Experimental Evaluation of Rule-Based Autonomic Computing Management Framework for Cloud-Native Applications

Joanna Kosińska, *Member, IEEE* and Krzysztof Zieliński, *Member, IEEE*

Abstract—The policy-based management paradigm in a flexible manner governs the system behavior. For Cloud-native applications, additionally, it simplifies the compliance with CI/CD objectives. Hence, the velocity of changes in requirements made at runtime does not influence the system implementation. Continuously the adjustments are integrated into the system on the fly. This paper evaluates the rule-based approach to representing policies in the context of Cloud-native applications. Deploying applications in orchestrated environments is one of the main principles of Cloud-native. Our approach represents the extension of the management characteristics that are available in current implementations of the orchestrators. The presented study also shows a general methodology for experimental evaluation of complex Cloud-native environments. We propose two categories of experiments. Both evaluate the rule-based approach. The first category evaluates the impacts of dynamic adjustment of resources in the context of the Cloud-native execution environment. The second category assesses the influence of the rule engine approach on the autonomic management process. Given the wide range of available experiments, we additionally assume that evaluation is performed from the point of view of the execution environment's resources. This approach tightly embraces the capabilities of the proposed solution realized by the AMoCNA system and demonstrates its usability.

Index Terms—Autonomic computing (AC), Cloud-native, resource management, policy-driven management, rule-based management, experimental evaluation

1 Introduction

Cloud-native [1], [2] is a concept which is changing our approach to developing, deploying and operating applications. While it brings clear advantages, definitely stated should be some negative aspects and difficulties.

To meet internal and external requirements [3], many organizations implement strong governance practices and controls. These rules ensure compliance with existing standards through policy-based management [4], [5], [6]. It is hard to avoid violating Service Level Agreement (SLA) requirements, especially when the demands are still growing. The resulting complication of basic management tasks has led researchers to seek novel ways of accomplishing these vital operations. One possible remedy is the Autonomic Computing (AC) paradigm [7], [8], [9], [10] in the context of Cloud-native environments.

High-level directives called policies usually include parameters of a SLA contract. They express the intended means of system management [11]. We propose a declarative policy management approach in Cloud-native execution environments to significantly reduce the burden of defining their enforced actions. Policy-based management aims to address the encountered obstacles. It represents a standard, default technique even in modern commercial systems (e.g., Microsoft SQL Server [12], Cisco Identity Services Engine [13], Motorola LTE System Management [14], etc.). An undeniable convenience is the ability to change system behavior without reimplementing it. Usage of policies can enhance the presets of managed elements at runtime while dynamically applying the changes without modifying the underlying implementation.

The proposed AMoCNA framework that we presented in [15] realizes the suggested recommendations. AMoCNA is an acronym for (**A**utonomic **M**anagement of **C**loud-**n**ative **A**pplications). We propose to achieve the autonomy features via declarative policies and to process these policies with rule engine support. This paper is a comprehensive evaluation of proposed rule-based management concepts.

The paper proposes a methodology that verifies principles leading to effective management of Cloud-native applications through an experimental study. The evaluation covers many areas of AMoCNA usage, each checking the effectiveness of the rule-based approach. The evaluation strategy requires a thorough understanding of Cloud-native environments and their current state of the art. Only detailed evaluation of all tiers which comprise the Cloud-native execution environment [15] can produce meaningful outcomes. The evaluation, based on experience drawn from using the AMoCNA system [16], assesses the quality of proposed concepts. Conducted experiments confirm that AMoCNA significantly enhances the autonomic management of CNApps. In particular by increasing the potential of the included

• *The authors are with the Faculty of Computer Science, Electronics and Telecommunications, Institute of Computer Science, AGH University of Science and Technology, 30-059 Krakow, Poland.*
E-mail: {kosinska, kz}@agh.edu.pl.

Manuscript received 12 Jan. 2021; revised 5 Mar. 2022; accepted 9 Mar. 2022. Date of publication 15 Mar. 2022; date of current version 10 Apr. 2023.
This work was supported by the funds assigned to AGH University of Science and Technology by the Polish Ministry of Science and Higher Education.
(Corresponding author: Joanna Kosińska.)
Digital Object Identifier no. 10.1109/TSC.2022.3159001

orchestrator. Cloud-native application (abbreviated as CNApp) is composed of communicating microservices running as containers. The orchestration process that manages the microservices is so tightly coupled with a CNApp that setting it up can be considered an integral part of the application. In our opinion, the paper should also consider AMoCNA's influence on the orchestrators.

Summarizing, the main contribution of this paper is the introduction of a general methodology for the experimental evaluation of complex Cloud-native environments. We propose two categories of experiments. Both evaluate the rule-based approach. The first category evaluates the impact of resources' dynamic adjustments, and the second assesses the influence on the autonomic management process.

We have chosen only those experiments that convincingly demonstrate AMoCNA's capabilities. In particular, selected experiments relate to horizontal and vertical scalability. The experiments also reveal that proper automation of Cloud-native environments may facilitate efficient allocation of resources.

The structure of this paper is as follows. The Related work section includes a brief description of policy-based management. It discusses some optimization techniques of the orchestrator operation. This section also compares AMoCNA with the management available in current clusters implementation. Based on this knowledge, the following section determines the requirements of a system for policy-driven infrastructure management for a Cloud-native application. The subsequent section divides the system into three management parts. It also provides a brief background addressing AMoCNA's rule-based management strength. The remainder of this paper evaluates AMoCNA. It proposes two categories of experiments that further are conducted. The proposed methodology requires to set up a dedicated testbed, governed by established SLA statements. Such constructed environment with installed example CNApp validates concepts underlying the AMoCNA framework. The final section concludes the paper.

2 Related Work

Many organizations research the structure of policy-based frameworks [18], [19], [20], with the most well-known approach jointly proposed by IETF and DMTF [21]. [22] proposes policy-based management. It elaborates in detail a security framework for resources in the cloud. Also, it presents an example of this framework implementation. However, the proposed model is too difficult for ordinary users. The work does not pay attention to the way policies are defined. Similar topic but related to different managed resources presents paper [23]. Proposed policy-based security architecture touches network aspects. The architecture derives from previously mentioned DMTF functional elements. But again, policy definition is in infancy. A CPVS (Cloud Policy Verification Service) proposed in [24] for simplicity of usage defines a Domain Specific Language (DSL) that allows creating XACML policies. IBM Research Center presented a work [25] that resulted in a proposed policy framework combined with the AC technology. Their findings constitute an interesting background to our research.

The Cloud-native application is treated here as a managed object, which utilizes resources (computing, network, storage) that are also indirectly managed. Acceptable system behavior depends on the specification of user requirements. The system needs tools that help acquire and represent these requirements (policies) and map them into lower-level actions [7]. Providing for the above aspects *policy* is defined as a high-level specification of goals and constraints, typically denoted as rules or utility functions.

The most common case is to use processing rules (rule engine) as a tool for implementing policy decisions point [26], [27], [28] (according to the IETF/DMTF Policy Architecture [21]). Rules express the system policies. Their processing [29] needs to enable easy modifications of policies, thus simplifying the development of policy management tools and policy storage in repositories. Moreover, the rule engine fulfills the objectives of the strategy pattern. At runtime, it selects actions that are to be enforced [30] depending on the processed policy. Support for specification and processing policies and runtime reconfiguration constitute the core set of rule engine capabilities, making them perfect tools to apply in Cloud-native environments. Therefore we propose to use rule engines as building blocks of our concepts in the context of Cloud-native applications [15]. Their principal role in these environments is to aid fulfillment of policy-based management. In [15] we discuss many Cloud-native resource management tools, each of them enhancing the execution environment. In [31] a self-management architecture is even proposed; however, to the best of our knowledge, none of the existing solutions involve rule-based management to process the declared policies.

