information

Article

Optimization and Prediction Techniques for Self-Healing and Self-Learning Applications in a Trustworthy Cloud Continuum

Juncal Alonso *, Leire Orue-Echevarria, Eneko Osaba, Jesús López Lobo, Iñigo Martinez, Josu Diaz de Arcaya and Iñaki Etxaniz

TECNALIA, Basque Research and Technology Alliance (BRTA), Parque Científico y Tecnológico de Bizkaia, 48160 Derio, Spain; Leire.orue-echevarria@tecnalia.com (L.O.-E.); Eneko.osaba@tecnalia.com (E.O.); jesus.lopez@tecnalia.com (J.L.L.); inigo.martinez@tecnalia.com (I.M.); josu.diazdearcaya@tecnalia.com (J.D.d.A.); inaki.etxaniz@tecnalia.com (I.E.)

* Correspondence: juncal.alonso@tecnalia.com

Abstract: The current IT market is more and more dominated by the "cloud continuum". In the "*traditional*" cloud, computing resources are typically homogeneous in order to facilitate economies of scale. In contrast, in edge computing, computational resources are widely diverse, commonly with scarce capacities and must be managed very efficiently due to battery constraints or other limitations. A combination of resources and services at the edge (edge computing), in the core (cloud computing), and along the data path (fog computing) is needed through a trusted cloud continuum. This requires novel solutions for the creation, optimization, management, and automatic operation of such infrastructure through new approaches such as infrastructure as code (IaC). In this paper, we analyze how artificial intelligence (AI)-based techniques and tools can enhance the operation of complex applications to support the broad and multi-stage heterogeneity of the infrastructural layer in the "computing continuum" through the enhancement of IaC optimization, IaC self-learning, and IaC self-healing. To this extent, the presented work proposes a set of tools, methods, and techniques for applications' operators to seamlessly select, combine, configure, and adapt computation resources all along the data path and support the complete service lifecycle covering: (1) optimized distributed application deployment over heterogeneous computing resources; (2) monitoring of execution platforms in real time including continuous control and trust of the infrastructural services; (3) application deployment and adaptation while optimizing the execution; and (4) application self-recovery to avoid compromising situations that may lead to an unexpected failure.

Keywords: optimization; self-learning; concept drift; anomaly detection; cloud continuum; self-healing

Citation: Alonso, J.; Orue-Echevarria, L.; Osaba, E.; López Lobo, J.; Martinez, I.; Diaz de Arcaya, J.; Etxaniz, I. Optimization and Prediction Techniques for Self-Healing and Self-Learning Applications in a Trustworthy Cloud Continuum. *Information* **2021**, *12*, 308. https://doi.org/10.3390/info12080308

Academic Editor: Willy Susilo

Received: 12 July 2021
Accepted: 29 July 2021
Published: 30 July 2021

1. Introduction

Infrastructure as code, or IaC, aims to automate the provisioning, configuration, and deployment of infrastructure resources based on a machine-readable file that is developed by Ops teams. A software agent tool then processes it and executes tasks to provision, configure, and deploy the user-defined infrastructure. This automatic deployment can be improved through optimization techniques, trying to provide the best selection of resources and configurations in each case. In this regard, it is worth mentioning that the formulation and solving of optimization problems by the AI community has been enhanced in recent years with the emergence of new paradigms related to problem modeling, as large-scale optimization or multi-objective problems characteristics that are typically present in the DevOps environments.

In any case, despite several works that have been proposed in last years, IaC still suffers from several important issues, such as the efficient optimization of the microservices deployed, the dynamic analysis of the service performance, or the implementation of corrective actions at run-time. The lack of available works in the literature that give an

efficient answer to issues like that has motivated the organization of the PIACERE project. The main objective of PIACERE is to develop a solution that covers the development, deployment, and operation of IaC of applications deployed on the cloud continuum. The main target users of PIACERE are DevSecOps teams.

In this work, we describe some of the most important aspects that compose the PIACERE project, in which we propose the use of AI-based techniques to assist DevOps teams in the whole lifecycle of infrastructure management. Namely, in the task of deploying distributed applications in heterogeneous cloud environments (cloud continuum), helping them first in selecting and combining the optimal infrastructure resources available; later, in detecting specific behaviors that could cause unexpected failures through continuous monitoring of infrastructural services; and finally, in the automatic self-healing of the application during run-time to allow the final users an optimal experience regardless of possible infrastructure shortcomings or failures. While relevant work exists [1,2], they differ in (1) the resource type that they target for the optimization as they focus only either on virtual machines (VMs) (ref cloudlightning) or on VMs, DB, and Storage (ref decide), leaving aside the computing continuum, and (2) they do not present a full-fledged solution that detects anomalies in the infrastructure which can involve the triggering of a self-healing strategy.

To reach these objectives, we describe how to use innovative approaches such as evolutionary computation to solve multi-objective optimization problems, machine learning techniques applied to the analysis of dynamic data streams, or the use of IaC approaches that allows taking corrective actions at run-time. This is precisely the main contribution of this paper: the description of the novel solving approach that we have proposed for efficiently deal with cornerstone aspects of the IaC lifecycle, such as (1) the optimized choosing of the deployed microservices, (2) the efficient monitoring and the adaptation of the system to the overall performance demonstrated, and (3) the effective and dynamic autonomous corrective actions at run-time.

The rest of the paper is structured in three main sections. Section 2 is dedicated to present the related work on how AI can support the DevOps operations and, in particular, on the fields that are most relevant for the article: optimization of the IaC, self-learning through monitoring and anomaly prediction, and self-healing mechanisms for corrective actions. Section 3 describes the PIACERE approach for an optimized and self-healed IaC. The main components of the proposed solution are defined—namely, optimization, self-learning, and self-healing—and an overview of how each of them works is provided. Finally, Section 4 provides the conclusions, a general overview of the research and its future works.

2. Motivation and Related Work

Operating and managing a DevOps environment implies a high degree of complexity. Furthermore, the emergence of approaches for the more and more heterogeneous infrastructural layer, such as IoT or the cloud continuum (combination of the cloud and the edge), requires novel solutions to monitor, manage, and operate distributed application environments. DevOps teams need to effectively gather, analyze, and take decisions based on the information available from the infrastructural layer to address and resolve potential issues so that the business continuity of the application running on top of this environment is always guaranteed. On the contrary, since humans are currently not well prepared to handle the massive volumes of data and computing in daily operations, artificial intelligence is expected to become the critical tool for computing, analyzing, and transforming how teams develop, deliver, deploy, and operate applications.

DevOps and AI are interdependent as DevOps is a business-driven approach to deliver software, and AI is the technology that can be integrated into the system for enhanced functionality. With the help of AI, DevOps teams can test, deploy, release, and monitor software more efficiently. AI can also improve automation and support decision making by quickly identifying issues and mitigation actions.

In the next subsections, we explore the state of the art on the three main relevant areas of the DevOps cycle where we think AI can positively impact the DevOps process, namely optimization of the deployment configuration, self-learning through autonomous monitoring, and prediction of anomalies techniques and self-healing mechanisms. We combine the analysis of these AI-related methods with the study of the current state of the art of approaches and technologies for the automatization of infrastructure generation and configuration, particularly IaC related.

2.1. Deployment Configuration Optimization Supported by AI Methods

The formulation and subsequent solving of optimization problems by employing advanced artificial intelligence methods is a hot topic in the current scientific community, maintaining the remarkable popularity gained a couple of decades ago. Particularly, in recent years, the attention received has been intensified due to the emergence of new solving paradigms mainly focused on new ways of problem modeling and the huge impact of knowledge streams such as evolutionary computation (EC) and swarm intelligence (SI) [3–5].

