

Received October 2, 2020, accepted December 11, 2020, date of publication December 18, 2020, date of current version December 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3045768

# Managing Feature Compatibility in Kubernetes: Vendor Comparison and Analysis

EDDY TRUYEN<sup>1</sup>, NANE KRATZKE<sup>2</sup>, DIMITRI VAN LANDUYT<sup>1</sup>,  
BERT LAGASSE<sup>1</sup>, AND WOUTER JOOSEN<sup>1</sup>

<sup>1</sup>imec-DistriNet, Department of Computer Science, KU Leuven, 3001 Leuven, Belgium

<sup>2</sup>Department for Electrical Engineering and Computer Science, Technische Hochschule Lübeck (Lübeck University of Applied Sciences), 23562 Lübeck, Germany

Corresponding author: Eddy Truyen (eddy.truyen@cs.kuleuven.be)


This work was supported in part by the Adaptive Distributed Software (ADDIS) Project (Research Fund KU Leuven), and in part by the CloudTRANSIT Project (German Federal Ministry of Education and Research) under Grant 13FH021PX4.

**ABSTRACT** Kubernetes (k8s) is a kind of cluster operating system for cloud-native workloads that has become a de-facto standard for container orchestration. Provided by more than one hundred vendors, it has the potential to protect the customer from vendor lock-in. However, the open-source k8s distribution consists of many optional and alternative features that must be explicitly activated and may depend on pre-configured system components. As a result, incompatibilities still may ensue among Kubernetes vendors. Mostly managed k8s services typically restrict the customizability of Kubernetes. This paper firstly compares the most relevant k8s vendors and, secondly, analyses the potential of Kubernetes to detect and configure compatible support for required features across vendors in a uniform manner. Our comparison is performed based on documented features, by testing, and by inspection of the configuration state of running clusters. Our analysis focuses on the potential of the end-to-end testing suite of Kubernetes to detect support for a desired feature in any Kubernetes vendor and the possibility of reconfiguring the studied vendors with missing features in a uniform manner. Our findings are threefold: First, incompatibilities arise between default cluster configurations of the studied vendors for approximately 18% of documented features. Second, matching end-to-end tests exist only for around 64% of features and for 17% of features these matching tests are not well developed for all vendors. Third, almost all feature incompatibilities can be resolved using a vendor-agnostic API. These insights are beneficial to avoid feature incompatibilities already in cloud-native application engineering processes. Moreover, the end-to-end testing suite can be extended in currently unlighted areas to provide better feature coverage.

**INDEX TERMS** Computer systems organization, architectures, distributed architectures, cloud computing.

## I. INTRODUCTION

Vendor lock-in avoidance is a common problem for almost all cloud system and application engineers. The core components of their distributed and cloud-based applications like virtualized server instances and basic networking and storage can be deployed using commodity services. However, further services—that are needed to integrate these virtualized resources in an elastic, scalable, and pragmatic manner—are often not considered in standards. These integrating and “glueing” service types, which are crucial for almost every cloud application, are usually not provided in a standardized way. It seems that all public cloud service providers try

The associate editor coordinating the review of this manuscript and approving it for publication was Chintan Amrit .

to stimulate cloud customers to use their non-commodity convenience service “interpretations” to bind them to their infrastructures and higher-level service portfolios. As a result, a transfer to another cloud infrastructure is very often a time-consuming and expensive one-time exercise due to non-obvious technological bindings.

According to an analysis performed in 2016 [1], the percentage of these commodity service categories that are considered in standards like CIMI, OCCI, CDMI, OVF, OCI, TOSCA is even decreasing over the years. This decrease has mainly to do with the fact that new cloud service categories are released faster than standardization authorities can standardize existing service categories. Fig. 1 shows this effect by the example of Amazon Web Services (AWS) over the years. For a more detailed discussion, we refer to [2], [3].

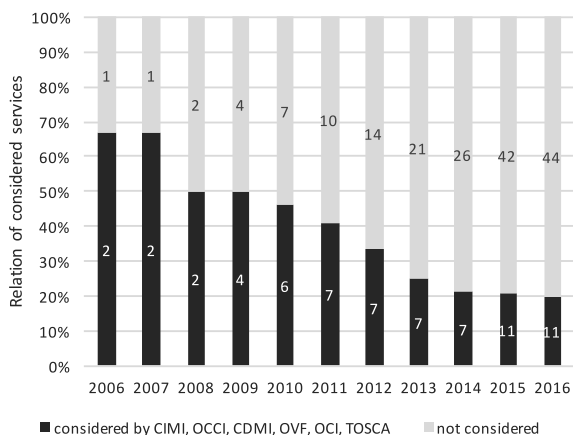


FIGURE 1. Decrease of standardization by the example of AWS.

