# Agent-based Simulation and Analysis of Infrastructure-as-Code Process to Build and Manage Cloud Environment

Andrzej Mycek and Daniel Grzonka
Department of Computer Science
Cracow University of Technology
Cracow, Poland

Jacek Tchórzewski
Institute of Computer Science
AGH University of Science and Technology
Cracow, Poland

## KEYWORDS

Infrastructure as Code; Cloud Computing; Terraform; Kubernetes; Agent-based Model; HASH Simulator

## ABSTRACT

Widespread cloud systems present new challenges time and time again. An essential element of such environments is their management. The Infrastructure as Code model has been gaining popularity for some time. In work presented here, we have proposed an agent-based approach to process execution within the Infrastructure as Code approach and have performed several numerical experiments. The work also includes an original formal agent model of the system. The results obtained allow us to develop trade-offs regarding computational demand and utilization.

## INTRODUCTION

The infrastructure creation, management, and subsequent maintenance approach has changed significantly. In the case of vast and complex environments implementing even the smallest change often becomes very problematic due to the sheer size of the infrastructure and the human factor. The more changes must be made manually, the greater the risk of making a mistake. The solution to this problem is approaching *Infrastructure as Code.*

Infrastructure as Code (IaC) makes it possible to manage the entire infrastructure (e.g., virtual machines, load balancers, virtual networks, and other services) by writing source code. For a modern company - building IT infrastructure in the cloud is the default method of managing all resources in the cloud. Cloud computing offers many advantages such as scalability, stability, security, and cost efficiency. Thanks to the scalability of the cloud, we do not have to invest in additional physical servers - we can flexibly increase resources. Cloud computing also gives us stability because cloud components are monitored 24/7/365, and additional mechanisms of the service provider ensure the high availability of services.

Security is also a huge advantage. Responsibility in the public cloud for security rests with the provider and ourselves. For example, in the case of the Infrastructure as a Service (IaaS) model, the provider offers us computing and network resources and is responsible for their basic security. The client's task is to secure the operating system, applications, or data. In the case of the Platform as a Service (PaaS), the provider deals with the protection of basic computing services as in the case of IaaS and is additionally responsible for the runtime environment, operating system, and middleware. The customer only needs to take care of access and data. The last type is Software as a Service (SaaS). In this model, the customer only needs to protect their data and other users.

Infrastructure as Code began to gain immense popularity quickly with the introduction of the so-called *agile software development.* In simple terms, agile software development is a method based on iterative and incremental programming for which the highest value is cooperation and flexibility [12].

Infrastructure as Code makes it easier for Development and Operations (DevOps) teams to perform complex and complicated tasks such as configuration or infrastructure maintenance using code instead of manual processes. The great advantage of Infrastructure as Code is the speed of implementation and reduced risk of making a mistake. With the Infrastructure as Code approach, the written source code should be stored in a code repository and tested for proper operation, stability, and security, just as in traditional software development. The above operations use tools in Continuous Integration (CI) and Continuous Delivery (CD) processes, such as Jenkins, Ansible, Chef, Puppet, or Terraform. Terraform is the most popular tool showcasing the *Infrastructure as Code* idea. This tool is a product of HashiCorp [16].

Terraform allows the user to create infrastructure and resources in cloud computing using HCL (HashiCorp Configuration Language). The HCL language is declarative and high-level; each code block defines a resource. The primary assumption of this language is syntax readability and the ability to interact with other tools. Declarative language (compared to imperative) describes what you want to get, not how you want to get something.

The rest of the paper is structured as follows. Section *Related Work* discusses the related work. We briefly described implementing a cloud environment based on the

Communications of the ECMS, Volume 37, Issue 1,
Proceedings, ©ECMS Enrico Vicario, Romeo Bandinelli,
Virginia Fani, Michele Mastroianni (Editors) 2023
ISBN: 978-3-937436-80-7/978-3-937436-79-1 (CD) ISSN 2522-2414

IaC model in Section *Principle of Operation.* The next section (IaC vs Traditional Infrastructure) presents the differences between the traditional approach and the IaC model. Within Section *Agent-Based Simulator*, we presented the original formal agent model of the simulator under test. Evaluation of the simulation is presented in the experiments section. Finally, Section *Summary* concludes conducted experiments and identifies potential opportunities for further work.

## RELATED WORK

Accordingly, to [7], multi-cloud models account for about 75% of the cloud market. Due to very high computing capability, multi-cloud systems are called "Sky Computing". Those models need special infrastructure tools like cloud orchestrators to handle information flow. In [7], authors tested six cloud orchestrators currently most referenced in the literature: Cloudify, Heat, CloudFormation, Terraform, Cloud Assembly, and the TOSCA standard. During the literature review, they found out that Terraform and Cloudify offer similar Sky Computing scenarios, but the practical experiment revealed that Terraform is outperforming Cloudify.

In [13], authors claimed that Infrastructure as code (IaC) is widespread in complex cloud systems. The idea of IaC is to deliver fast and reliable services to users. Big companies like Facebook, Google, and GitHub currently utilize this idea. In the article, the authors tried identifying potential flaws (like security breaches) and research areas related to the IaC concept. They analyzed 32 articles and concluded that this topic is well-studied; however, conducting more research on security flaws is necessary.