On this subject, and due to the inherent social and scientific value behind the efficient resolution of optimization problems, a myriad of solving schemes have been proposed in recent years, which have shown great performance on a countless variety of real-world applications. In this regard, the most recognized techniques can be classified into three main categories: exact methods, heuristics, and metaheuristics. Among these categories, it is the last one that enjoys great popularity due to its efficiency and adaptability features.

Focusing our attention to optimization itself, a rich spectrum of valuable optimization kinds has been formulated up to now, such as robust optimization, dynamic optimization, transfer optimization, large scale global optimization and many others. Arguably, one of the most successful and applicable kinds of optimization is known as multi-objective optimization (MOO, [6]), which in fact is the one on which PIACERE focuses its attention.

Briefly explained, the principal goal of MOO is to find a group of solutions that efficiently balance several conflicting optimization objectives. It is important to mention that these objectives are defined on a single domain, which means that a single search space is explored. This last feature distinguish MOO from other optimization types such as transfer optimization [7]. MOO works with the existence of Pareto trade-off among the objectives at hand. Thus, MOO algorithms build an estimation in the form of a group of potential solutions. Consequently, there is not a sole solution for each problem but a set of solutions that meet all objectives to a certain level.

In this context, methods that can be framed in the wide category EC have particularly gained remarkable popularity in the related community for solving MOO problems [8]. This fact has led to an unprecedented amount of interesting research works presenting algorithmic procedures for the efficient tackling of MOO problems through the leverage of the self-learning feature of their immitted natural phenomena. The principal reason behind the popularity gained by this broad family of solvers is their remarkable performance, which has been proved in hundreds of problems and real-world problems. In this line, many different inspirational sources have been used for building optimization techniques, such as the behavioral patterns of bees, bats, and buffalos; the social procedures of humans or animals; or the physical procedures behind natural phenomena such as the rain or the water cycle.

With all this, the election of one specific algorithm for tackling an optimization problem is usually a hard task. Usually, this decision depends on different factors, such as the amount of optimizing objectives, the non-functional requirements, or the complexity of the objective function. Anyway, there are plenty of well-established and reputed MOO solvers in the current literature, which frequently guide the decision and the final implementation of the deployed algorithms. In this regard, some of the methods which will be carefully analyzed in the context of the PIACERE IOP platform are:

- *Non-dominated sorting genetic algorithm II* (NSGA-II, [9]): The NSGA-II is one of the most used and recognized MOO methods. Essentially, it is a generational genetic algorithm, which employs a ranking procedure for easing the estimation of the crowding distance density and the convergence of the Pareto. This also contributes to the enhancing of population diversity.
- *Strength Pareto evolutionary algorithm 2* (SPEA2, [10]): the main advantages of this reputed method are: (i) the improvement of the fitness assignment pattern (which considers the number of individuals dominated), (ii) the incorporation of a nearest neighbor density estimator for more fine guidance of the search procedure, and (iii) a novel archive truncation technique, which assures the preservation of boundary solutions.
- *Multi-objective evolutionary algorithm based on decomposition* (MOEA/D, [11]): MOEA/D is characterized by the decomposition of the MOO problem into several scalar subproblems, optimizing them in a simultaneous way. Specifically, each subproblem is tackled employing information from its nearer neighborhood problems. This last feature makes MOEA/D be a less demanding method in comparison with other similar strategies.
- *Speed-constrained multi-objective particle swarm optimization* (SMPSO, [12]): this method is based on the well-known particle swarm optimization algorithm. Thus, the principal feature of this method is the use of a velocity constraint mechanism to avoid particles flying beyond the limits of the search space. This method employs an external bounded-sized archive for storing the non-dominated solutions encountered along with the search procedure, from which the leader is also selected.
- *Multi-objective cellular genetic algorithm* (MOCell, [13]): the MOCell is a recently proposed MOO algorithm that is inspired by the reputed cellular genetic algorithm family. This method also employs an external archive for saving non-dominated solutions, and it implements a feedback mechanism in which solutions from this archive are drawn and used in the selection step of the algorithm.

Usually, multi-objective problems are conceived for optimizing two different confronting objectives. In any case, in many real-world applications, the number of objectives to optimize can be higher than two. This trend gives rise to many objective algorithms, which should deal with the existence of many confronting objectives, while also trying to define the main relationships between them. These are some interesting alternatives to consider [14]:

- *Non-dominated sorting genetic algorithm III* (NSGA-III): this successful method follows the same philosophy and structure of the above-mentioned NSGA-II. NSGA-III is principally characterized by emphasizing a non-dominated population of individuals near a group of supplied reference points.
- *Multi-objective evolutionary algorithm based on dominance and decomposition* (MOEA/DD): this advanced and successful method unifies some efficient concepts, combining dominance and decomposition-based approaches, for many-objective optimization.
- *Strength Pareto evolutionary algorithm based on reference direction* (SPEA/R): This method is an extension of the above mentioned SPEA, introducing a reference direction-based density estimator, a new fitness assignment procedure, and a new environmental selection mechanism.

In the specific context of PIACERE, the optimization problem to be modeled consists of having a service to be deployed and a catalogue of infrastructural elements, with the challenge of obtaining the best combination of infrastructural elements and resource configuration to optimally deploy the service. Despite being an incipient research field, some studies dealing with similar problems can be found in the literature.

In [15], a novel optimization system is presented capable of matching the specific functional and non-functional requirements of a certain Big Data application to the capabilities offered by an IaaS infrastructure and the Big Data platform deployed therein. To do that, the optimization problem is modeled and formulated as a many-objective one,

and it is efficiently solved using the above introduced NSGA-II. This paper studies the optimal balance among three relevant design aspects when building an IaaS infrastructure for Big Data applications: reliability, cost, and net computing capacity (non- functional requirements—NFRs).

Using as inspirations the above-mentioned pioneering works, Ref. [16] presents a similar optimization problem in a quite similar real-world situation. In that case, the problem is formulated as a MOO one, trying to leverage two principal objectives: meeting of the characteristics fixed by the microservices developers (classification, public IP, disk space, RAM, and number of cores) and the fulfillment of the microservices' NFRs (location, availability, cost, performance, and legal level).

These two works are especially inspiring for this research, the synergy among Big Data and cloud computing encounters in IaC a practical cloud model by which organizations can deploy Big Data functionalities externally with the independence of the service provided and in a cost-efficient fashion [17]. Thus, instead of buying hardware materials, users can buy IaaS on-demand usage time depending on their specific demands, in a similar manner to electricity billing [18].

Lastly, it is noteworthy that the literature is scarce in studies focused on the optimal deployment of infrastructural elements for giving an answer to a specific service. A slightly related example can be found in [19]. This study introduced an automated system for answering the sizing cluster question by means of diverse functionalities as tailored cluster resources, performance prediction, task scheduling procedure, or configuration requirements. In any case, the system implemented in this research does not offer a sole solution to infrastructure definition but rather assists the user in the design process, giving an answer to the most frequent doubts.

2.2. Self-Learning with AI for Data Streams in the Presence of Concept Drift and Anomalies

Nowadays, many machine learning models in production are still static, i.e., they were developed and trained by data scientists or researchers on historical data, and from that point on, they will not be able to incorporate new knowledge. In most real applications, data arrive in the form of fast streams, and new data characteristics or trends should be incorporated into the existing models. When they remain static, these models should be retrained on a fairly regular basis (daily or even more frequently). However, this is not very efficient because:

1. It implies that an expert would have to be focused on deciding which is the best moment to train the models again.
2. Nowadays, data are produced in the form of fast streams.
3. Data are affected by non-stationary phenomena that occur fast, and a human cannot successfully detect changes in a real-fashion environment.