Optimization in a Cloud-native environment regards diverse vital tasks. These include techniques on cloud scheduler configuration, load balancing [32] or sustainability to bursts in traffic [33], etc. These themes are so vast that they need separate studies. However, the most obvious is the resource utilization [34], [35] that management we included in section 6. Also, the developed in work [36] AdaptiveConfig runtime configurator that automatically adapts to the changes in workload use a rule engine to compare the performance of different configurations in the cloud. However, the constructed Facts consider only the characteristics of the workload and available resources. In turn, we propose to reason over observations related to the whole environment. The same topic, but with AI technique, is proposed in [37].

All the cited work optimize at runtime the orchestrator's operation. However, our method is simple to use by ordinary users. The declarative policy approach that uses DSL, gives another significant benefit. The AMoCNA potential over the mentioned research lies in the possibility of declaring many policies at once. They can touch, at the same time couple of aspects. Policies can be composed on the fly at runtime. With AMoCNA, it becomes possible to declare any management action.

CNApp represents a graph of communicating microservices [38]. Microservices run as containers in an orchestrated environment. Cloud-native paradigm tightly couples containers with their orchestrators. Table 1 presents key advantages of AMoCNA management compared to management currently available in Kubernetes (AMoCNA successfully

TABLE 1
Comparison of Management Characteristics Offered by Proposed AMoCNA Framework and the Current Orchestrators' Implementations (On the Example of Kubernetes [17])

Feature	AMoCNA	Orchestrator
Capabilities	Orchestrator's capabilities + their enhancements.	Determined set.
The simplicity of usage	Feasible for ordinary users.	Meant for suitably experienced engineers.
Encapsulation	The burden of CNApp management and its internal logic is hidden from users.	The importance of hands-on experience and academic training while operating a cluster is required.
Centralized management	Management processes are triggered from a single secure location, effectively protecting against policy chaos.	Management directives are stored in multiple places (each cluster has its configuration), leading to duplicates or conflicts.
Policy ordering	Policies are decorated with a parameter indicating their priority.	Management actions are triggered as they are defined. Orchestrator tries to achieve the desired state without concerning users' intention.
Separation of concerns (SoC)	Policies is categorized, addressing separate concerns.	The management issues are categorized only by the controllers (e.g. within VPA or Cluster Autoscaler). Moreover, their configuration is a mixture of diverse data.
Linkage of preconditions	Preconditions are gathered from different, often unconnected controllers or plugins and exposed as a single requirement.	Preconditions are valid within given controller's scope.

used also Docker Swarm as an orchestrator). Distinguished in this Table, features and their values allow classifying the declarative approach to policy management as a right choice to achieve autonomic capabilities of a Cloud-native application. Such an attitude improves the quality of resource management offered by existing orchestrators.

Support for policy specification, processing, and runtime reconfiguration makes AMoCNA an essential tool in Cloud-native environments. The following sections address this topic.

3 Requirements of Policy-Driven Infrastructure

A policy is a set of restrictions imposed on all possible forms of system behavior in a way that the result is a subset of acceptable system behaviors [39]. We propose to process declared policies via a rule engine. Our concept of the MRE-K loop (Monitor - Rule Engine - Execute over a shared Knowledge, explained in [15]) enables enforcement of declared management policies in the context of Cloud-native environments. MRE-K loop is an adaptation, with appropriate modifications, of the MAPE-K loop in the context of Cloud-native operation. MRE-K loop uses a rule engine as a means of enforcing management policies.

The main requirements of the rule-based AC management framework involved at the system design stage in the context of Cloud-native applications include:

- Low resource management overhead – the overload generated through the operation of the framework's components should be as low as possible. At the same time, the overhead generated by resource control and management techniques should also remain low. This requirement determines the selection of observation techniques [40], [41], [42].
- The possibility to specify the application's demand for resources (particularly application level bandwidth guarantees [43]) – the specification should describe the configuration of computational resources influencing the effectiveness of processing operations. The requirements can change dynamically at runtime.
- Reacting to events concerning resource allocation – the environment has to discover changes in requirements for resources and changes in the level of resource availability on each layer of the Cloud-native stack [44]. Hence, it should be possible to discover cases of inefficient resource allocation resulting from their partial use or lack of required resources.
- Transforming system state to the form required by the decision module – apart from transforming information about the application or the state of environment components to a proper structure, it is necessary to update such transformation regularly, or as a reaction to significant events.
- Flexible definition of system operation rules – in the Resource Management System (RMS) a crucial role falls to the decision module that performs resource allocation modifications. These decisions should be taken based on the current state of the application, its requirements, and restrictions established based on system policies. Decision rules should enable a flexible definition of all aspects of the resource allocation subsystem and provide a way to adapt to various application requirements.
- Undertaking decisions – decisions result from policy configuration based on system state information. Then they are delegated to proper components. These decisions usually involve changes in resource allocation parameters and the execution of operations related to the management of the CNApp lifecycle.

The proposed design and development of a framework for Cloud-native applications management involves several challenges. The main one is undoubtedly the design and deployment of a rule-based closed loop implementing the concepts of self-management and adaptability [45].

4 Organization of the Rule-Based Closed Loop

The proposed AMoCNA framework has been thoroughly described in [16] and is available at AGH University's Main Library. Just for the record, AMoCNA framework (Fig. 1) is

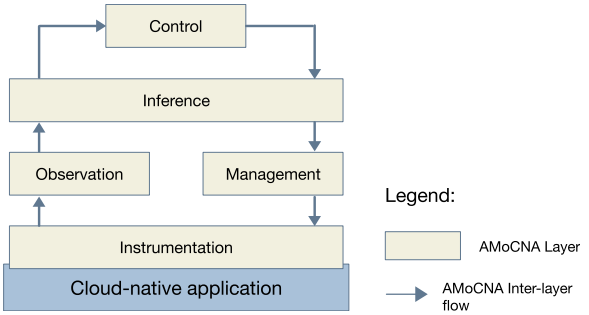


Fig. 1. Simplified view of AMoCNA architecture.

built of five layers that realize the Cloud-native autonomic element's (this concept is introduced in [15]) functions. The presented layers realize the capabilities of a closed loop of feedback control [46] (in Fig. 1 it is a set of consecutive blue arrows). Moreover, the AMoCNA framework is compliant with the model of AS3 element presented in [45]. As depicted in Fig. 1 these layers are (starting from the bottom):

- Instrumentation Layer – This is the lowest AMoCNA layer and is the only layer having direct access to the managed CNApp. Its responsibility is to gather data from different sensors (a term introduced in AC). In Fig. 1 it is the beginning of the loop. The effectors (a term introduced in AC) are closing the loop. Through them, the layer executes management actions. These actions reduce to actuating the resources.
- Observation Layer – The management of a CNApp connects with its observability. It is natural and required that the CNApp is equipped with sensors to start any control of an application. The sensors enable gathering observability-related information and the mechanisms enabling further processing of collected data. The following layer realizes the component's observability requirement.
- Inference Layer – This layer realizes primary AMoCNA functionalities. The functionalities correspond to retrieval of observation data, their transformation into a format accepted by the rule engine, namely into facts, reasoning over the facts, and producing management actions based on declared policies and prior results.
- Control Layer – This layer accomplishes two essential AMoCNA tasks. First, it drives the overall management process of CNApp, based on declared management policies. And second, it ensures the compatibility between contracted SLA and enforced management actions.
- Management Layer – This layer indirectly complements the management loop. It operates at the same level as the Observation Layer. Based on produced earlier management actions, this layer triggers the reconfiguration of the Cloud-native application.