In [14], authors described their own IaC system. They proposed an architecture and implemented a cloud benchmarking Web service. The presented model was based on the assumptions of reusable and representative benchmarks. Authors also claimed that classical benchmarking cloud services are cumbersome and error-prone, whereas the presented IaC concept is reproducible well-defined and easy to test.

Another solution utilizing the IaC concept was presented in [6]. The authors designed a search-based problem-solving agent named YUMA. The algorithm's core is a search tree holding the state space and transition model. The article presents the search tree-building algorithm and two additional algorithms determining the minimal composition plan. Results presented in the paper indicate that YUMA fulfills requirements and may be helpful for cloud architects.

The Virtual Machine (VM) evaluation method is necessary to choose the optimal VM for a particular task. Usually, it is done via benchmarking or a black-box search. In [11], authors presented their system called Framework with Infrastructure-as-Code (IaC) support For VM Evaluation (FIFE). This framework is an easily-configurable abstraction layer separating the searcher, selector, deployer, and interpreter. The whole process can be automated with the usage of JSON files. Results presented in the paper prove that the framework does not influence search efficiency when VMs are from different cloud providers and significantly improves parallel search time efficiency.

In [10], authors investigated open-source cloud technology called OpenStack. They analyzed the architecture, requirements, setup process, and related problems. The authors also analyzed the resource utility (with a full load and without). The conclusion was that OpenStack is mainly supported on Ubuntu and demands RAM. However, it is a good platform for educational and testing purposes and has an extensive computer infrastructure. On a single machine, the loss of performance is very significant.

A novel Multi-Agent System for Cloud Monitoring (MAS-CM) model was described in [9]. The presented solution is focused on performance and security during gathering task results and scheduling in cloud systems. The authors proved that their model could prevent unauthorized task injection and modification. It is also optimizing the scheduling process and maximizing resource utilization. The effectiveness of MAS-CM was investigated using an evolutionary driven implementation of the Independent Batch Scheduler and FastFlow framework. Results indicated that MAS-CM is increasing the performance of the system.

## PRINCIPLE OF OPERATION

Creating resources with Terraform consists of three main steps: writing the resources we want to create in HCL, generating a plan and running the tool (see: fig. 1).

### I. Init and resource creation

The first stage involves gathering requirements, planning, and writing the source code. At this stage, the supplier and services we care about are defined. The provider's definition and the resources we want to create are placed in a file with the *.tf* extension (see: fig. 2).

### II. Plan

After writing the code, the next step is to generate the plan. Issuing *terraform plan* command Terraform looks in local directories for configuration files. The tool creates a plan and checks the current state of the remote objects that will be introduced in the infrastructure.

### III. Apply

The final stage of the workflow is *terraform apply.* Without passing a saved plan file, Terraform asks for approval of this plan and then takes the appropriate action. When a saved plan file is handed over, Terraform will perform the actions without asking for confirmation.

In Terraform we deal with something called Providers. Provider in Terraform is nothing more than a special plugin allowing API interaction. This includes cloud computing providers such as Amazon Web Services, Microsoft Azure, Google Cloud Platform, or Or-

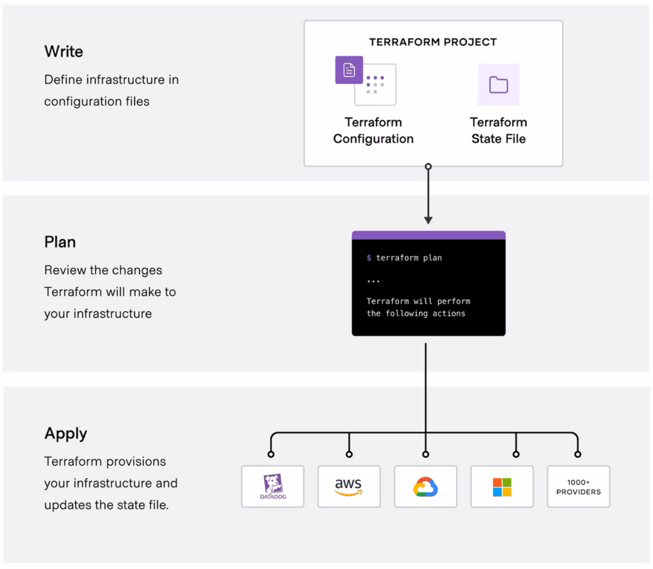
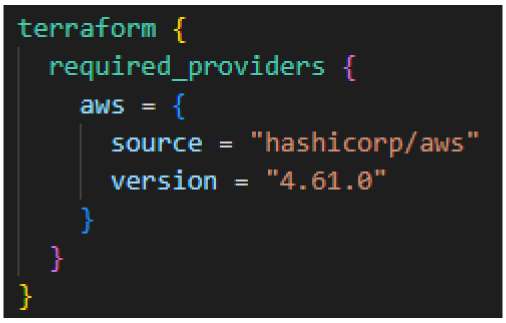

Write
Define infrastructure in configuration files
TERRAFORM PROJECT
Terraform Configuration
Terraform State File
Plan
Review the changes Terraform will make to your infrastructure
$ terraform plan
...
Terraform will perform the following actions
Apply
Terraform provisions your infrastructure and updates the state file.
DATADOG
aws
1000+ PROVIDERS

Fig. 1: Terraform Workflow



```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.61.0"
    }
  }
}
```

Fig. 2: Use of provider

acle Cloud Infrastructure, and additionally providers of various types of software available as services. The abovementioned providers are just the four most popular providers. The list is much more extensive and amounts to almost 3,000. The complete list of available providers can be found on the official Terraform website.