Therefore, some level of automation (self-learning) is crucial, and the state of the art is ready to provide us with some interesting solutions [20,21]. In the literature, we can find the term self-learning referring to unsupervised learning, self-supervised learning, self-labelling, or even reinforcement learning. In all cases, the idea is to automatically generate some kind of supervisory signal to solve some tasks, e.g., to learn data representations [22] or to automatically label a dataset. On other occasions, it refers to autoencoders (neural networks) [23]. However, in IaC, the other well-known meaning [24–27] refers to the ability of a model of:

- Ingesting new data as it becomes available (incremental learning);
- Detecting by itself changes (drifts) in data distribution and automatically retraining after this occurs;
- Warning the system when anomalies are detected;
- Self-optimizing and self-calibrating in case of performance issues due to concept drift or anomalies.

Under these circumstances, *self-learning* becomes a perfect ally in those scenarios where the change or anomaly may be present. An autonomous model allows systems to be more accurate and reliable in production for much longer periods of time. However, this is hard to achieve and presents several challenges:

- These models are based on algorithms that are usually more difficult to fine-tune;
- Overfitting can be a great concern;
- The stability of the model must be assured;
- False alarms (drift detections) may provoke that the retraining process is useless, even degrading the performance of the model;
- An anomaly must not be confused with a drift.

The latter point is not trivial [28] since one of the challenges for *concept drift* handling algorithms is not to mix the true drift with an outlier or noise, which refers to a once-off random deviation or anomaly [29,30]. No adaptivity is needed in the latter case; as Figure 1 shows, the concept drift phenomenon can appear in different forms: abrupt (the new concept completely replaces the old one in just one step), gradual (the new concept replaces the old one in several steps), incremental (the new concept appears incrementally), recurrent (often related to seasonal changes), blip/noise/outliers, etc.

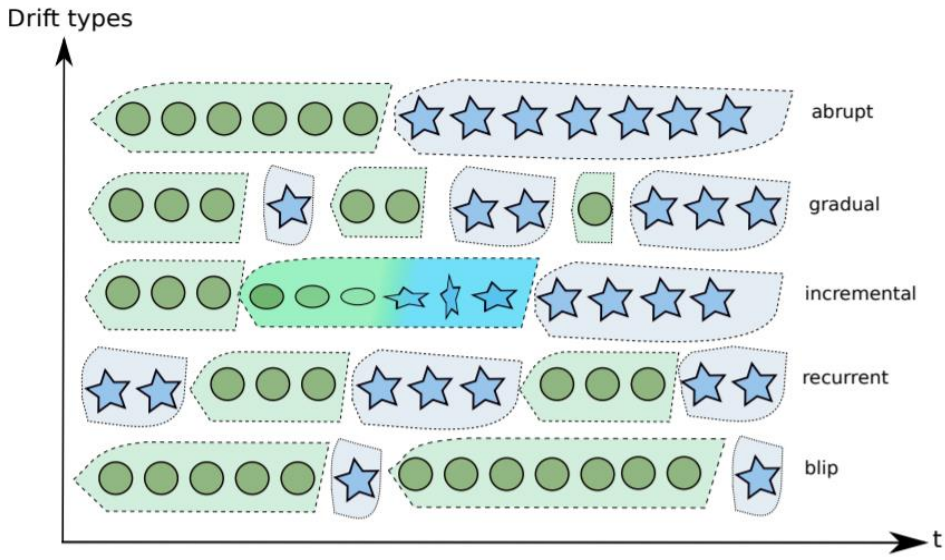

Figure 1. Types of drift according to severity and speed of changes, and noisy blips. Here the stars and circles represent the prevailing concept at every time instant [31].

Finally, it is worth mentioning the cyber risk component that is usually present in cloud technologies and concretely in those that deal with real-time data [32]. In order to mitigate these kinds of risks, PIACERE incorporates a specific component of self-learning for identifying security issues, such as the use of anomaly detection mechanisms to early detect attacks, unauthorized accesses, etc.

The next section provides the required background on these relevant topics.

2.2.1. Data Stream Analysis

Applications generating huge amounts of data in the form of fast streams are increasingly prevalent. These applications collect data from almost any source and analyze them to find answers that enable cost and time reductions, new product developments, optimized offerings, smart decision making, etc., or try to improve the operation in IaC. In these scenarios, instead of all training data being available from the beginning, data are often received over time in streams of samples or batches. Data streams are the basis of the real-time analysis, which is composed of sequences of items, each having a timestamp and thus a temporal order. Data can be previously labeled or not, implying supervised,

unsupervised, and semi-supervised techniques. A stream data environment shows several particularities [33] that we should consider when designing our algorithms:

- Each sample or batch is processed only once on arrival. Stream data analysis solutions should be able to process information sequentially, accordingly to its arrival. These solutions must not put the resources (mainly memory space and processing time restrictions) at risk;
- The processing time must be small and constant, without exceeding the ratio in which new samples arrive. Otherwise, some kind of temporal storage should be considered;
- The stream data analysis solution should use only a preallocated amount of main memory;
- The model/algorithm on which this stream data analysis solution is based should be completely trained before the next sample arrives.

In the case at hand, data also arrive in the form of data streams, and thus they may suffer anomalies and concept drift phenomenon, as we will demonstrate later. Finally, it deserves mentioning here that data streams of the monitored infrastructure may be time series, in which case the streaming strategy should be similar to [34] due to the fact that the temporal dependence emerges.

2.2.2. Concept Drift Detection

The data generation process in these real-time applications is not always stationary because it is subject to dynamic externalities that affect the stationarity of such data streams, e.g., seasonality, errors, etc. This causes such applications to suffer from the concept drift phenomenon. The predictive models that are trained over these data streams may become obsolete and have problems adapting suitably to the new conditions. Thus, in these scenarios, there is a pressing need for drift detection and adaption algorithms that detect and adapt to these changes as fast as possible in order to keep the applications updated and providing a good performance [35].

Consequently, drift detection turns into a relevant factor for those active mechanisms that need a triggering mechanism to perform an adaptation after drift occurs [36]. A drift detector should estimate the time instant at which change occurs over the data stream so that when the detection appears, the adaptation mechanism is applied to the base learner in order to avoid the degradation of its predictive performance. The successful design of an effective detector is not straightforward, yet it is primordial to achieve a more reliable system. The way to find the best strategy for concept drift detection still remains an open research issue, as confirmed in [36]. This challenge to find the best universal solution becomes evident in the most recent comparison among drift detectors carried out by [37]. In light of the results achieved in the [37] manuscript, we can realize that there is no method with the best metrics or even showing the best performance in most cases. We can state that the ideal goal is to develop detectors (1) that detect all existing drifts in the stream; (2) with low latency; (3) with as few as possible false alarms; (4) with as few as possible missed detections; and (5) that minimize the distance of the true positive detections, always assuring a good classification performance. Therefore, as there is not an ultimate detector, we will have to choose one, depending on the characteristics of the application or scenario, giving more importance to some metrics than others.

2.2.3. Anomaly Detection

Data analysis nowadays faces a number of challenges. One of them has been extensively studied due to its importance in the field, anomaly detection. When analyzing real-world data, data that differ from the norm can be found; such data are called an anomaly or outlier. Anomalies can be caused by inaccurate concepts. Hawkins [38] defines an outlier as *"an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism"*. Anomalies are also referred to as abnormalities, deviants, or outliers in the data mining and statistics literature [39].