The management process of a CNApp (depicted in Fig. 2) divides into three logical parts. These are as follows: (i) Cloud-native Execution Environment Part, (ii) Cloud-native Autonomic Computing Part, and (iii) Cloud-native Management Policies Part. Part names are self-explanatory and indicate the main functionality of each.

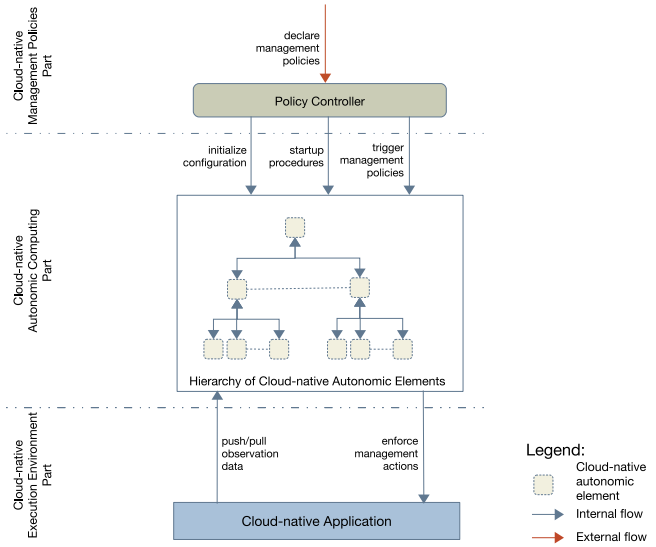


Fig. 2. High-level view of AMoCNA management process design.

AMoCNA distinguishes two types of actions: (i) those triggered internally by Cloud-native autonomic elements (as a result of autonomic capabilities), and (ii) actions that are declared externally to the system. The latter type, comprising declarative management policies (the topmost entity in Fig. 2), drives the overall process of setting up a hierarchy of Cloud-native autonomic elements. However, the principal aim of these policies is to manage the operation of AMoCNA. They influence the operation of the Policy Controller entity (depicted in Fig. 2), which serves as a wrapper for all components responsible for policy processing. The Policy Controller performs three categories of actions (depicted as an internal flow) against the AMoCNA system. The type of actions refers to those run before, during, and after AMoCNA startup. First, the initialization configuration action triggers proper actions before AMoCNA startup. These actions set up the Cloud-native Execution Environment for AMoCNA usage. They reduce to specifying directives that contain the initial configuration and then passing them to the Cloud-native platform. The Cloud-native Execution Environment is ready, and the Policy Controller proceeds with AMoCNA startup (middle arrow). This step constructs the hierarchy of Cloud-native autonomic elements.

Finally, Policy Controller triggers management policies. Policies are requested by the administrator or by AMoCNA clients. For human readability, policies are expressed in a Domain Specific Language (DSL) using a graphical notation (e.g., in the Eclipse Modeling Framework [47]), decision tables, etc. In most cases, AMoCNA clients would declare policies triggering the actions of upper layers of the Cloud-native Execution Environment using DSL and/or natural language constructs. These policies are meta-policies and need mapping to include actions, pointing to specific resources that can execute them. An example policy is in listing Policy 1.

Policies ensure that the Cloud-native execution environment is configured following defined rules. CNApp begins to run. Runtime reconfiguration in the cluster triggers the orchestrator controllers to enforce management actions, including mentioned load balancing.

Policy 1. Policy That Triggers Scaling up of a Microservice (Meta-Policy)

Require: RT – [milliseconds] (response time) duration of the execution of the given microservice.
Ensure: Load-balancing of CNApp.

```
if RT > 20 then
  trigger scaling up
end if
```

5 Evaluation Methodology

AMoCNA evaluation aims to confirm that the proposed rule-based framework significantly enhances the autonomic management of CNApps. Particularly by increasing the potential of the Kubernetes orchestrator. Regardless of the level of available resources, AMoCNA helps to meet the agreed SLA. Particular policies (rules) need to be declared to fulfill this capability.

A comprehensive and objective evaluation of concepts underlying AMoCNA is performed based on two categories of experiments:

Experiment 1 The impacts of dynamic adjustments of the Cloud-native execution environment's resources.

The execution environment describes the context in which the execution of a CNApp takes place. [15] proposes its model. This model follows that its further advancement should refer to distinguished layers. The essential aim of this experiment is to show the impact of management in the Cloud-native execution environment. As an example of management action, the executed test scenarios reveal the horizontal and vertical scaling of the Cloud-native execution environment at runtime. These experiments that adjust various computing resources show another approach of augmenting Kubernetes cluster scheduling capabilities.

Experiment 2 Influence of the rule engine approach upon autonomic management process.

This category of experiments reveals the effectiveness of the rule engine for autonomic management purposes. The conducted experiments uncover the weak and strong points of including a rule engine in Cloud-native execution environments. To research the potential of this approach, we propose that the evaluation consists of (i) evaluation of the duration of translation of metrics, (ii) assessment of the system response time and (iii) estimation of system behavior under fluctuating SLA.

A necessary complement to the above evaluations contains paper [15]. In a couple of experiments, it evaluates the system overhead. The experiments estimate the imposed load of infrastructure resources generated by deploying proposed autonomic management solutions. Regardless of the quantity (that essentially is not high) of introduced additional load, gained valuable performance improvements of Cloud-native application allow to state that the described solution is an effective enhancement of Cloud-native environments.

A constructed dedicated environment depicted in Fig. 3 comprehensively evaluates the concepts underlying the AMoCNA framework. The environment consisted of a

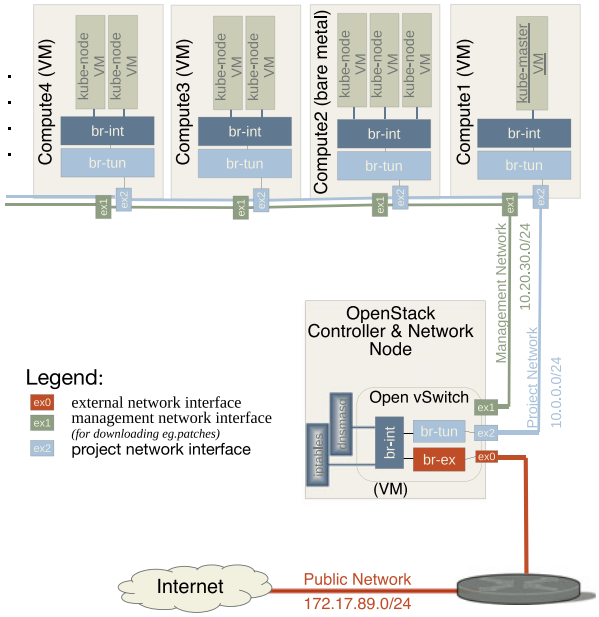


Fig. 3. Network and Service layout of the evaluation infrastructure.