Container orchestration platforms like Kubernetes, Mesos, Swarm, or Nomad emerged in recent years and provide nowadays what could be considered “a unifying cloud infrastructure” [4]–[6]. Standardization bodies like the OCI (Open Container Initiative), CNCF (Cloud Native Computing Foundation [7]) and the Kubernetes Special Interest Groups (SIGs) [8] arose around these trends. These industry-wide initiatives could and should ideally protect the customer from vendor lock-in.

More specifically, the CNCF has created a certification program for Kubernetes vendors that builds upon the end-to-end (e2e) testing suite of Kubernetes that is co-developed by the k8s SIGs and constitutes an essential part of the open-source k8s distribution [9], [10]. This program thus improves interoperability and portability of application and cluster resources across different Kubernetes vendors.

However, we have found that some resources of the RESTful API of Kubernetes may not be migrated between certified k8s vendors if the CNCF conformance program does not enforce certain functional features required for that particular k8s resource. The CNCF program cannot enforce some features because Kubernetes is highly customizable via various customization interfaces, and thus many features are considered optional.

Kubernetes vendors can freely decide how to encapsulate these customization interfaces from the customer: (i) one can offer the customization interfaces as-is, (ii) provide a proprietary, higher-level configuration interface that internally maps to one or more k8s customization interfaces or (iii) one hides the customization interface completely and locks it in a particular setting.

When the customization interfaces are locked by a vendor, likely, some features cannot be activated. As a result, it is not possible to migrate to that vendor any k8s resources that depend upon that feature. Neither is it possible to replicate these k8s resources in an interoperable manner across a federation of cloud providers [11], [12].

These type of vendor lock-in and migration problems typically appear the most for Kubernetes vendors of the

hosted product type. This type of vendor typically offers proprietary customization interfaces but also lock-in various interfaces.

The rationale for selecting vendors of the hosted product type is that the customer is offered a higher-level user interface that substantially eases the maintenance of the k8s platform and manages the availability and service level of the platform. In opposition, CNCF-certified Kubernetes vendors of the distribution or installer type offer more customization interfaces ‘as is’, but these vendors also expect more expertise from the customer [13].

In this paper, we assess feature incompatibilities between default cluster configurations of three leading vendors of the hosted product type that we consider type-representative: Azure Kubernetes Service (AKS) [14], AWS’ Elastic Kubernetes Service (EKS) [15] and Google Kubernetes Engine (GKE) [16]. However, the methodologies used for this study are not limited to AKS, EKS and GKE, but can be applied for every other Kubernetes vendor of the hosted product type. These commercially managed services have been chosen because they form the largest market share of the current cloud computing industry.

Moreover, we analyze the potential of Kubernetes to detect and configure compatible support for desired features across these vendors in a uniform manner. This analysis is inspired by the fact that existing state-of-the-art on model-driven migration of clusters across cloud providers [17] is very similar to the declarative configuration management approach of Kubernetes. Declarative configuration management approaches enforce a desired cluster state by a control loop that detects differences between desired and actual cluster state. Differences are resolved through automated reconfiguration [36].

Thus, we make three contributions. Firstly, we compare differences between vendors concerning cluster architecture and how the customization interfaces of Kubernetes are encapsulated by the vendors (as-is, proprietary, locked). This comparison allows us to distill a systematic overview of the ensuing feature incompatibilities.

Secondly, we analyze the potential of existing information assets of Kubernetes that are amendable for automated processing to allow for the detection of desired features by a control loop: (i) leveraging the aforementioned e2e testing suite of Kubernetes to test if a desired feature is functioning well, (ii) inspection of the visible configuration state of running clusters to determine that the feature is activated (iii) analysis of vendor documentation about whether the feature is supported. These three information assets are analyzed and mapped to an existing taxonomy of documented features of the open-source k8s distribution [21]. This mapping will show the potential of using these assets to detect compatible support for required features across the studied vendors.

Thirdly, we analyze the potential of defining a vendor-agnostic API for activating desired features across vendors in a uniform manner employing the control loop. This analysis also generates cloud migration guidance that helps

application managers or cluster administrators to avoid feature incompatibilities already in cloud-native application engineering processes.