Providers allow us to use different resources and data sources that Terraform can manage. Without a provider, infrastructure management using Terraform is impossible; a given provider implements each type of resource.

In the Infrastructure as Code idea, one must include another vital tool - Ansible. This tool often needs clarification with Terraform, and there is a fundamental difference between them. Ansible is a tool written by Michael DeHaan, currently developed by Red Hat Inc. Ansible combines declarative configuration, as in the case of Terraform, and additional procedural configuration. The procedural configuration specifies the exact state of our infrastructure.

A trendy tool that works like Terraform is Cloud Formation. It allows to creates and manages infrastructure; unfortunately, it is not as flexible as Terraform. Terraform is entirely independent of the provider, unlike Cloud Formation. The latter works only in the Amazon cloud.

## IaC VS TRADITIONAL INFRASTRUCTURE

There are often debates about Infrastructure as Code or Traditional Infrastructure. Which of these approaches seems more appropriate in today's IT world? As with any novelty, and not only in the IT industry but in any other industry, we have many skeptics. Creating, managing, and storing infrastructure in the form of code was something unacceptable to many. Writing source code has been chiefly a task for software engineering teams. However, IT Administrators soon discovered that this approach has no weak points - the only downside is that they have to master the new technology. However, everyone more or less related to the IT industry is used to constantly expanding their skills.

In addition, it must be taken into account that there has long been friction between the teams of developers and administrators. Developers were not very interested in where the application would be deployed and what infrastructure resources a company or an external client had. These conditions contributed to the creation of the DevOps trend. Administrators often needed to learn how to build or operate specific developer tools. Therefore, a new role has arisen in the IT industry - DevOps. DevOps engineers partly deal with tasks that fell on the heads of programmers and some of the tasks intended for IT Administrators. The combination of these two distinct roles eliminated a seemingly impossible conflict.

The power of Infrastructure as Code will be noticed by anyone who has enjoyed creating and maintaining infrastructure in the classic model. In the traditional approach, all the work is done manually or with the help of automation scripts, but with scripts, we can, at most, configure something. We cannot create infrastructure from scratch - it will require manual work. All three most important activities - provisioning of servers, configurations of servers, and deployment of software - can be automated using Infrastructure as Code tools. In the event of a failure or migration to another cloud computing provider, thanks to IaC, we can easily recreate or migrate our infrastructure thanks to the phenomenon of repeatability. The traditional model, of course, still has its adherents. Most often, these people need to be better acquainted with modern technologies or have yet to desire to learn them thoroughly. The traditional model is, of course, still valid. It will work better in small and uncomplicated environments where the IT staff is not up to date with technological innovations in the IT industry.

In summary - choosing the right concept depends mainly on the organization's requirements, needs, and goals. Nevertheless, in most cases, without a doubt, the only right choice will be the Infrastructure as Code concept.

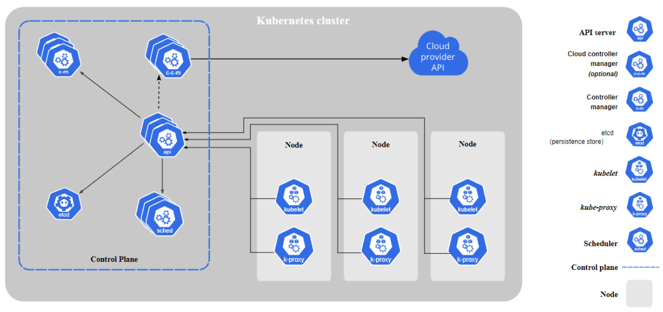


Fig. 3: Components of Kubernetes

## AGENT-BASED SIMULATOR

As part of this work, we have made several experiments based on a dedicated simulator implemented using the HASH platform [1], which provides an end-to-end solution for safely automating decision-making. Terraform-based Cloud Infrastructure Simulator ([4]) allows demonstrating how to generate a representation of cloud infrastructure using the Infrastructure as Code model. The simulator's code is available under an MIT license so that anyone can tailor the project to their needs.

The simulator is based on the idea of a platform for managing, automating, and scaling containerized applications - Kubernetes. As for the Kubernetes architecture, a Kubernetes cluster consists of a master node called a control plane and workers nodes. The master node distributes work to worker nodes, monitors the status of the application and cluster itself, issues an *API (Application Programming Interface)*, and ensures integration with the cloud provider. The task of worker nodes is to perform the work assigned by the control plane. A node is a worker machine in Kubernetes and is the environment in which pods run (see: fig. 3). So-called pods handle requests forwarded by users to the cloud. Pods (which take their name from a pea pod) are the most minor, short-lived computing units that can share resources such as memory, storage, or network. Notably, the contents of a Pod container are always co-located and co-scheduled and run in a shared context [2].