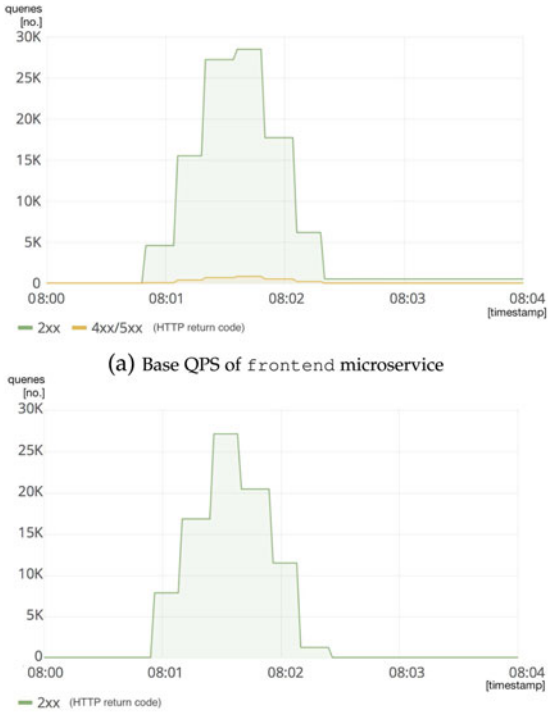
seven-member Kubernetes cluster. The cluster initially consisted of one master, which subsequently joined six additional nodes to carry the workload. The constructed testbed constitutes a baseline for further experiments. Its initial configuration is described below.

To make the evaluation more realistic, a specific CNApp is used. Weaveworks [48] and Container Solutions [49] have developed a microservice-based demo application named Sock Shop [50] (abbreviated here as SS). The SS Cloud-native application consists of 14 microservices, distributed as Docker containers or incorporated in code. SS was tested with many simultaneous users. A load simulation (also provided with the microservices demo) with the Locust [51] tool tested SS against a specified number of clients and requests. Its configuration is given in Table 3. Results of this simulation (of only two microservices) are depicted in Fig. 4. Depicted graphs show that the highest load is generated after 1.5 minutes. Correspondingly after that time, the noticed increased load starts to generate high latency (Fig. 5).

Experiments reveal that the highest latency (i.e., the interval measured between the input and output of a microservice) occurs in `frontend` and `orders` microservices (Table 2). Therefore these two represent targets for further performance improvements and are presented in most figures. We propose that base metric values for all experiments is the defined SLA:

- The `orders` microservice latency is lower than 230 ms.
- The `frontend` microservice latency is lower than 850 ms.
- Every node has CPU utilization lower than 50%.

SLA thresholds are defined as lower values than observed in many experiments. However, they are still achievable and present a compelling area for further improvements. In the case of latency, values should remain below the 99th percentile and derive from graphs depicted in Fig. 5.



(a) Base QPS of `frontend` microservice

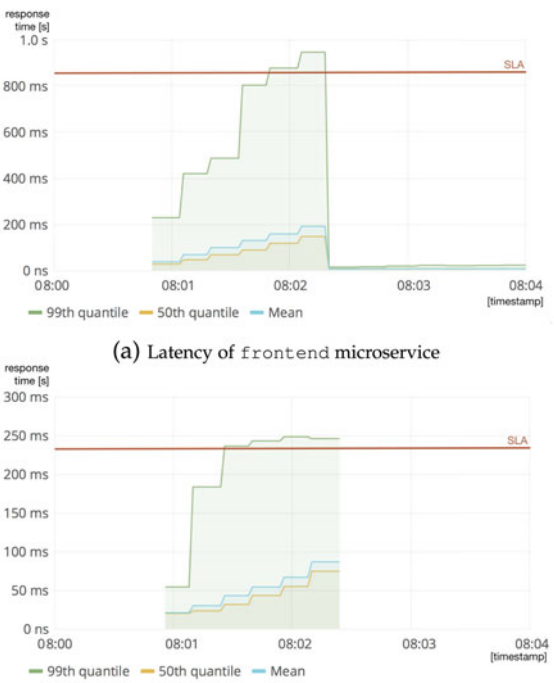
(b) Base QPS of `orders` microservice

Conducted experiments began at 8:00 a.m. and constitute a chain hierarchy. The initial state of each corresponds to the state left by the previous experiment. This dependency is visible on the vertical axis (the time of given experiment) for successive experiments (it can be noticed that the time elapses).

Fig. 4. Queries per Second (QPS) simulation with load tests.

The presented research environment represents an ideal situation where hardware is allocated only for the CNApp. In a real-world scenario, we must assume that diverse workloads will run simultaneously and share the resources. The simulation hypothesizes that the test scenarios experience a significantly increased workload. For this purpose, we use the docker-stress [52] container. It can generate CPU load, memory load, I/O load, and disk load. By using it, we can simulate increases in resource consumption and therefore failures to meet SLA commitments regarding CPU utilization. Table 4 shows the parameters of the stress container set in the presented scenarios. Knowledge regarding the placement of microservices (Table 5) enables running the stress container on a specific node while increasing to the maximum that node's resource consumption. This type of execution is only meaningful at the node level because the application runs in a separate container and does not influence the operation of other containers.

Table 5 presents the initial configuration of SS microservices among Kubernetes cluster nodes. At runtime, this configuration changes because of Kubernetes dynamic provisioning and runtime reconfiguration management capabilities. More importantly, because of the potential of AMoCNA (shown practically in further experiments) to autonomously manage the Cloud-native application.



(a) Latency of `frontend` microservice

(b) Latency of `orders` microservice

The annotated SLA red lines, depicted in both subfigures, are strictly SLA threshold values.

Fig. 5. Latency of SS for aforementioned (base) configuration.

TABLE 2
The Peek Values of SS Microservices Latency

Microservice	Latency [ms]
catalogue	61
cart	82
orders	250
payment	5
shipping	10
user	41
frontend	980

Two microservices with highest latency are highlighted.

6 Experiment 1 - The Impacts of Dynamic Adjustment of the Resources

This experiment aims to present how AMoCNA increases the potential of a Cloud-native execution environment by adjusting its resources at runtime.

In this scenario, we distinguish two aspects of autonomic management, resulting in a range of subexperiments. As a whole, the experiments concern:

1) *Autonomic management in the Containerization Layer* – underscores the importance of autonomic changes in resource requests and limits.

TABLE 3
Configuration of Load Tests

Key	Value
No clients	3000
No requests (total)	15 000
Duration of the experiment	10 minutes
Distribution of clients	5 clients/second
Number of orders	16 831
Number of logins	24 514

TABLE 4
Configuration of `stress` container

Parameter	Meaning	Value
placement	Compute machine that runs the `stress`	worker-2
cpu	No. of spawned workers spinning on sqrt()	2
vm	No of. spawned spinning on malloc()/free()	2
vm-bytes	No. of malloc bytes per vm worker	512M
timeout	Duration of `stress` container	120s

2) *Autonomic management in the Infrastructure Layer* – mainly highlights the importance of autonomic horizontal and vertical scalability.

Because the present research does not cover Continuous Integration/Continuous Delivery (CI/CD) objectives, we cannot distinguish the third aspect that covers management in the Application Layer.

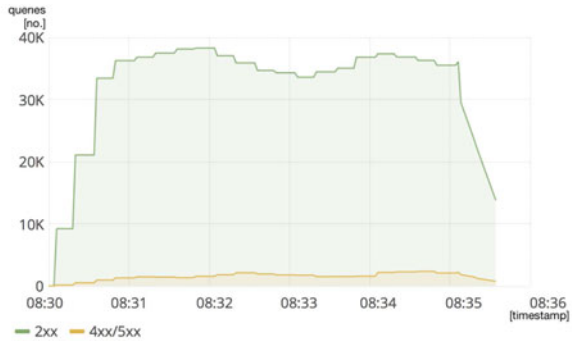
Autonomic Management in the Containerization Layer. The principal paradigm of Cloud-native operation is that containerized entities are managed in an orchestrated environment. The characteristics of the orchestrator dictate the behavior and capabilities of the Cloud-native execution environment embodied in deployed objects. Autonomic management in the Containerization Layer regards runtime modifications of the current configuration. This type addresses the following experiment in which declared management actions cause scaling of a Cloud-native execution environment.