Anomaly detection provides a set of algorithms and techniques that can be used to spot instances dissimilar to all others.

In recent years, an important growth of deep neural networks, a subset of the machine learning field, has been seen, with astonishing outcomes in different application areas, also when applied to anomaly detection [40,41]. Therefore, deep learning-based anomaly detection (DAD) algorithms have obtained a privileged position and are one of the focus areas. It is important to note that boundaries between abnormal and normal data are not precisely defined in evolving environments. This lack of boundaries represents challenges for both conventional and deep learning techniques.

Due to the large-scale nature of the data to be analyzed in the cloud continuum due to the heterogeneity of the resources, it becomes nearly impossible for the traditional machine learning techniques to scale and find anomalies properly. DAD techniques can handle this large amount of data and are also able to learn hierarchical discriminative features solving the problem end-to-end by removing the need of developing manual features by domain experts.

2.3. Self-Healing Techniques for the Autonomous Implementation of Corrective Actions at Run-Time

Once the IaC is deployed to create and deploy the software components into production, it becomes necessary to monitor the run-time environment behavior to detect a deterioration of quality-of-service parameters and take according actions to improve the resource usage of the overall system.

Thereby, it becomes possible to take actions to prevent the quality of service and service level agreement violations. These actions usually include a redeployment or readaptation of the application to the new situation, implying the selection and configuration of new resources totally (for all the components) or partially (substituting only the failing resource), as well as porting the affected components from one resource to another, and setting up the network (i.e., specific bandwidth or dedicated network connections). Thus, changes in the IaC need to be made so that the new resources are created and configured. All this needs to be done without compromising the business continuity, especially when tackling critical infrastructures where the response time becomes vital, and the system must be resilient and able to operate in different scenarios, including lack of connectivity that can be replaced by alternative communication technologies (i.e., new edge nodes).

Application of self-healing, referring to autonomous and responsible behavior of the applications to changes in the execution environment, has been a challenge since cloud and distributed computing paradigms appeared on the software operation gameboard. Several solutions have been proposed for specific scenarios such as IoT [42,43] or traditional cloud environments [44]. Other solutions are focused on specific steps in the self-healing and self-configuration process [45]; in the resolution of specific problems such as scalability [46] or trust enforcement [47]; or in the usage of specific novel architectures such as multi-agents to address the problem [21]. However, the cross/multi-layer and the networking aspects are challenges that have not been addressed, nor faced the problem in a generic way, covering the whole self-healing process from the discovery, optimization, and configuration of the resources, to the network preparation and deployment of all the software layers. All these actions can be automatized through IaC creation and update.

Self-healing mechanisms can include several actions, going from the starting up of the failing resource with a new one to the reconfiguration of the network or the orchestration of portability workflows so that business continuity is ensured. Several approaches have been applied to automatically create computational resources (usually virtual machines), but seldom few initiatives can be found when addressing heterogeneous environments from different nature (edge and cloud) and networking set up. In such a complex environment, self-healing actions may need dynamic orchestration of the services used for the redeployment following the proper workflow as well as network resources re-allocation and setting up.

There are a wide variety of tools that tackle the problem of redeployment and reconfiguration of existing infrastructure in response to an event or business need. The first are the so-called configuration management tools. These tools promote the creation and configuration of software environments based on a set of prerequisites in existing infrastructure. To summarize, these tools manage a set of already existing devices. They require a secure connection with such devices and are able to execute commands on them with an existing user of the system. Some of the most common configurations available manage users, services, files, and packages, among others.

All the tools analyzed were developed at least ten years ago; hence, they are considered stable enough to be adopted in production environments. Ansible [48] and Saltstack [49] both promote the use of YAML as the configuration language, whereas Puppet [50] and Chef [51] have their own domain-specific language. In terms of language style, Ansible and Chef have adopted a procedural language style, in which a series of steps are defined to achieve the final state. On the other hand, Puppet and SaltStack utilize a declarative language in which the final state is defined, but how that state will be reached is up to the tool itself. Finally, we try to measure the popularity of each tool; it is hard to establish such a thing, especially since Chef and Puppet are such frequently used words in the English language. Therefore, we opt to compare the number of stars on their respective GitHub repositories. Using this metric, Ansible is by far the most popular having more than three times as many stars as the second one, SaltStack.

Secondly, infrastructure provisioning tools differ from the tools mentioned above in which these do not communicate with the servers or devices directly; instead, they operate at a higher level by communicating with the cloud API. By doing so, they can elastically create, modify, and destroy the infrastructure. The first of these tools is Heat [52], which is a service to orchestrate cloud applications using templates. It promotes the use of templates for the description of the infrastructure in a format that aspires to be both human- and machine-readable. However, it focuses on OpenStack and lacks integration with other cloud providers. The second one is AWS CloudFormation [53]; it is a set of tools provided by AWS that promises an easy manner to model a set of resources that will be deployed in the AWS cloud. It provides means for deploying, reconfiguring, and tearing down cloud resources that may span across different regions and accounts. Both Heat and AWS CloudFormation provide similar functionalities but focus on particular cloud providers. In this sense, Terraform [54] tries to unify the provisioning of cloud resources of different providers in a common tool. It promotes the use of a unified workflow in order to manage cloud resources: (1) write the infrastructure as code, (2) plan and preview the changes before being applied, and (3) apply the changes in a reproducible manner. Terraform has gained a lot of traction in the last few years, and it is commonly used for managing the infrastructure of projects in both industry and academia. All of these tools excel at managing the lifecycle of cloud resources and delegate in the aforementioned tools the configuration of such resources.

Thirdly, managing the lifecycle of applications from the development phase to deploying them in production is a process that has been thoroughly scrutinized in the last few years. Methodologies such as DevOps and continuous integration and deployment practices tackle this process and promote a seamless deployment of applications in production environments, minimizing the needed time for such a process and reducing errors. To this end, applications are commonly being shipped as containers, and several applications for the management of such containers have appeared in recent years. Docker Swarm [55] is a solution that excels for being easy to set up and manage and provides easy integration with the docker containers technology, the de facto solution for the containerization of applications. Kubernetes [56] arose from Google's Borg [57] and offers a myriad of possibilities to manage the lifecycle of container applications, provides built-in auto scaling and fine configuration, and is offered by many of the public cloud providers as a service, which promotes the migration of existing solutions in an easy manner.

In this sense, some standards have appeared, such as TOSCA [58], which provides a way to enable portable automated deployment and management of composite applications. Some of the challenges that TOSCA tries to alleviate are the automated management of such applications, the portability of the applications, and the interoperability and reusability of applications components. Yet another tool addressing a similar challenge is CAMEL [59] which enables the specification of aspects such as provisioning, deployment, service level, monitoring, scalability, providers, and users, among others. In general, CAMEL can be used to describe different aspects of self-adaptive cross-cloud applications.

However, other tools approach this problem in a more general way, without the use of containerized applications. Apache Brooklyn [60] is a framework for modeling, monitoring, and managing applications primarily in cloud environments. It is envisioned to be a tool that unifies the provisioning, deployment, and monitoring of an application in a single tool. It does so by using the concept of blueprints, which is a YAML file in which an application is defined. The first benefit of using blueprints is that they are composable, and a blueprint of a process can be incorporated into another one easily. The second benefit is that they can be treated as source code, which enables testing, tracking, and versioning them and integrating them as part of the DevOps process. Similarly, Spinnaker [61] defines itself as a multi-cloud continuous delivery platform envisioned to release software with high velocity and confidence. It is used for managing the cloud resources as well as to manage the application deployment by defining a pipeline that defines the set of steps for an application to be updated in a production environment.