The HASH platform itself is based on an agent-based paradigm. In the solution used, the agents are made up of State, Behaviours, and Context. Compared to a typical agent system (cf. [17], [18]), behavior can be identified with actions, state with the agent's internal state, while context can be identified with the environment state. Based on the agent model presented in [15], we propose the following formal multi-agent system model embedded in the idea of a Terraform-based Cloud Infrastructure Simulator:

$$MAS = \{AG, ID, TP, K, ES, ACT, ST, GL\}, \tag{1}$$

where:
$AG$ — a set of agents belonging to a multi-agent system;
$ID$ — a set of unique identifiers for agents;
$TP$ — a collection of all agent types;
$K$ — a set of all possible agent's states;
$ES$ — a set of all possible contexts;
$ACT$ — a set of behaviors that agents can perform within the context;
$ST$ — a set of strategies implemented by agents;
$GL$ — a set of agent objectives.

An agent ($ag$) is defined as follows:

$$AG \ni ag = \{id, tp, st, k, gl, \alpha, \beta, \gamma\}, \tag{2}$$

where:
$id \in ID$ — a unique system-wide identifier for the agent;
$tp \in TP$ — agent's type;
$st \in ST$ — agent's strategy;
$k \in K$ — current state of the agent;
$gl \in GL$ — agent's current objective;
$\alpha \in ACT$ — a behavior function that, based on the current state and context of the agent, is updating the internal agent's state:

$$\alpha : K \times ES \to \mathcal{M}(\mathcal{K}), \tag{3}$$

where $\mathcal{M}$ denotes the space of probabilistic measures over the set;
$\beta$ — strategy selection function:

$$\beta : TP \times K \to ST; \tag{4}$$

$\gamma$ — a decision-making function which, based on the strategy and objective, selects actions:

$$\gamma : ST \times GL \to ACT. \tag{5}$$

We can equate behaviors with actions in a typical agent model. Internal states and agent contexts define them. Only an agent can change its internal state. Another agent can induce a change in its state, but the decision is up to the agent whose state it is. In the case of contexts, as a rule, they are fixed; hence this model does not contain a function that updates the context.

For more information on the modeling approach, see [8].

The simulator also has several parameters and data sources that control its operation. Different types of instances from the Amazon Web Services cloud were used for the experiment, and requests are generated using accurate data - the distribution of requests shows the percentage of requests during the day broken down by each hour. The simulation has a rich database of controlling parameters, including:
- number of daily requests; triangular item distribution that determines the duration of each running request;
- the number of CPU cores and gigabytes of memory allocated to the pod;
- the number of gigabytes of memory and CPU cores in compute node in the cluster;
- the auto-scaler automatically removes and adds nodes (e.g., if the maximum utilization of the cluster

| Parameter | Value |
|---|---|
| vCPU | 48 |
| Memory (GiB) | 96 |
| Instance Storage (GB) | EBS-Only |
| Network Bandwidth (Gbps) | 12 |
| EBS Bandwidth (Mbps) | 9,500 |

TABLE 1: C5.12xlarge instance specification

is exceeded, an additional node will be added to it, the same is true when the utilization falls below the minimum value - then one node is removed from the cluster);
• the minimal number of nodes required to be in the cluster;
• the initial number of nodes in the cluster;
• the time delay before a request to add a node to our cluster is fulfilled.

## EXPERIMENTS

A Kubernetes cluster was built for research. We used Amazon EKS (Elastic Kubernetes Service), which allows you to use Kubernetes and thus enables deployment, management, and scaling of container applications in the AWS Cloud. A typical Kubernetes cluster consists of two parts, namely the control plane, and nodes. Both parts are critical, and it is impossible to say which is more important. The good news is that in the case of the EKS service (similarly to Microsoft in the case of the Azure Kubernetes Service), it is the cloud computing provider, i.e., Amazon, who takes responsibility for the control layer. The customer only deals with the issue of nodes.

Our cluster was built with 24 nodes - *c5.12xlarge* instances (see: tab. 1). Amazon EC2 C5 instances have much computing power, are great for distributed analysis or scientific modeling, and are cost-effective due to their low price-to-computing power ratio. The c5.12xlarge instance has 48 vCPU and 96 GiB of RAM. It has an Intel Xeon Platinum 8275L processor clocked at 3.0 GHz.

An application for simulators and calculations was installed on a working Kubernetes cluster. Because the cluster is located in the Amazon cloud, we took advantage of the possibility of scaling nodes and the mechanism that automatically changes the number of pods. The simulator based on the HASH platform allows us to forecast the cloud resources we will need to keep our application available and provide services at a high level.