Kubernetes orchestrator provides two products for scaling, namely, Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA). Both scale available resources for containers. HPA often uses CPU measurements to trigger the scaling of a pod, whereas VPA allocates more (or less) CPU or memory to existing pods. Similar to VPA, in the presented experiment, AMoCNA's exemplary declared policy adjusts resource requests based on the observed latency correlated with resource usage.

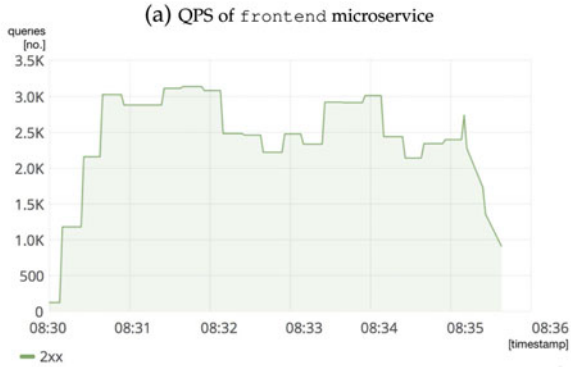
Observations of QPS enable to classify the Cloud-native application as a high- or low-traffic system and, with a properly declared management policy that triggers reconfiguration, to handle more load while maintaining adherence to the SLA. This behavior is shown in the present experiment that automatically adjusts the CPU limits and requests of a pod. Graphs depicted in Fig. 6 qualify the SS as a high-traffic system and enable us to assume that the system is overloaded. Therefore AMoCNA proceeds some improvements in the Cloud-native execution environment. These improvements concern enhancement of the Containerization Layer that results from the execution of declared rule. The declared management policy triggers autonomic management in a way which is similar to VPA (Policy 2).

TABLE 5
Initial Deployment of Microservices

Node	Microservice
worker-1	orders
worker-2	frontend
worker-3	payment
worker-4	shipping
worker-5	cart, catalogue
worker-6	user



(a) QPS of `frontend` microservice



(b) QPS of `orders` microservice

In given scenario, compared to base configuration that values are included in Table 3 we assume that the number of clients does not change, but the number of produced requests increases 10 times.

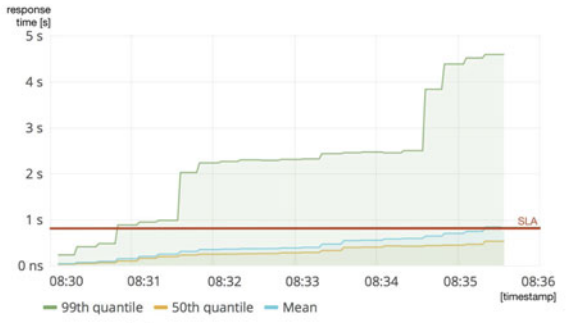
Fig. 6. QPS observations during load tests configured to produce 3000 clients and 150 000 requests.

Policy 2. Check Whether it is Necessary to Increase CPU Request

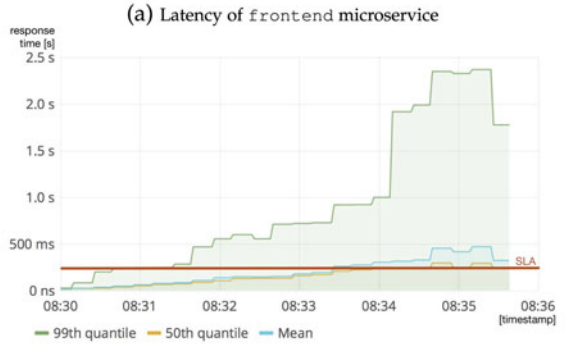
Require: RT – [milliseconds] (current response time) duration of the execution of a given microservice.
SLA_RT – [milliseconds] (acceptable response time) duration of the execution of a given microservice.
Ensure: Initialize the increase of Pod's CPU request.
if $RT > SLA\_RT$ **then**
`create rule engine fact (`*fact*`)`
for *fact* **do**
`increase CPU Request`
It is important to check the number of successive neighbor facts, hence to check whether SLA is violated occasionally or is SLA violated repeatedly in a period and only in this case take preventive actions.
end for
end if

Fig. 8 depicts latency of `frontend` microservice[1] after AMoCNA executes Policy 2. Comparing the gained results with Fig. 7a, one can notice that CPU request modifications have influenced the pod's latency, decreasing it almost by 50% (from about 5s to 2.5 s depicted in Y-axis).

1. To remind a microservice is a collection of containers. Particularly the relation can be 1:1 as in SS. The `front-end` Pod is directly mapped to `frontend` microservice.



(a) Latency of `frontend` microservice



(b) Latency of `orders` microservice

The timestamp on all graphs depicted in Fig. 6 and Fig. 7 regards the same experiment hence we can deduce that observed QPS loads cause that the SLA contract is significantly violated.

Fig. 7. Latency of SS for load depicted in Fig. 6.

Fig. 9b depicts CPU usage of frontend microservice also during the present experiment[2]. AMoCNA executes Policy 2 twice what is denoted by two trigger points. After a while, from the trigger point, the incline decreases. And only if the same policy executes two times significantly improves CPU usage. Graph 9a is presented in this paper only as a comparison with graph 9b to emphasize the AMoCNA improvements. Meanwhile, optimizing the pod's latency, the CPU usage of `frontend` microservice is also optimized. The CPU usage reduces from 1.0 core to 0.5 core.

The drawback (even in HPA and VPA solutions) is the necessity to create a new pod while killing the original one leading to microservice downtime. In the present scenario, AMoCNA deletes the `front-end` pod twice (depicted in Fig. 8 as two white areas). Each time Kubernetes orchestrator recreates the pod with new settings. At that time `frontend` microservice is not processing any requests.

The result achieved by AMoCNA is similar to the outcome of Kubernetes VPA controller actions. However, Kubernetes does not provide vertical autoscaling functionality out of the box. An additional controller that has to be installed and configured enables this feature. AMoCNA also must be installed and configured, but its usage is straightforward and convenient for ordinary users. The next difference is in the implementation mechanism that we propose to accomplish by means of a rule engine.

This experiment points to the conclusion that if we execute only a single policy, this does not necessarily solve all arisen problems. The SLA contract, despite significant improvements, can still be violated. This is depicted in Fig. 8.

2. Because the timestamp in both Figs. 8 and 9b is the same.

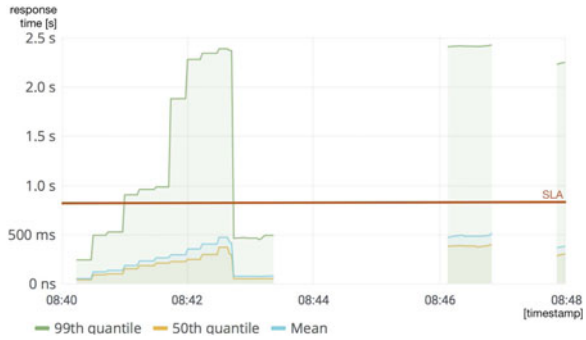
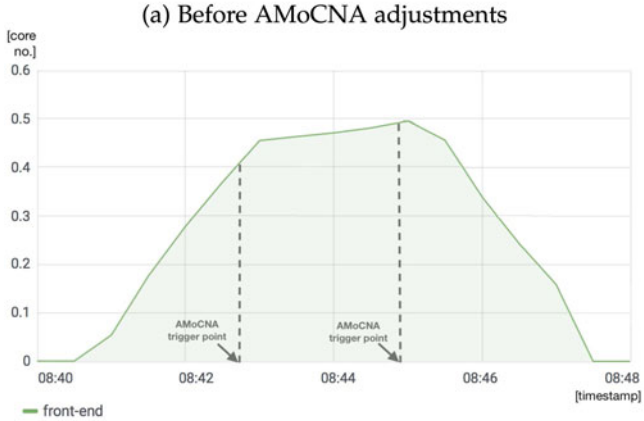


Fig. 8. Latency of `frontend` microservice after AMoCNA adjustments.