3. PIACERE Approach for Optimized and Self-Healed IaC

In this section, we present the PIACERE position and approach to support DevOps teams in the optimization and self-healing of the infrastructural elements and the code that generates these (IaC) in complex distributed cloud continuum environments. The main envisioned PIACERE components for the optimization and self-healing of IaC are depicted in Figure 2.

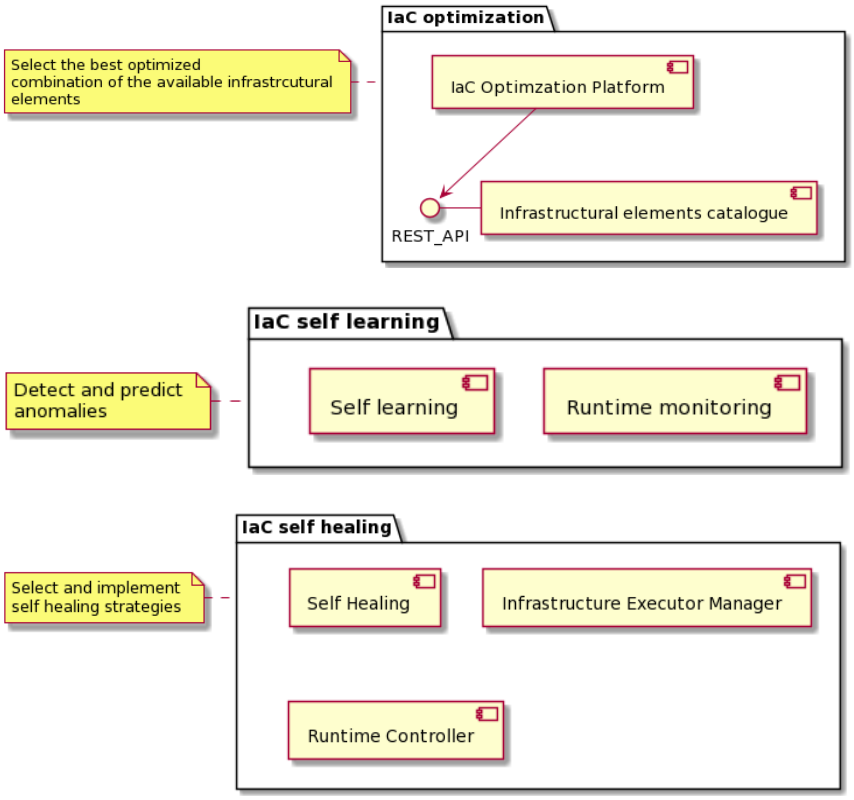

Figure 2. PIACERE approach for optimized and self-healed IaC.

Each phase of optimization, self-learning, and self-healing provides several advantages both at the pre-deployment and run-time phase of the IaC management. Figure 2 represents the main logical building blocks of the components providing support to each of the processes, optimization of the IaC, autonomous prediction of malfunctioning on the IaC (IaC self-learning), and automatic self-recovery mechanism once a failure is detected or predicted. Each of these logical components is further explained in the subsequent Sections 3.1–3.3.

3.1. IaC Optimization

As mentioned in Section 2, the optimization problem to be formulated in PIACERE consists of having a service to be deployed and a catalogue of infrastructural elements, with the principal challenge of finding an optimized deployment configuration of the IaC on the appropriate infrastructural elements that best meet the predefined constraints (e.g., types of infrastructural elements, the fulfillment of the microservices' NFRs such as location or availability, and so on).

Arguably, the problem modeled will be of MOO nature, at least, with the positivity of evolving it to a many-objective one. The system to be deployed should also meet not only the defined functional requirements but also some optimization related to non-functional requirements.

In this sense, it is interesting to show here a brief definition of these concepts, which are crucial for properly designing the solving systems. Regarding functional optimization requirements, they can be referred to as *what the system should do*, and it is a concept strictly related to the main objective the system is built for. Logically, the establishment of these requirements leads to the building of the objective function, or in the specific MOO case of PIACERE, objective functions. Regarding non-functional requirements, they can be defined in different ways. Davis defines them as *the required of overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability* [62]. These non-functional requirements are essential for the proper election of solving method, and the non-consideration of them can suppose the re-design of the whole research, involving both economic and time costs.

Having said this and considering that the requirements that should be met in the PIACERE context will be high-demanding, it is a wise decision to consider the use of a well-reputed framework that can offer the possibility of implementing and applying different algorithms of different nature. This framework should be flexible enough not only to use the algorithms considered but also for adapting, modifying, and merging them, seeking an ultimate algorithm that can perfectly adapt to the requirements needed in PIACERE. Furthermore, this framework should be focused on the resolution of optimization problems considering different objectives, but undoubtedly, it should be oriented to the resolution of MOO problems.

In this regard, some of the frameworks that will be considered are ECJ, HeuristicLab, jMEtal, or PlatEMO. Deeming the proven strength of these frameworks, as part of the PIACERE project, experimentation will be made comparing the performance of algorithms drawn from some of these platforms. That is, the objective is not just to decide which is the best algorithm for the use cases but also the most efficient framework to implement it. This is important in case non-functional requirements change along with the execution of the project. In this context, a framework that allows a fast adaptation and contains plenty of efficient algorithms is appreciated.

There could be several reasons for choosing one framework or another. The programing language could be one of these reasons, along with the orientation of the frameworks on single- or multi-objective optimization problems. The kind of software licenses can also be crucial for disabling the choice of a particular package.

Once the framework is defined, the algorithm that gives a response to the real-world optimization should be implemented. In this regard, not only the direct application of existing algorithms will be conducted. In PIACERE, new solving mechanisms will

be developed and tested, giving special importance to the merging and enhancing of well-reputed mechanisms. Furthermore, in order to fill the non-functional requirements, different optimization paradigms will also be considered. More specifically, the adequacy of research topics such as transfer optimization will be explored. Transfer optimization is a relatively new knowledge field within the wider area of optimization, whose main objective is to exploit what has been learned for optimizing one given optimization problem toward tackling another related or unrelated problem.

Finally, once the solving techniques are developed, different kinds of fine-tuning procedures will be carried out for improving their performance as much as possible. This process can be divided into two different steps. The first one is devoted to the proper optimization of the code, which can be conducted through the application of profiling tools. Due to these tools, code parts presenting high time consumption can be detected, leading to their subsequent rewriting. The second step of this fine-tuning process is devoted to the efficient adjustment of the parameters of the algorithms. This way, the efficacy can be improved even more. This adjustment can be conducted following two different approaches: ad-hoc pilot tests and automatic configuration. Due to the nature of PIACERE, we will embrace the first of these approaches, which is the most employed by researchers and also the most recommended one when having a high degree of expertise.

3.2. IaC Self-Learning

As is well-known, IaC suffers from trustworthiness aspects, which are often left for the end of the cycle, for once the code is already in operation, it is already too late. The errors provoked by abrupt changes, e.g., in the deployment process, may be expensive to correct, affecting the business continuity of the application. For this reason, PIACERE adopts a self-learning approach, where these errors (drifts and/or anomalies) will be early detected, triggering a self-healing mechanism that will optimize the IaC parameters to adapt to the new situation.

The monitoring system will provide a self-learning module with real-time data (data streams/time series) composed of many of the informative metrics/variables of the running operating system such as CPU load, memory occupation, consumed bandwidth, among many others, with significant discriminative capacity that can provide significant clues. These metrics will be properly preprocessed so that they can be consumed by the algorithm without penalty. Then, the self-learning module will be capable of performing incremental learning by acquiring new knowledge every time a new data instance is received, and it will be composed of two mechanisms: a drift detector and an anomaly detector. Both are addressed to guarantee the constant high-level performance of IaC, being decisive to assure the truthfulness and completeness of the metrics, which will have a substantial impact on the detection of a drift/anomaly within the Failure/SLA Violation early prediction.