This paper is structured as follows. Section II introduces the necessary background on Kubernetes and the above-mentioned approach for feature compatibility management. It also presents the methods used for the feature-based vendor comparison and analysis of the potential of the e2e testing suite. Then, Section III presents a detailed account of the findings for the three studied vendors, i.e., what are their feature incompatibilities, what is the feature coverage of the testing suite and other information assets and what reconfiguration actions are supported for activating a missing feature. Subsequently, Section IV summarizes these findings and further discusses the potential of Kubernetes for supporting feature compatibility management. Thereafter, Section V presents related work, and Section VI presents our conclusions.

## II. BACKGROUND AND METHODS

This section is structured as follows. Section A first presents the minimal amount of introduction to Kubernetes to understand the remainder of this Section. Thereafter, Section B sketches our previous work for transferring cluster configurations across cloud providers. Then Section C proposes an extension of this approach for automated feature compatibility management across Kubernetes vendors and identifies three concerns related to how to conduct the research. These three concerns are discussed in the remaining sections. Section D presents our previous work on a feature taxonomy for container orchestration platforms and refines it towards transferring Kubernetes clusters across cloud providers. Section E presents the information assets that are amendable for automated processing to detect support for the desired feature across a set of vendors. Section F presents the method used for quantifying to which extent the studied vendors can support a vendor-agnostic API for installing missing features across vendors in a uniform manner.

### A. KUBERNETES

Kubernetes is the de-facto standard for container-based cluster orchestrators. A cluster consists of a master control plane and worker nodes. It has been a pioneering platform that puts forward the declarative configuration management approach for management of container-based distributed applications. Declarative configuration management entails that a user-specified, desired system state is enforced upon an actual system by a control loop that detects differences between desired and actual state [18]. In Kubernetes, the desired state is specified and managed using a RESTful API and is composed of various types of resources such as containers, nodes and services. Different control loops exist for life cycle management of different kinds of resources.

The following components run within the master control plane: an API server for submitting and querying k8s resources, an etcd database for storing the resources, various

controller managers that each implement the control loop for particular types of resource and a scheduler for placing containers on worker nodes.

On every master and worker node runs the kube-proxy and the kubelet. The former constitutes the core load balancer of Kubernetes. The latter is a local agent for bootstrapping the nodes, for running the containers, provisioning their network endpoints and managing persistent storage volumes.

### B. TRANSFERRING CLUSTER CONFIGURATIONS

Throughout a project called CloudTRANSIT [20], we searched intensively for solutions to overcome “cloud lock-in”—the overall goal was to transfer cloud-native applications at runtime without downtime between different cloud providers. We analyzed commonalities of existing public and private cloud infrastructures via a review of industrial cloud standards and cloud applications and via a systematic mapping study of cloud-native application-related research [5]. It became clear that cloud-native applications share many common characteristics that can be exploited for transferability. As such, we compiled a reference model that plenty of cloud-native applications have in common [1]. As Fig. 2 shows, two basic operation modes of cloud-native applications are reasonable. One can deploy a highly-available k8s cluster within the same cloud provider, or one can deploy multiple k8s clusters across different federated cloud providers [11], [12] (multi-cloud, according to Fig. 2).

In both cases, the need arises to ensure that a particular application or cluster resource works the same across multiple cloud providers either when migrating or replicating these resources to another cloud provider. The existing state-of-the-art in cloud migration and multi-cloud resource orchestration has thoroughly validated the model-driven approach that allows for infrastructure-agnostic configuration management of container orchestration platforms across multiple cloud providers [17], [6], [101], [102].

In particular, our previous work [6] on a model-driven approach for transferring clusters across cloud providers is conceptually very similar to the declarative configuration management approach of Kubernetes and its control loop (cfr. Section II.A).

As shown in Fig. 3, the intended state of a Kubernetes platform, specified as a desired cluster configuration is enforced by the control loop, typically each time when a new cluster is added to the federation. Differences between intended and current state can be detected through monitoring the actual cluster configuration. The control loop then translates these differences into a set of reconfiguration actions that are supported by a vendor-agnostic API. This API is implemented by multiple drivers, one for each cloud provider.

### C. TOWARDS MANAGEMENT OF FEATURE COMPATIBILITY

The motivation for this work is the observation above that many optional or alternative features exist in the open-source k8s distribution, and many vendors, especially vendors of the hosted type, hide several of the existing customization