Autonomic Management in the Infrastructure Layer. An emerging area of research in autonomic management of CNApps is vertical and horizontal scalability. In the case of the Infrastructure Layer, scalability implies changes in the number of infrastructure resources allocated and available in the given cluster. The present experiment demonstrates horizontal scalability (cluster scaling) by increasing the number of Kubernetes nodes.

To meet the SLA criterion – Every node has CPU Utilization lower than 50% – additionally, it is necessary to trigger

(a) Before AMoCNA adjustments



(b) After AMoCNA adjustments

The initial experiment configuration sets every Pod CPU request to 0.1 (measured in CPU units). There are no CPU utilization limits set. Successive AMoCNA trigger points caused by executing Policy 2 increase Pods CPU request by 0.1 core.

Proposed CPU request values are prepared on the basis of tests and good practices among microservice-based applications.

Fig. 9. CPU usage of `frontend` microservice before and after AMoCNA adjustments.

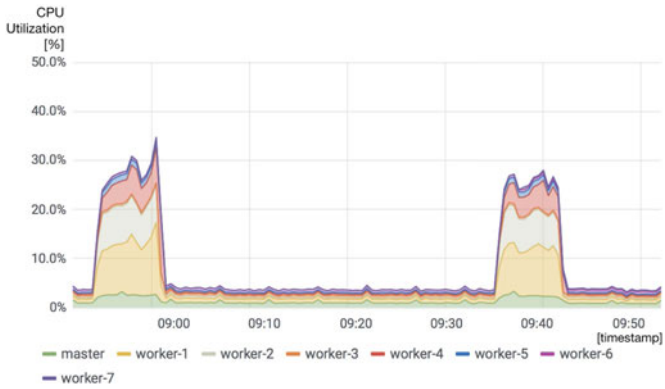


Fig. 10. CPU Utilization of the cluster.

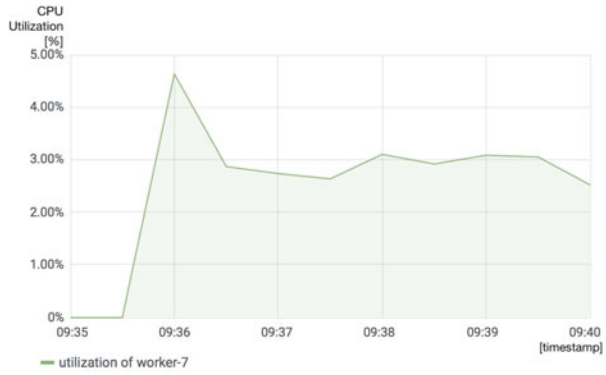


Fig. 11. CPU Utilization of `worker-7` node.

appropriate actions in the Infrastructure Layer. Kubernetes provides a Cluster Autoscaler (as an additional component) controller. It automatically adjusts the cluster size based on one of the preconditions: (i) there are pods awaiting to be scheduled, but the available cluster resources are insufficient, and (ii) there are underutilized nodes in the cluster. AMoCNA augments these possibilities, and instead of taking into account only the resource requests and limits, the decision is made based on observations of actual resource utilization.

The result of AMoCNA improvement is shown in Fig. 10.

The first peak denotes the usual SS run. The second peak indicates the same load but after AMoCNA optimizations. The optimizations resulted in adding a new node (`worker-7` in this case) to the cluster. After joining another node to the cluster, the total utilization automatically declines. In this case, the utilization did not decrease significantly. Nevertheless, the new node is at first strongly underutilized (depicted in Fig. 11). The existing workload in the cluster does not immediately migrate to the new node. This reflects the configuration of the Kubernetes orchestrator.

Obtained in this subexperiment results are satisfactory; however, they suggest that AMoCNA does not always offer noticeable improvements in a CNApp. Other factors, such as the configuration of the Kubernetes scheduler, play a crucial role.

Discussion of Results. Autonomic management of Cloud-native applications typically involves some minor or major improvements of the Cloud-native execution environment. These include adjustments in one of its layers. A union of adjustments in multiple layers is also possible, considerably enlarging the scope of autonomic management. At its core, AMoCNA does not support direct Cloud-native application code modifications. It follows the management possibility only in the Infrastructure and Containerization Layers. The presented experiments show the dynamic adjustment of the Cloud-native execution environment in those layers. The outcomes are strongly dependent on the declared management policies, yet the overall effect of the presented modifications is favorable. AMoCNA enables different policies to coexist, suitably prioritizing them in the rule engine. This feature facilitates parallel adjustments in different layers, which further strengthens the results of dynamic management.

The experiment that involves increases in the pod's CPU requests highlights the benefits of operating in a control loop, whereas the experiments focusing on different layers indicate advantages of combining features provided by various controllers. The presented experiments reveal that, through the proper definition of policies, AMoCNA integrates the capabilities of both controllers – VPA and Cluster Autoscaler. Finally, AMoCNA lets to take decisions at runtime. Hence it supports runtime modifications and reconfigurations of the Cloud-native execution environment. These changes enable to achieve the contracted SLA.

7 Experiment 2 - Influence of the Rule Engine Approach Upon Autonomic Management Process

The intent behind the presented experiment is to understand the proposed MRE-K loop. MRE-K loop improves the processing performance of the declared management policies. This experiment reveals the effectiveness of such an approach.

To meet the stated objectives, the evaluation conforms to the following scheme:

- Realistic appraisal of the duration of translation of metrics.
- Assessment of the system response time.
- Estimation of system behavior under dynamic change of SLA.

The proposed scheme implies the types of experiments. The first two items relate to the duration of brokering between monitoring and reasoning controllers. During these experiments, we turned off the autonomic management capabilities of AMoCNA as their effects are less critical in this particular case. In turn, the evaluation of the third item is dependent on the declared management policies, which reflect modifications of the SLA. Subsequent subsections present the pointed scheme of experiments.

Realistic Appraisal of the Duration of Translation of Metrics. The monitoring and reasoning controllers use different formats to represent data (e.g., monitoring system metric format versus rule engine facts). These data need translation to benefit from cooperation between both controllers. Fig. 12

This node is alive since 09:35:34. After that time its CPU utilization rises from 0% to achieve its peak of 4.5% and then drops to around 3% (these low values suggest underutilization of the `worker-7` node.). The peak is noticed because of the installation of `kube-system` pods, namely `kube-proxy` and `kube-flannel` bound to network communication. Additionally, the pods required by AMoCNA are also installed. This concerns the `node-exporter` pod which provides the monitoring features of a node.