Due to the fact that PIACERE monitoring data have the form of time series, and thus the temporal dependence is present, we will consider a drift detection strategy based on [34,63]. As these papers show, the presence of temporal dependence leads us to consider a different detection approach and a different set of metrics to evaluate the performance of the drift detector mechanism. These references show us that it is not enough to show the performance of a change detector working with a classifier, even though a no-change detector (it is a no-change detector in the sense that it is not detecting the change in the stream, just outputs change every x instance) can obtain better results than known detectors of the literature. This is due to the temporal dependence. Finally, it deserves mentioning that the drift detection in time series is often referred to as "change point detection".

The cloud-based distributed nature of the PIACERE components, including the large-scale generated data, will require a distributed approach along with deep learning architecture based on [64,65]. These papers show how to solve anomaly detection challenges in resource-scarce environments or with the requirement of not being computationally intensive but achieving high detection accuracy and false alarm rates. The applied approaches

are also able to adapt to new changes in a non-stationary environment, such as the one present in PIACERE.

Therefore, the self-learning module will warn the self-healing component once risky conditions are detected and threatening the QoS of the IaC deployment, which will eventually start up the optimization process aiming at obtaining a new optimal IaC infrastructural and resource configuration.

3.3. IaC self-Healing

The IaC lifecycle comprises the following stages: infrastructure provisioning, configuration management, application deployment, and infrastructure monitoring. The IaC Self-Healing in PIACERE aspires to cover some of these stages. In particular, the IaC Executor Manager (IEM) will oversee (1) the infrastructure provisioning, where the initial set of infrastructural elements defined by the developer will be deployed; (2) the configuration management of the aforementioned elements, where the software requirements and nuances of all such elements are configured; and (3) the application deployment in which the application workflow of the different use cases is deployed on the infrastructure. This process will need to be executed each time the creation of the infrastructural layer is needed, in the first deployment, or in subsequent redeployments triggered by the self-learning module once an anomaly has been detected.

All these tasks are time-consuming, not fully automated, and "stage" dependent; hence, specific tools will be evaluated and selected for each of these stages.

In addition, IEM aspires to minimize the operationalization of heterogeneous application workflows and reduce the downtime of such workflows between redeployments. One of the techniques to achieve these goals is to provide a wide variety of interfaces for different IaC languages and tools that will assist the use cases during their workflows. Another technique is to provide an interface for the redeployment and reconfiguration of the infrastructure, which enables to shortcut the heavyweight and time-consuming process of tearing down and provisioning the entire ecosystem. This way, only specific pieces of the architecture will be provisioned, fine-tuned, and deployed, which reduces not only the total time but also the application downtime when performing these IaC adjustments.

To provide this functionality, it is key to select the appropriate tool for each of the stages defined above. Some of the particularities that will assist in achieving this overarching goal are declarative languages and idempotence. The former aids in defining what the final picture should comprise, but how to reach that state is up to the tool itself, whereas the latter is a programing principle that promotes that no matter how many times the IaC code will be executed, the same stage will be reached. Following these two principles guarantees the maintainability, reusability, and reproducibility of subsequent IEM executions.

As part of the self-healing process, the PIACERE self-healing component plans to provide, develop, and implement a tool to orchestrate the different actions that need to be taken once an anomaly or an improvement in the execution environment is detected by the self-learning module. The events will be classified in order to know what type of action needs to be executed, as some mitigation actions can imply modifying IaC while others don't, thus impacting the decision of which other PIACERE modules to execute. For example, the criticality of the incidence (predicted versus already happened) and the need for a complete or partial redeployment of the source causing the event (the infrastructural element or the container deployed on top of it) may impact the mitigation action. In this sense, several self-healing activities could be put in place with different consequences for the system in general and for the IaC in particular: reboot the machine (no new IaC is needed), vertical (more memory, more disk, more cores, different disks quality, more network capability) and/or horizontal scalability (adjust Kubernetes deployment scalability, add node to Kubernetes cluster, new server for a given component, new server in a different zone), new deployment configuration needed (partial and or complete), etc.

PIACERE will consider the self-recovery from failures or non-compliance of NFRs, such as the performance of the infrastructural element (i.e., disk, memory, CPU, etc.), the

cost, or the availability. Furthermore, security events and incidents will also be considered through on-demand (HTTP/HTTPS endpoints, infrastructure vulnerabilities such as misconfigurations or CVEs) and continuous (Integrity of system files, Kernel/network activity, etc.) monitoring. Therefore, PIACERE self-healing component will autonomously decide, optimize, plan, orchestrate, and execute the proper set of activities to assure a successful deployment through the requests to the different components. It will include a catalogue of different strategies that can be applied when moving applications components (i.e., porting data or stateful components) as well as the implementation of a set of strategies that can be applied automatically when re-deploying the components and setting up the re-optimized infrastructural layer. The selection of which strategy will be implemented over another one will come from the needs and prioritization of the pilots to be supported in PIACERE.

The mitigation actions will be selected from a set of predefined mitigation actions stored in the knowledge database of the self-healing component, and the subsequent process will be followed:

1. Classification of the event: Classification of the event detected by the self-learning component. The events identified can be of different nature for different reasons:
    a. Predicted failure vs. already detected failure: available time for the self-healing process.
    b. The main cause of the event: it can be caused by a software failure or by a CSLA violation in the infrastructural element.
    c. Others to be analyzed: NFR affected, components affected, etc.
2. Selection of a self-healing strategy: Based on the initial classification and on the ruleset, the best self-healing strategy will be selected. This strategy may imply selecting a new set of infrastructural elements and consequently regenerating the IaC with these new requirements so that the new deployment schema is realized. In this case, the self-healing strategy should cover not only the bringing up of the new infrastructural elements but also the teardown of the previous infrastructure.
3. Orchestration of the self-healing process: Once the strategy is selected, it has to be executed. The different modules in charge of implementing the self-healing activities need to be executed properly. In PIACERE, the run-time controller will be responsible for orchestrating the process, the related tasks, and the relevant components.

4. Conclusions and Future Work

The growth of cloud services in recent years has led to more and more complex applications that leverage the power and ubiquity of computing capabilities and data in the cloud. Such applications require, in turn, new approaches for the creation, management, and operation of the cloud infrastructure, such as IaC.

This article has presented research carried out as part of the PIACERE project that studies the utilization of AI techniques and methods for the optimization of the operation of cloud continuum applications and the correspondent infrastructural code or IaC. We have shown how several processes from the application's run-time lifecycle can be improved by the incorporation of optimization techniques, data stream analysis, concept drift, and anomaly detection methods. We also introduced the concept of self-healed application to refer to applications that can be self-reactive to anomalies in their run-time.

The novel concepts of the PIACERE IaC optimization, IaC self-learning, and IaC self-healing are the key findings of the work. The IaC optimization is based on functions that have to meet functional requirements and optimize non-functional requirements, solving a MOO problem with the use of reputed frameworks and paradigms as transfer optimization. The IaC self-learning is based on the early detection of drift and anomalies in the monitored real-time data streams of the cloud infrastructure, being also capable of performing incremental learning. Finally, the IaC self-healing, which will be triggered when such anomalies are detected, is in charge of performing the corrective actions at run-time to maintain the status of the system within the predefined limits. Actions are

selected depending on the triggering event and that could imply the deployment of new optimized infrastructure, thus calling to the IaC optimization and closing the loop.