A simulation was conducted with the following daily requests: 1,000, 2,000, and 10,000. Considering our environment and the application itself, according to the simulator, at 10,000 received requests per day, seven nodes will be enough (see: fig. 6). This balance is also evident in the following graph showing the cluster's memory usage (see: fig. 7), where for the received 10,000 requests, we have seven nodes with 672 GiB of memory, and in the case of the "Cluster CPU Usage" plot (fig. 8) showing the total amount of RAM in the

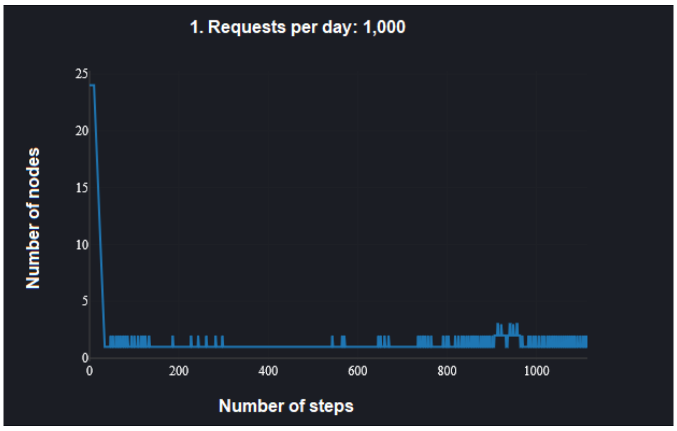


Fig. 4: Number of nodes with 1,000 requests received

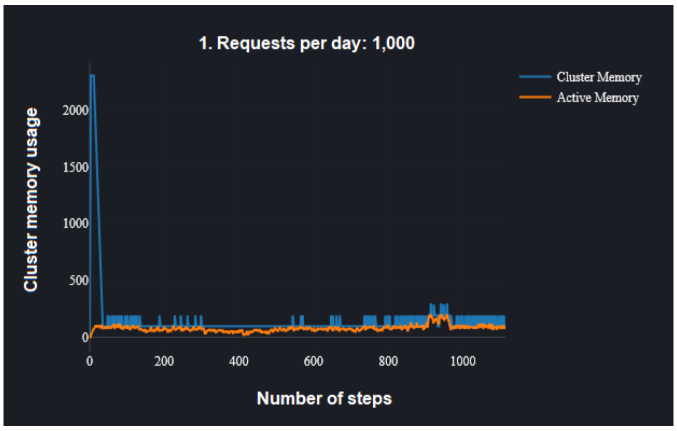


Fig. 5: Cluster memory usage for 1,000 requests received

cluster and the current usage in a time step defined by us.

Using the words "Kubernetes scaling" we cannot forget about changing the number of pods of a given Deployment. With the increase in traffic, it is necessary to scale the application to handle a more significant number of users, which can be seen in the example of our experiment.

Another part of the experiment was to increase the number of received requests and, more precisely, to double our initial values. Our inputs were three values: 2,000, 4,000, and 20,000. As expected, with more received requests, the application needed more computing power. With 20,000 requests received, the simulator calculated that we need fourteen nodes representing 1,344 gigabytes of RAM and 672 vCPUs (see: fig. 6). So, as can be seen, the number of resources needed has slightly more than doubled. The same is also illustrated by the number of pods (fig. 9). For 20,000 requests received daily, the cluster's RAM usage was around 640 GB at peak (see: fig. 7).

