affect the operation of other components.

The provided model also allows for system extensibility. Since the system is distributed, the limit of system expansion is determined only by the performance of the Control Plane and its characteristics. At the same time, the poor characteristics of the Control Plane do not indicate that the system cannot be scaled, but only indicate that the response time of the system will be longer.

Compared to other possible solutions such as AWS IoT Service, an approach using K8S CRD gives the next advantages and novelty:

- Flexibility - k8s has an extensive ecosystem that allows integrating any type of device with different varieties of hardware resources.
- Integration - k8s is a highly extensible and modular platform, allowing integration of third-party software and tools for controlling and monitoring devices based on FPGA. Kubernetes provides a flexible framework that supports the plugging in of various components and extensions to enhance its functionality.
- Provider Agnostic - k8s is an open solution and is agnostic to any hardware/cloud provider and can be running on different clouds. AWS IoT is fully integrated with the AWS ecosystem and cannot work without it.
- Communication channel - k8s allows using a communication channel defined by the k8s API, because AWS uses another one (MQTT, LWM2M, etc.), which makes it more maintainable. Such communication is provided by K8S and does not need to be configured.

In summary, using AWS IoT for FPGA devices provides a comprehensive, managed solution with built-in device management capabilities and tight integration with the wider AWS ecosystem. It offers convenience and ease of use, especially if you plan to use other AWS Services. On the other hand, using Kubernetes CRDs provides more flexibility and control over the management and behavior of FPGA devices in a Kubernetes environment. This approach requires more effort to create and maintain the custom infrastructure and controllers but offers extensive customization options and benefits from the rich Kubernetes ecosystem.

In future studies, it is necessary to thoroughly investigate the fault tolerance of the system, as well as its scaling limit. It is necessary to conduct a study of the response time of nodes under high load, as well as the possibility of the system working in real-time.

References

[1] W. Wolf, *FPGA-Based System Design*, Prentice-Hall PTR, USA, 2004, 576 p.

[2] P. Käsgen, M. Messelka and M. Weinhardt, "HiPReP: High-performance reconfigurable processor – Architecture and compiler," *Proceedings of the 2021 IEEE 31st International Conference on Field-Programmable Logic and Applications (FPL)*, Dresden, Germany, 2021, pp. 380-381, https://doi.org/10.1109/fpl53798.2021.00074

[3] C. Zhao, C. Xiao and Y. Liu, "A real-time reconfigurable edge computing system in industrial Internet of Things based on FPGA," *Proceedings of the 2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA)*, Chengdu, China, 2021, pp. 480-485. https://doi.org/10.1109/iciea51954.2021.9516225

[4] A. Melnyk and V. Melnyk, "Remote synthesis of computer devices for FPGA-based IoT nodes," *Proceedings of the 2020 IEEE 10th International Conference on Advanced Computer Information Technologies (ACIT)*, Deggendorf, Germany, 2020, pp. 254-259, https://doi.org/10.1109/acit49673.2020.9208882

[5] A. Melnyk and V. Melnyk, "Specialized processors automatic design tools – The basis of self-configurable computer and cyber-physical systems," *Proceedings of the 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, Kyiv, Ukraine, 2019, pp. 326-331. https://doi.org/10.1109/atit49449.2019.9030481

[6] A. Melnyk, V. Melnyk and A. Kit, "UNIX-like operating system extension for real-time FPGA-based SCCS support," *Proceedings of the 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Bucharest, Romania, 2017, pp. 20-25, https://doi.org/10.1109/idaacs.2017.8095042

[7] F. P. Mahimai Don Bosco, E. Chitra, and S. Ryan Ebenezer, "A survey of low-latency IoT system using FPGA accelerator," *Journal of Physics: Conference Series*, vol. 1964, no. 6, p. 062014, 2021, https://doi.org/10.1088/1742-6596/1964/6/062014

[8] A. Shrivastava, D. Haripriya, Y. D. Borole, A. Nanoty, C. Singh, and D. Chauhan, "High performance FPGA based secured hardware model for IoT devices," *International Journal of System Assurance Engineering and Management*, vol. 13, no. S1, pp. 736–741, 2022, https://doi.org/10.1007/s13198-021-01605-x

[9] "Overview," Kubernetes. [Online]. Available at: https://kubernetes.io/docs/concepts/overview/

[10] V. Medel, R. Tolosana-Calasanz, J. Á. Bañares, U. Arronategui, and O. F. Rana, "Characterising resource management performance in Kubernetes," *Computers & Electrical Engineering*, vol. 68, pp. 286–297, 2018, https://doi.org/10.1016/j.compeleceng.2018.03.041

[11] Z. Kang, K. An, A. Gokhale and P. Pazandak, "A comprehensive performance evaluation of different Kubernetes CNI plugins for edge-based and containerized publish/subscribe applications," *Proceedings of the 2021 IEEE International Conference on Cloud Engineering (IC2E)*, San Francisco, CA, USA, 2021, pp. 31-42, https://doi.org/10.1109/ic2e52221.2021.00017

[12] J. Bartlett, "Getting started with Kubernetes," *Cloud Native Applications with Docker and Kubernetes*, pp. 53–60, 2022, [Online]. Available at: https://doi.org/10.1007/978-1-4842-8876-4_6