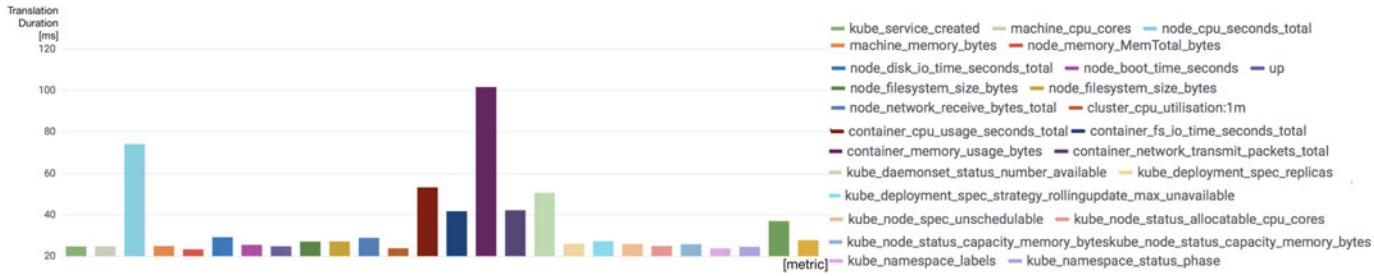


Fig. 12. Duration of translating individual metrics configured in the current AMoCNA scenario.

depicts the total time consumed while translating individual metrics configured in the current AMoCNA scenario. The difference in the value of particular measurements mainly relates to the number of time series. The time series needs translation. For two metrics:`container_memory_usage_bytes` and `node_cpu_seconds_total` it significantly differs from the rest. Notably, the former metric has 114 time series while the latter has 7 (the relation between the metric and its time series illustrates the following example: for the `node_name` metric and seven nodes in the cluster, there are seven time series, each related to one individual node).

Assessment of the System Response Time. [15] describes this subexperiment and widely discusses the overhead introduced by AMoCNA. The obtained results conclusively prove that, in the set configuration of AMoCNA, its latency and response time is relatively low, definitely below 2 seconds. The evaluation shows that the AMoCNA response time is closely dependent on the number of collected metrics and is also dependent on the number of defined rules.

Evaluation of System Behavior Under Dynamic Change of SLA. This part of the experiment evaluates the system's response to dynamic change of SLA observed at runtime. It can occur in two situations. Either the changes are manual, or they result from autonomic management actions. The purpose of the presented experiment is to evaluate the quality of adaptation mechanisms and, most importantly, to determine to what extent the new SLA definition is obeyed. It is not crucial whether the SLA values increase or decrease. Any change in the declared values that violates SLA contract triggers AMoCNA actions.

Fig. 13. CPU Utilization of `worker-2` and `worker-7` nodes after dynamic change of SLA.

Let remind that the SS application is under load according to values listed in Table 3. As a result of prior adjustments (mainly experiments involving autonomic management in the Containerization and Infrastructure layers), the SLA is not violated. AMoCNA does not trigger any management actions. After some time, one of the SLA statements changes in the following way: every node has *CPU Utilization lower than* 80%. Defining the SLA values as global variables causes that it is not necessary to declare new policies. It is sufficient to modify the value of the existing variable.

We run stress container on `worker-2` node, to simulate an exceed of the new SLA requirements. Fig. 13 shows the results generated in the presented scenario. As depicted, the CPU utilization of `worker-2` increases to maximum. This situation occurs due to the additional `stress` load run at the beginning of the experiment. AMoCNA notices the SLA violation (at `AMoCNA trigger point` time), and through declared management policies, attempts to migrate the load (node `worker-7`, created in the previous experiment carries no load and therefore, Kubernetes algorithms schedule load migration to that node). The management policy is fired at 09:51:47, just after AMoCNA notices the rapid increase of CPU utilization that exceeds SLA limits. Kubernetes scheduling mechanisms trigger the `Front-end` pod migration to `worker-7` which results in the increase of CPU utilization on that node at 09:52. Worth noticing is that despite the improvements, CPU utilization remains high. That happens due to the nature of `stress` container, which is a standalone Docker container not running in an orchestrated Kubernetes cluster.

Nevertheless, the adjustments prove that the proposed concepts can cope with SLA changes during runtime. K8s does not support this feature.

Discussion of Results. The proposed concept of using a rule engine in the context of CNApps offers a flexible mechanism of enforcing autonomic management. The rule engine introduces initial processing delays (although the presented scenario shows that these delays are not significant) – they result from matching the declared management policies against factual metrics (observations). Notwithstanding, the overall quality of the proposed concept remains satisfactory.

8 Conclusion

Autonomic management is not a new topic, but it remains a hot trend in the context of Cloud-native applications. This paper evaluates the proposed rule-based framework for the management of Cloud-native applications. AMoCNA helps

to meet the SLA agreement. Particular policies (rules) need to be declared to fulfill this capability. The recommended by us solution is flexible enough to declare meaningful policies for Cloud-native environments.

This research presents a general methodology of experimental evaluation of Cloud-native environments. We propose two categories of experiments to evaluate the rule-based approach. The first category evaluates the impacts of dynamic adjustment of resources. The second category evaluates the influence during the autonomic management process.

Meanwhile, conducted experiments proved the importance of observability. Knowledge gained in this process is a good source of information about the past and current state of the Cloud-native applications. Utilizing such knowledge while declaring management policies is an effective means of achieving autonomic management. The presented study highlights the usefulness and added value of the proposed MRE-K loop concept. Successfully validated is the concept of a rule engine treated as a means of enforcing management policies. The same experiment also reveals that the policy-based approach to autonomic management of Cloud-native applications is accurate even when dynamic and frequent modifications occur. The defined policies need to cover many aspects of the execution environment to strengthen the consequences of dynamic adjustments. We also considered the potential of the declarative management policy approach to autonomic management of Cloud-native applications by comparing its benefits to those offered by the management options available in Kubernetes. This discussion basis on experiences gained while operating the AMoCNA framework.

All experiments and the discussion proved that AMoCNA features significantly enhance the orchestrator's capabilities. Particularly AMoCNA helps meet the agreed SLA regardless of the level of available resources. Our approach represents a significant extension of the management characteristics available in current implementations of orchestrators. This area requires further research. Requirements of policy-driven infrastructure (stated in section 3) provide a solid foundation for further research. However, some of them are worth additional in-depth investigation. Highly desirable are improvements in the area of management through declarative policies. The undertaken efforts could direct towards Group-based policy [53] and an attempt to map its concepts to Kubernetes abstractions. Such an approach to management many top vendors already established (e.g., Cisco Application Centric Infrastructure (ACI)).

REFERENCES

[1] Cloud native computing foundation, Jan. 2020. [Online]. Available: https://www.cncf.io

[2] A. Currie, *Tttpe Cloud Native Attitude*, Amsterdam, Netherlands: Container Solutions Publishing, 2017.

[3] N. Carter, *Auditing the ISO 19011 Way*, B S I Standards, 2003. [Online]. Available: https://books.google.pl/books?id=Gal_HUpu3IEC

[4] M. Sloman, "Policy driven management for distributed systems," *J. Netw. Syst. Manage.*, vol. 2, no. 4, pp. 333–360, Dec. 1994.

[5] J. Strassner, *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in NetWorking)*. San Francisco, CA, USA: Morgan Kaufmann, 2003.

[6] V. Subramanian, R. Seker, S. Ramaswamy, and R. B. Lenin, "PCIEF: A policy conflict identification and evaluation framework," *Int. J. Informat. Comput. Secur.*, vol. 5, no. 1, pp. 48–67, Dec. 2012.

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[8] P. Horn, "Autonomic computing: IBM's perspective on the state of information technology," IBM, Armonk, NY, USA, Tech. Rep., 2001.

[9] S. Hariri *et al.*, "The autonomic computing paradigm," *Cluster Comput.*, vol. 9, no. 1, pp. 5–17, Jan 2006.

[10] R. Murch, *Autonomic Computing*. Indianapolis, IN, USA: IBM Press, 2004.

[11] J. Adamczyk, R. Chojnacki, M. Jarząb, and K. Zieliński, "Rule engine based lightweight framework for adaptive and autonomic computing," in *Proc. Int. Conf. Comput. Sci.*, 2008, pp. 355–364.