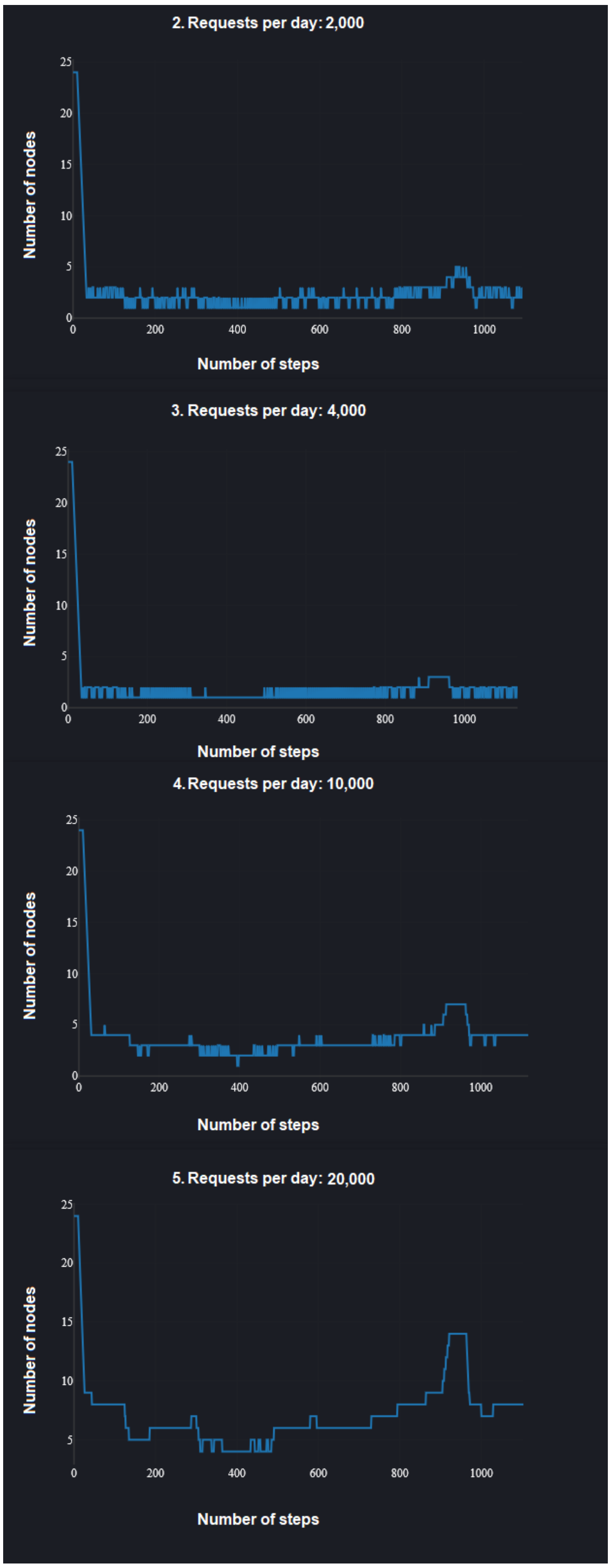


Fig. 6: Number of nodes with 2,000, 4,000, 10,000, and 20,000 requests received

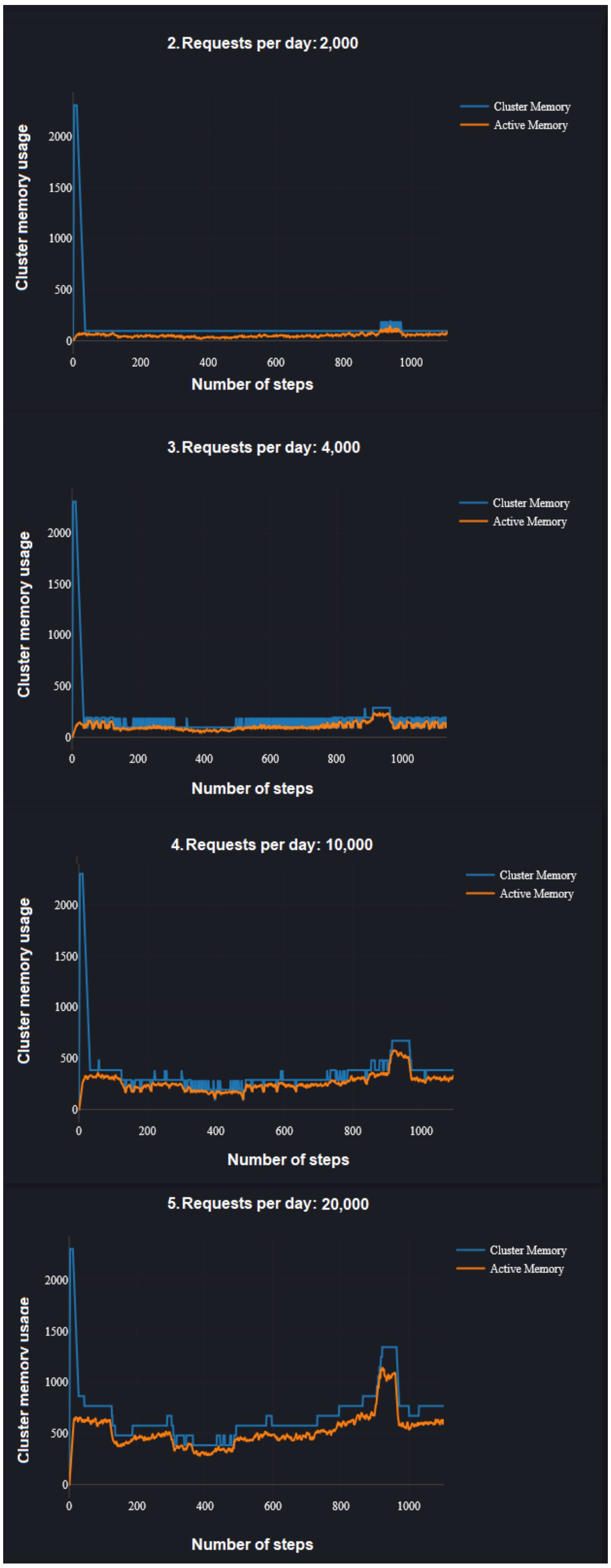


Fig. 7: Cluster memory usage for 2,000, 4,000, 10,000, and 20,000 requests received