The research is foreseen to further advance with future steps that will include the structural and behavioral architectural description of the PIACERE optimization, self-learning, and self-healing components described in the paper, and also with the development of the corresponding POCs (proof of concept) or minimum versions of the components that will serve to validate the presented approach.

Author Contributions: Conceptualization, J.A. and L.O.-E.; Investigation, J.A., L.O.-E., E.O., J.L.L., I.M. and J.D.d.A.; Methodology, J.A. and L.O.-E.; Writing—original draft, J.A., L.O.-E., E.O., J.L.L., I.M. and J.D.d.A.; Writing—review and editing, J.A. and I.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the European project PIACERE (Horizon 2020 research and innovation Program, under grant agreement no 101000162).

Data Availability Statement: Not applicable.

Acknowledgments: We thank our colleagues from the PIACERE consortium who provided insight and expertise that greatly assisted the research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lynn, T.; Xiong, H.; Dong, D.; Momani, B.; Gravvanis, G.; Filelis-Papadopoulos, C.; Elster, A.; Khan, M.M.Z.M.; Tzovaras, D.; Giannoutakis, K.; et al. CLOUDLIGHTNING: A Framework for a Self-organising and Self-managing Heterogeneous Cloud. In Proceedings of the 6th International Conference on Cloud Computing and Services Science, Rome, Italy, 23–25 April 2016; pp. 333–338. [CrossRef]
2. Alonso, J.; Orue-Echevarria, L.; Escalante, M.; Benguria, G. *DECIDE: DevOps for Trusted, Portable and Interoperable Multi-Cloud Applications towards the Digital Single Market*; Parque Científico y Tecnológico de Bizkaia: Bizkaia, Spain, 2017. [CrossRef]
3. Kennedy, J. *Swarm Intelligence*; Springer: Berlin, Germany, 2006; pp. 187–219. [CrossRef]
4. Zedadra, O.; Savaglio, C.; Jouandeau, N.; Guerrieri, A.; Seridi, H.; Fortino, G. *Towards a Reference Architecture for Swarm Intelligence-Based Internet of Things*; Springer: Berlin, Germany, 2018; pp. 75–86. [CrossRef]
5. Darwish, A.; Hassanien, A.E.; Das, S. A survey of swarm and evolutionary computing approaches for deep learning. *Artif. Intell. Rev.* **2019**, *53*, 1767–1812. [CrossRef]
6. Marler, R.; Arora, J. Survey of multi-objective optimization methods for engineering. *Struct. Multidiscip. Optim.* **2004**, *26*, 369–395. [CrossRef]
7. Osaba, E.; Martinez, A.D.; Del Ser, J. Evolutionary Multitask Optimization: A Methodological Overview, Challenges and Future Research Directions. *arXiv* **2021**, arXiv:2102.02558. Available online: http://arxiv.org/abs/2102.02558 (accessed on 23 March 2021).
8. Del Ser, J.; Osaba, E.; Molina, D.; Yang, Y.-S.; Salcedo-Snaz, S.; Camacho, D.; Das, S.; Suganthan, P.N.; Coello, A.C.; Herrera, F. Bio-inspired computation: Where we stand and what's next. *Swarm Evolut. Comput.* **2019**, *48*, 220–250. [CrossRef]
9. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
10. Zitzler, E.; Laumanns, M.; Thiele, L. SPEA2: Improving the strength pareto evolutionary algorithm. *Comput. Sci.* **2001**, *103*. [CrossRef]
11. Zhang, Q.; Li, H. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Trans. Evol. Comput.* **2007**, *11*, 712–731. [CrossRef]
12. Nebro, A.J.; Durillo, J.J.; García-Nieto, J.; Coello, C.C.; Luna, F.; Alba, E. SMPSO: A new PSO-based metaheuristic for multi-objective optimization. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making(MCDM), Nashville, TN, USA, 30 March–2 April 2009; pp. 66–73. [CrossRef]
13. Nebro, A.J.; Durillo, J.J.; Luna, F.; Dorronsoro, B.; Alba, E. MOCell: A cellular genetic algorithm for multiobjective optimization. *Int. J. Intell. Syst.* **2009**, *24*, 726–746. [CrossRef]
14. Bechikh, S.; Elarbi, M.; Ben Said, L. *Many-Objective Optimization Using Evolutionary Algorithms: A Survey*; Springer: Cham, Switzerland, 2016; pp. 105–137. [CrossRef]
15. Alonso, J.; Stefanidis, K.; Orue-Echevarria, L.; Blasi, L.; Walker, M.; Escalante, M.; Lopez, M.; Dutkowski, S. DECIDE: An Extended DevOps Framework for Multi-cloud Applications. In Proceedings of the 2019 3rd International Conference on Cloud and Big Data Computing, Oxford, UK, 28–30 August 2019; pp. 43–48. [CrossRef]
16. Arostegi, M.; Torre-Bastida, A.; Bilbao, M.N.; Del Ser, J. A heuristic approach to the multicriteria design of IaaS cloud infrastructures for Big Data applications. *Expert Syst.* **2018**, *35*, e12259. [CrossRef]