[12] Apply intelligence across all your data with SQL Server 2019, 2020. [Online]. Available: https://www.microsoft.com/en-us/sql-server

[13] Cisco identity services engine, 2020. [Online]. Available: https://www.cisco.com/c/en/us/products/security/identity-services-engine/index.html

[14] Broadband system management, 2020. [Online]. Available: https://www.motorolasolutions.com/en_us/products/lte-broadband-systems/operations-support-systems/system-management.html

[15] J. Kosińska and K. Zieliński, "Autonomic management framework for cloud-native applications," *J. Grid Comput.*, vol. 18, no. 4, pp. 779–796, 2020.

[16] J. Kosińska, "Autonomic management framework for cloud-native applications," Ph.D. dissertation, AGH-Univ. Sci. Technol., Kraków, Poland, 2019.

[17] Production-grade container orchestration, 2020. [Online]. Available: https://kubernetes.io

[18] J. Strassner *et al.*, "The design of a new context-aware policy model for autonomic networking," in *Proc. Int. Conf. Autonomic Comput.*, 2008, pp. 119–128.

[19] K. Webb, *Building Cisco Multilayer Switched Networks*. Indianapolis, In, USA: Cisco Press, 2000.

[20] A. Shrivastwa, S. Sarat, K. Jackson, C. Bunch, E. Sigler, and T. Campbell, *OpenStack: Building a Cloud Environment*, Birmingham, U.K.: Packt Publishing, 2016.

[21] IETF/DMTF, "Policy framework architecture," IBM, Armonk, NY, USA, Tech. Rep., 1999.

[22] S. De and A. K. Pal, "A policy-based security framework for storage and computation on enterprise data in the cloud," in *Proc. 47th Hawaii Int. Conf. Syst. Sci.*, 2014, pp. 4986–4997.

[23] V. Varadharajan, K. Karmakar, U. Tupakula, and M. Hitchens, "A policy-based security architecture for software-defined networks," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 4, pp. 897–912, Apr. 2019.

[24] M. Ayache, A. Khoumsi, and M. Erradi, "Managing security policies within cloud environments using aspect-oriented state machines," in *Proc. Int. Conf. Adv. Commun. Technol. Netw.*, 2019, pp. 1–10.

[25] J. O. Kephart and W. E. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *Proc. 5th IEEE Int. Workshop Policies Distrib. Syst. Netw.*, 2004, pp. 3–12.

[26] P. Herrero, J. L. Bosque, M. Salvadores, and M. S. Pérez, "A rule based resources management for collaborative grid environments," *Int. J. Internet Protoc. Technol.*, vol. 3, no. 1, pp. 35–45, 2008.

[27] F. Rosenberg and S. Dustdar, "Design and implementation of a service-oriented business rules broker," in *Proc. 7th IEEE Int. Conf. E-Commerce Technol. Workshops*, 2005, pp. 55–63.

[28] S. Gusmeroli, S. Piccione, and D. Rotondi, "A capability-based security approach to manage access control in the Internet of Things," *Math. Comput. Model.*, vol. 58, pp. 1189–1205, Sep. 2013.

[29] Y. Sun, T. Wu, X. Li, and M. Guizani, "A rule verification system for smart buildings," *IEEE Trans. Emerg. Top. Comput.*, vol. 5, no. 3, pp. 367–379, Third Quarter 2017.

[30] Y. Chen and B. Bordbar, "DRESS: A rule engine on spark for event stream processing," in *Proc. IEEE/ACM 3rd Int. Conf. Big Data Comput. Appl. Technol.*, 2016, pp. 46–51.

[31] G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, and T. M. Bohnert, "Self-managing cloud-native applications: Design, implementation, and experience," *Future Gener. Comp. Syst.*, vol. 72, pp. 165–179, 2017.

[32] Y. Niu, F. Liu, and Z. Li, "Load balancing across microservices," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 198–206.

[33] P. Jin *et al.*, "PostMan: Rapidly mitigating bursty traffic by offloading packet processing," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 849–862. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/jin

[34] X. Yi, F. Liu, D. Niu, H. Jin, and J. C. S. Lui, "Cocoa: Dynamic container-based group buying strategies for cloud computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 2, no. 2, pp. 1–31, Feb. 2017.
[35] N. M. Faseeh Qureshi *et al.*, "Dynamic container-based resource management framework of spark ecosystem," in *Proc. 21st Int. Conf. Adv. Commun. Technol.*, 2019, pp. 522–526.
[36] R. Han *et al.*, "Workload-adaptive configuration tuning for hierarchical cloud schedulers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2879–2895, Dec. 2019.
[37] X. Wei, C. Zhou, Y. Sheng, Y. Wu, L. Li, and S. Gao, "RLConfig: Run-time configuration of cluster schedulers via deep reinforcement learning," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Social Comput. NetWork.*, 2021, pp. 92–99.
[38] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, and S. Dustdar, "Microservices: Migration of a mission critical system," *IEEE Trans. Serv. Comput.*, vol. 14, no. 5, pp. 1464–1477, Sep./Oct. 2021.
[39] D. Agrawal, S. Calo, K.-W. Lee, J. Lobo, and D. Verma, *Policy Technologies for Self-Managing Systems*, 1st ed. Indianapolis, IN, USA: IBM Press, 2008.
[40] J. Joyce, G. Lomow, K. Slind, and B. Unger, "Monitoring distributed systems," *ACM Trans. Comput. Syst.*, vol. 5, pp. 121–150, Mar. 1987.
[41] L. Kufel, "Tools for distributed systems monitoring," *Found. Comput. Decis. Sci.*, vol. 41, no. 4, pp. 237–260, Dec. 2016.
[42] B. Brazil, *Prometheus - Up and Running: InfraStructure and Application Performance Monitoring*. Newton, MA, USA: O'Reilly Media, 2018.
[43] F. Liu, J. Guo, X. Huang, and J. C. S. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, p. 506–519, Feb. 2017.
[44] N. Kratzke and R. Peinl, "Clouns - A cloud-native application reference model for enterprise architects," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1709.04883
[45] J. Kosinski, R. Szymacha, T. Szydlo, K. Zielinski, J. Kosinska, and M. Jarzab, "Adaptive SOA solution stack," *IEEE Trans. Serv. Comput.*, vol. 5, no. 2, pp. 149–163, Second Quarter 2012.
[46] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM Syst. J.*, vol. 42, no. 1, pp. 5–18, Jan. 2003.
[47] Eclipse modeling framework (EMF), 2020. [Online]. Available: https://www.eclipse.org/modeling/emf/
[48] Weaveworks: Automate enterprise kubernetes the GitOps way, 2020. [Online]. Available: https://www.weave.works
[49] Container solutions homepage, 2020. [Online]. Available: https://container-solutions.com
[50] Sock Shop - A microservices demo application, 2020. [Online]. Available: https://microservices-demo.github.io
[51] Locust homepage, 2020. [Online]. Available: https://locust.io/
[52] A Docker container for stress, a tool for generating workload, 2020. [Online]. Available: https://hub.docker.com/r/progrium/stress
[53] What is group-based policy, 2020. [Online]. Available: https://gbp.readthedocs.io/en/latest/usage.html

Joanna Kosińska (Member, IEEE) received the PhD degree in information and communication technology discipline. She is currently an assistant professor with the Institute of Computer Science, AGH University of Science and Technology, Krakow, Poland. Her research interests include distributed computing, specifically cloud-native computing and resource management.

Krzysztof Zieliński (Member, IEEE) is currently a full professor with the Institute of Computer Science, AGH-UST. His interests include networking, service-oriented distributed systems engineering and cloud computing. He is the author of more than 200 papers in this area. He has recently published papers about autonomic systems, NFV and cloud-native applications. He is a member of ACM.