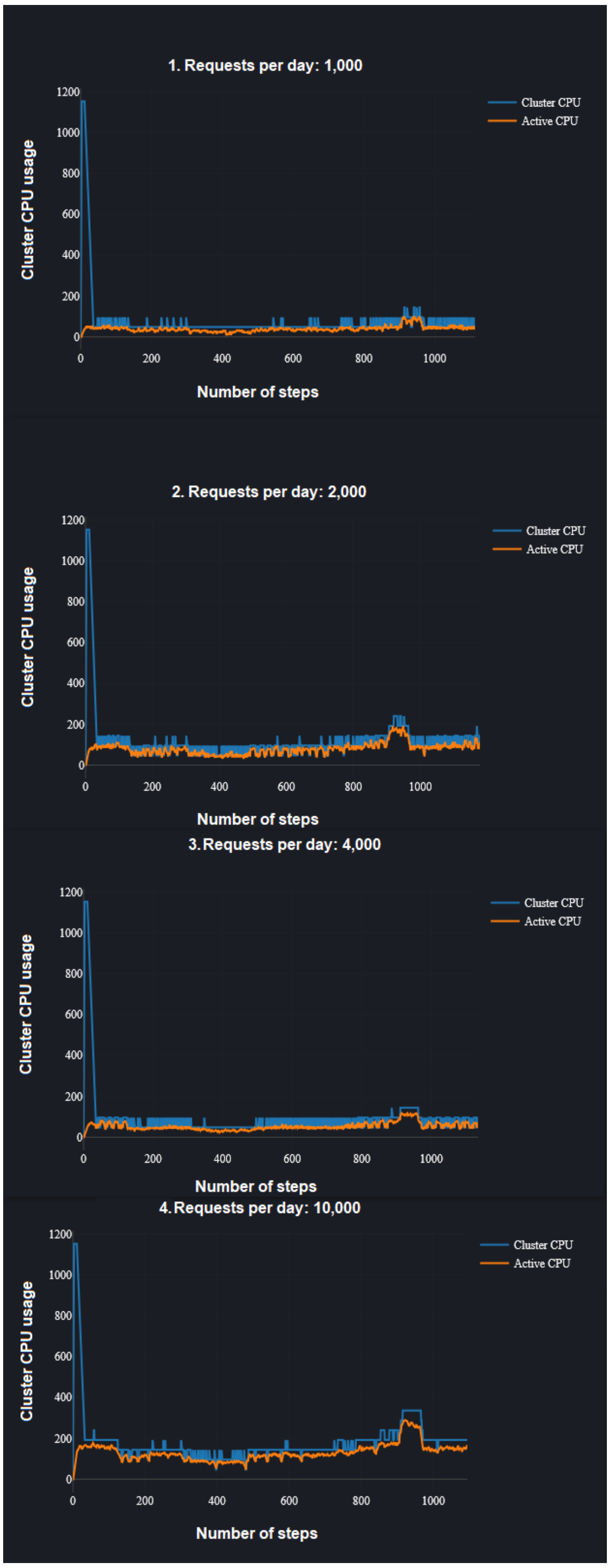


Fig. 8: Cluster CPU usage for 1,000, 2,000, 4,000, and 10,000 requests received

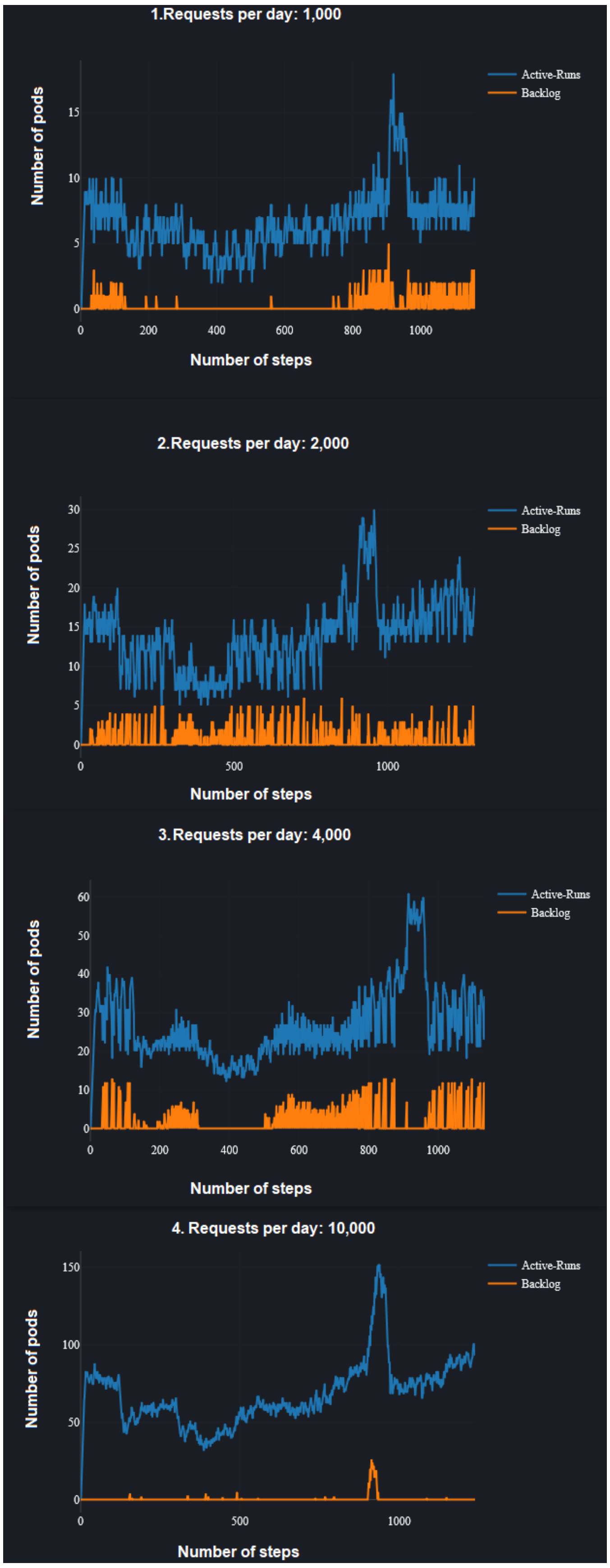


Fig. 9: Number of pods for 1,000, 2,000, 4,000, and 10,000 requests received

## SUMMARY AND FUTURE WORK

Thanks to the simulator used in our experiment, we can estimate what resources we will need depending on the load. Having an application and infrastructure in the cloud is essential, so are costs and their ongoing monitoring - we can also estimate and forecast potential costs depending on several factors and finding a trade-off between performance and cost. Our research balances the need for and delivery of computing resources.