[13] N. Sabharwal and S. Kasiviswanathan, "Submit, orchestrate, and monitor jobs on a Kubernetes cluster," *Workload Automation Using HWA*, pp. 139–144, 2022, https://doi.org/10.1007/978-1-4842-8885-6_9

[14] R. Levensalor, "Give your edge an adrenaline boost: Using Kubernetes to orchestrate FPGAs and GPU," *CableLabs*, Jan. 28, 2020. [Online]. Available at: https://www.cablelabs.com/blog/edge-adrenaline-boost-kubernetes-orchestrate-fpgas-gpu

[15] V. Medel, R. Tolosana-Calasanz, J. Á. Bañares, U. Arronategui, and O. F. Rana, "Characterising resource management performance in Kubernetes," *Computers & Electrical Engineering*, vol. 68, pp. 286–297, 2018, https://doi.org/10.1016/j.compeleceng.2018.03.041

[16] "Overview of Cloud Native Security," Kubernetes. [Online]. Available at: https://kubernetes.io/docs/concepts/security/overview/

[17] X. Long, B. Liu, F. Jiang, Q. Zhang, and X. Zhi, "FPGA virtualization deployment based on Docker container technology," *Proceedings of the 2020 IEEE 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, Harbin, China, 2020, pp. 473-476, https://doi.org/10.1109/icmcce51767.2020.00109

[18] C. Bobda et al., "The future of FPGA acceleration in datacenters and the cloud," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 3, pp. 1–42, 2022, https://doi.org/10.1145/3506713

[19] J. Hoozemans, J. Peltenburg, F. Nonnemacher, A. Hadnagy, Z. Al-Ars, and H. P. Hofstee, "FPGA acceleration for big data analytics: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 30–47, 2021, https://doi.org/10.1109/mcas.2021.3071608

[20] V. Laxmi, C. S. Adiga, and S. V. Harish, "FPGA based reconfigurable computing systems: A new design approach – A review," *Advanced Materials Research*, vol. 403–408, pp. 4272–4278, 2011, https://doi.org/10.4028/www.scientific.net/amr.403-408.4272

[21] G. Stitt, F. Vahid, and S. Nematbakhsh, "Energy savings and speedups from partitioning critical software loops to hardware in embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 1, pp. 218–232, 2004, https://doi.org/10.1145/972627.972637

[22] Z. Wang, S. Zhang, B. He, and W. Zhang, "Melia: A MapReduce framework on OpenCL-based FPGAs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3547–3560, 2016, https://doi.org/10.1109/tpds.2016.2537805

[23] I. Bravo, P. Jiménez, M. Mazo, J. L. Lázaro, and E. Martín, "Architecture based on FPGA's for real-time image processing," *Reconfigurable Computing: Architectures and Applications*, 2006, pp. 152–157, https://doi.org/10.1007/11802839_21

[24] W. Chen et al., "FPGA-based parallel implementation of SURF algorithm," *Proceedings of the 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, Wuhan, China, 2016, pp. 308-315, https://doi.org/10.1109/icpads.2016.0049

[25] "What is AWS IoT? - *AWS IoT Core*. [Online]. Available at: https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

[26] "Custom Resources," Kubernetes. [Online]. Available at: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

[27] "Extend the Kubernetes API with CustomResourceDefinitions," Kubernetes. [Online]. Available at: https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/

[28] "Operator pattern," Kubernetes. [Online]. Available at: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

***MYKHAILO MAIDAN** is a PhD student in Computer Engineering Department at Lviv Polytechnic National University. Graduated from Lviv Polytechnic National University with the engineer degree in computer engineering in 2018. Working as an assistant and lecturer in Computer Engineering Department at Lviv Polytechnic National University. Participated in developing software for Wind Turbine industry, Charging station Software for USA customer. From 2017 working as Software Engineer with main technologies Embedded System and programming languages C/C++/Rust.*

***ANATOLIY O. MELNYK** has been a Head of the Department of Computer Engineering at Lviv Polytechnic National University since 1994 and a Head of the Department of Artificial Intelligence of John Paul II Catholic University of Lublin since 2018. He graduated from Lviv Polytechnic Institute with Engineer Degree in Computer Engineering in 1978. In 1985 he obtained his PhD degree in Computer Systems from Moscow Power Engineering Institute. In 1992, he received his DSc degree from the Institute of Modeling Problems in Power Engineering of the National Academy of Science of Ukraine. He was recognized for his outstanding contributions to high-performance computer systems design as a Fellow Scientific Researcher in 1988. He became a Professor of Computer Engineering in 1996. From 1982 to 1994 he was a Head of the Department of Signal Processing Systems at Lviv Radio Engineering Research Institute. From 1994 to 2008 he was a Scientific Director of the Institute of Measurement and Computer Technique at Lviv Polytechnic National University. From 1999 to 2009 he was a Dean of the Department of Computer and Information Technologies at the Institute of Business and Perspective Technologies. Since 2000 he has served as President and CEO of Intron ltd. He was also a professor at the Kielce University of Technology, Rzeszow University of Information Technology and Management, a visiting professor at University of Bielsko-Biala.*