17. Hashem, I.A.T.; Yaqoob, I.; Anuar, N.B.; Mokhtar, S.; Gani, A.; Khan, S.U. The rise of "big data" on cloud computing: Review and open research issues. *Inf. Syst.* **2015**, *47*, 98–115. [CrossRef]
18. Rappa, M.A. The utility business model and the future of computing services. *IBM Syst. J.* **2004**, *43*, 32–42. [CrossRef]
19. Herodotou, H.; Dong, F.; Babu, S. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In Proceedings of the 2nd ACM Symposium on Cloud Computing, Cascais, Portugal, 26–28 October 2011; p. 18. [CrossRef]
20. Nawaratne, R.; Alahakoon, D.; De Silva, D.; Chhetri, P.; Chilamkurti, N. Self-evolving intelligent algorithms for facilitating data interoperability in IoT environments. *Futur. Gener. Comput. Syst.* **2018**, *86*, 421–432. [CrossRef]
21. Rajput, P.K.; Sikka, G. Multi-agent architecture for fault recovery in self-healing systems. *J. Ambient. Intell. Humaniz. Comput.* **2020**, *12*, 2849–2866. [CrossRef]
22. Doersch, C.; Zisserman, A. Multi-task self-supervised visual learning. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2051–2060.
23. Gogna, A.; Majumdar, A. Semi supervised autoencoder. In Proceedings of the International Conference on Neural Information Processing, Kyoto, Japan, 16–21 October 2016; pp. 82–89.
24. Pathak, K.; Kapila, J. Reinforcement evolutionary learning method for self-learning. *arXiv* **2018**, arXiv:1810.03198.
25. Cerquitelli, T.; Proto, S.; Ventura, F.; Apiletti, D.; Baralis, E. Automating concept-drift detection by self-evaluating predictive model degradation. *arXiv* **2019**, arXiv:1907.08120.
26. Lu, J.; Liu, A.; Song, Y.; Zhang, G. Data-driven decision support under concept drift in streamed big data. *Complex. Intell. Syst.* **2019**, *6*, 157–163. [CrossRef]
27. Ramakrishnan, A.K.; Preuveneers, D.; Berbers, Y. Enabling Self-learning in Dynamic and Open IoT Environments. *Procedia Comput. Sci.* **2014**, *32*, 207–214. [CrossRef]
28. Carreño, A.; Inza, I.; Lozano, J.A. Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework. *Artif. Intel. Rev.* **2020**, *53*, 3575–3594. [CrossRef]
29. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly detection: A survey. *ACM Comput. Surv.* **2009**, *41*, 1–58. [CrossRef]
30. Gomes, H.M.; Read, J.; Bifet, A.; Barddal, J.P.; Gama, J. Machine learning for streaming data: State of the art, challenges, and opportunities. *ACM SIGKDD Explor. Newslet.* **2019**, *21*, 6–22. [CrossRef]
31. López Lobo, J. New Perspectives and Methods for Stream Learning in the Presence of Concept Drift. Ph.D. Thesis, University of Pais Vasco, Barrio Sarriena, Spain, 2018.
32. Radanliev, P.; De Roure, D.C.; Nurse, J.R.C.; Montalvo, R.M.; Cannady, S.; Santos, O.; Maddox, L.; Burnap, P.; Maple, C. Future developments in standardisation of cyber risk in the Internet of Things (IoT). *SN Appl. Sci.* **2020**, *2*, 1–16. [CrossRef]
33. Domingos, P.; Hulten, G. A General Framework for Mining Massive Data Streams. *J. Comput. Graph. Stat.* **2003**, *12*, 945–949. [CrossRef]
34. Žliobaitė, I.; Bifet, A.; Read, J.; Pfahringer, B.; Holmes, G. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Mach. Learn.* **2014**, *98*, 455–482. [CrossRef]
35. Bahri, M.; Bifet, A.; Gama, J.; Gomes, H.M.; Maniu, S. *Data Stream Analysis: Foundations, Major Tasks and Tools*; Wiley: Hoboken, NJ, USA, 2021; p. e1405.
36. Hu, H.; Kantardzic, M.; Sethi, T.S. No Free Lunch Theorem for concept drift detection in streaming data classification: A review. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2019**, *10*, e1327. [CrossRef]
37. Barros, R.S.M.; Santos, S.G.T.C. A large-scale comparison of concept drift detectors. *Inf. Sci.* **2018**, *451–452*, 348–370. [CrossRef]
38. Hawkins, D.M. *Identification of Outliers*; Springer: Berlin, Germany, 1980. [CrossRef]
39. Aggarwal, C.C. *Outlier Analysis*; Springer: Cham, Switzerland, 2017. [CrossRef]
40. Peng, H.-K.; Marculescu, R. Multi-Scale Compositionality: Identifying the Compositional Structures of Social Dynamics Using Deep Learning. *PLoS ONE* **2015**, *10*, e0118309. [CrossRef]
41. Javaid, A.; Niyaz, Q.; Sun, W.; Alam, M. A Deep Learning Approach for Network Intrusion Detection System. *Endorsed Transact. Safe.* **2016**, *3*, e2. [CrossRef]
42. Serhani, M.A.; El-Kassabi, H.T.; Shuaib, K.; Navaz, A.N.; Benatallah, B.; Beheshti, A. Self-adapting cloud services orchestration for fulfilling intensive sensory data-driven IoT workflows. *Futur. Gener. Comput. Syst.* **2020**, *108*, 583–597. [CrossRef]
43. Angarita, R.; Manouvrier, M.; Rukoz, M. An Agent Architecture to Enable Self-Healing and Context-aware Web of Things Applications. In Proceedings of the International Conference on Internet of Things and Big Data (IoTBD 2016), Rome, Italy, 23–25 April 2016. [CrossRef]
44. Gill, S.S.; Chana, I.; Singh, M.; Buyya, R. RADAR: Self-configuring and self-healing in resource management for enhancing quality of cloud services. *Concurr. Comput. Pract. Exp.* **2018**, *31*, e4834. [CrossRef]
45. Gao, H.; Huang, W.; Yang, X.; Duan, Y.; Yin, Y. Toward service selection for workflow reconfiguration:An interface-based computing solution. *Futur. Gener. Comput. Syst.* **2018**, *87*, 298–311. [CrossRef]
46. Toffetti, G.; Brunner, S.; Blöchlinger, M.; Spillner, J.; Bohnert, T.M. Self-managing cloud-native applications: Design, implementation, and experience. *Futur. Gener. Comput. Syst.* **2017**, *72*, 165–179. [CrossRef]
47. El-Kassabi, H.T.; Serhani, M.A.; Dssouli, R.; Navaz, A.N. Trust enforcement through self-adapting cloud workflow orchestration. *Futur. Gener. Comput. Syst.* **2019**, *97*, 462–481. [CrossRef]
48. RedHat Ansible. Available online: https://www.ansible.com/ (accessed on 27 January 2021).
49. Hatch, T.S. SaltStack Documentation. Available online: https://docs.saltproject.io/en/latest/ (accessed on 12 April 2021).

50. Webteam, P. Powerful Infrastructure Automation and Delivery | Puppet. Available online: https://puppet.com/ (accessed on 12 April 2021).
51. Chef Automate. Available online: https://www.chef.io/products/chef-automate (accessed on 12 April 2021).
52. Heat—OpenStack. Available online: https://wiki.openstack.org/wiki/Heat (accessed on 12 April 2021).
53. AWS CloudFormation—Infraestructura Como Código y Aprovisionamiento de Recursos de AWS'. Amazon Web Services, Inc. Available online: https://aws.amazon.com/es/cloudformation/ (accessed on 12 April 2021).
54. Terraform by HashiCorp, Terraform by HashiCorp. Available online: https://www.terraform.io/ (accessed on 12 April 2021).
55. Swarm Mode Overview Docker Documentation. Available online: https://docs.docker.com/engine/swarm/ (accessed on 12 April 2021).
56. Kubernetes. Available online: https://kubernetes.io/ (accessed on 12 April 2021).
57. Verma, A.; Pedrosa, L.; Korupolu, M.; Oppenheimer, D.; Tune, E.; Wilkes, J. Large-Scale Cluster Management at Google with Borg. In Proceedings of the Tenth European Conference on Computer Systems, Bordeaux, France, 21–24 April 2015; p. 18. [CrossRef]
58. Binz, T.; Breiter, G.; Leyman, F.; Spatzier, T. Portable Cloud Services Using TOSCA. *IEEE Internet Comput.* **2012**, *16*, 80–85. [CrossRef]
59. Rossini, A.; Kritikos, K.; Nikolov, N.; Domaschka, J.; Griesinger, F.; Seybold, D.; Romero, D.; Orzechowski, M.; Kapitsaki, G.; Achilleos, A. The Cloud Application Modelling and Execution Language (CAMEL). Available online: https://oparu.uni-ulm.de/xmlui/handle/123456789/4378 (accessed on 29 July 2021).
60. Home—Apache Brooklyn. Available online: https://brooklyn.apache.org/ (accessed on 12 April 2021).
61. Spinnaker, Spinnaker. Available online: https://www.spinnaker.io/ (accessed on 12 April 2021).
62. Davis, A.M. *Software Requirements: Objects, Functions, and States*; PTR Prentice Hall: Englewood Cliffs, NJ, USA, 1993.
63. Bifet, A. Classifier concept drift detection and the illusion of progress. In Proceedings of the International Conference on Artificial Intelligence and Soft Computing, Zakopane, Poland, 11–15 June 2017; pp. 715–725.
64. Luo, T.; Nagarajan, S.G. Distributed Anomaly Detection Using Autoencoder Neural Networks in WSN for IoT. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6. [CrossRef]
65. Kakanakova, I.; Stoyanov, S. Outlier Detection via Deep Learning Architecture. In Proceedings of the 18th International Conference on Computer Systems and Technologies, Ruse, Bulgaria, 23–24 June 2017; pp. 73–79. [CrossRef]