In the future, this simulator can also be expanded with the resources of other cloud computing providers (e.g., Azure, Google Cloud Platform, Alibaba Cloud, or Oracle Cloud). The simulator can also be helpful for a private cloud. It can be easily adapted to our needs. We also plan to implement various optimization mechanisms for modeling problems of large-scale computing environments based on the IaC paradigm.

## REFERENCES

[1] HASH platform - https://hash.ai/platform/.
[2] Kubernetes Documentation - https://kubernetes.io/docs/home/.
[3] Microsoft Threat Modeling - https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling/.
[4] Terraform-based Cloud Infrastructure Simulator - https://hash.ai/@hash/terraform-simulation/.
[5] Threat Modeling - https://github.com/trustoncloud/threatmodel-for-aws-s3/.
[6] F. Baer and M. Leyer. Yuma–an ai planning agent for composing it services from infrastructure-as-code specifications. 2023.
[7] L. R. de Carvalho and A. Patricia Favacho de Araujo. Performance comparison of terraform and cloudify as multi-cloud orchestrators. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 380–389, 2020.
[8] D. Grzonka. Inteligentne systemy monitoringu procesów harmonogramowania w rozproszonych środowiskach dużej skali w ujęciu wieloagentowym. *Praca doktorska*, 2018.
[9] D. Grzonka, A. Jakóbik, J. Kołodziej, and S. Pllana. Using a multi-agent system and artificial intelligence for monitoring and improving the cloud performance and security. *Future Generation Computer Systems*, 86:1106–1117, 2018.
[10] D. Grzonka, M. Szczygiel, A. Bernasiewicz, A. Wilczynski, and M. Liszka. Short analysis of implementation and resource utilization for the openstack cloud computing platform. In V. M. Mladenov, G. Spasov, P. Georgieva, and G. Petrova, editors, *29th European Conference on Modelling and Simulation, ECMS 2015, Albena (Varna), Bulgaria, May 26-29, 2015. Proceedings*, pages 608–614. European Council for Modeling and Simulation, 2015.
[11] Y. Lin, J. Briggs, and A. Barker. Fife: an infrastructure-as-code based framework for evaluating vm instances from multiple clouds. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pages 91–100, 2020.
[12] R. Martin. *Agile Software Development, Principles, Patterns, and Practices.* Pearson, 2002.
[13] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams. A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65–77, 2019.
[14] J. Scheuner, P. Leitner, J. Cito, and H. Gall. Cloud work bench – infrastructure-as-code based cloud benchmarking. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 246–253, 2014.
[15] A. Sosnicki, D. Grzonka, and L. Gaza. Agent and evolutionary-based modelling and simulation of A simplified living system. In I. A. Hameed, A. Hasan, and S. A. Alaliyat, editors, *Proceedings of the 36th ECMS International Conference on Modelling and Simulation, ECMS 2022, Ålesund, Norway, May 30 - June 3, 2022*, pages 296–302. European Council for Modeling and Simulation, 2022.
[16] R. Wang. Infrastructure as code, patterns and practices: With examples in python and terraform. New York, NY, USA, 2022. Manning Publications.
[17] G. Weiss. *Multiagent Systems.* The MIT Press, 2013.
[18] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.

## AUTHOR BIOGRAPHIES

**ANDRZEJ MYCEK** received his B.Eng. and M.Sc. in the field of Computer Science at Cracow University of Technology, Poland, in 2019 and 2022, respectively. Currently, he is a Research and Teaching Assistant at the same university. His main areas of interest include software engineering, cloud architectures, infrastructure automation, project management, and cybersecurity. If you want to contact him, please send an e-mail to andrzej.mycek@pk.edu.pl

**DANIEL GRZONKA** received his B.Eng. and M.Sc. degrees from Cracow University of Technology, Poland, in 2012 and 2013, respectively. In 2019 he received his Ph.D. from the Polish Academy of Sciences (in cooperation with Jagiellonian University). All degrees (with distinctions) are in Computer Science. He is an Assistant Professor at the Cracow University of Technology and Vice-Dean for Education at the Faculty of Computer Science and Telecommunications. He is also the laureate of the prestigious START 2019 competition of the Foundation for Polish Science for the most outstanding young scientists. The main topics of his research are multi-agent systems, grid and cloud computing, task scheduling problems, data mining, and high-performance computing. For more information, please visit: www.grzonka.eu or contact by e-mail: daniel.grzonka@pk.edu.pl

**JACEK TCHÓRZEWSKI** received his B.Sc. and M.Sc. degrees from Cracow University of Technology, Poland, in 2016 and 2017, respectively. In 2023 he received his Ph.D. from the AGH University of Science and Technology. All degrees (with distinctions) are in Computer Science. He is a Research and Teaching Assistant at the AGH University of Science and Technology. His e-mail address is: jacek.tchorzewski@onet.pl