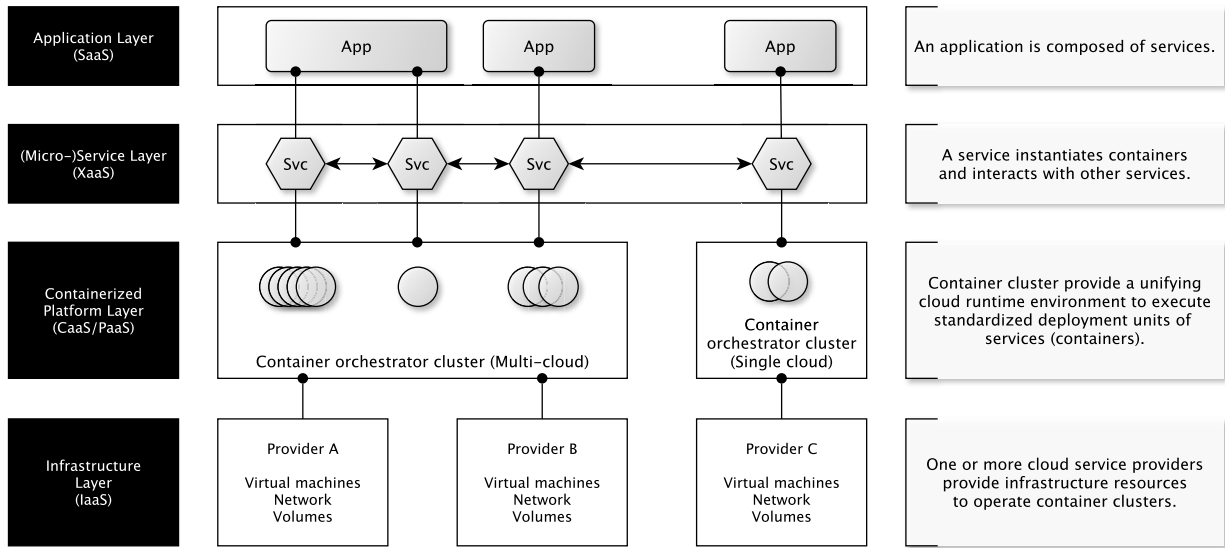


FIGURE 2. A reference architecture that many cloud-native applications have in common [1].

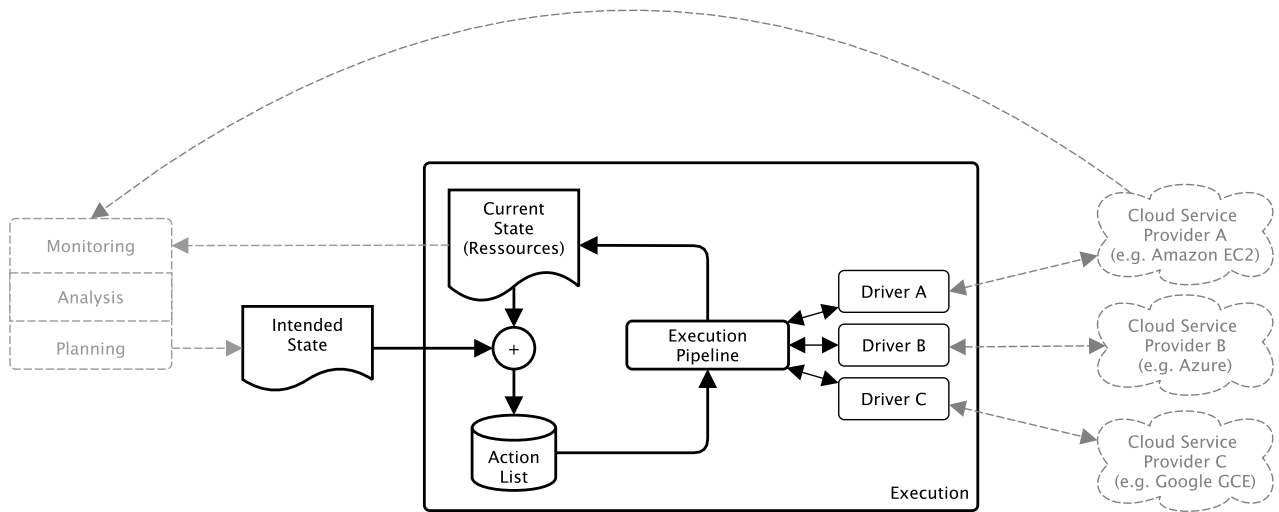


FIGURE 3. A vendor-agnostic control loop for enforcing a desired declarative state of the Kubernetes platform.

interfaces for activating these features and installing their dependent systems components.

Our vision is that the vendor-agnostic control loop (see Fig. 3) may also be used for ensuring that the same set features are consistently installed, and activated:

- The desired state of a cluster configuration could be augmented with a specification of desired features.
- Differences between intended and current state can be detected by relying on a feature detection module as part of the monitoring aspect of the control loop. Such a module can detect if the desired feature is supported by the default cluster configuration of a target vendor by extracting and processing information from the target vendor.

- If the desired feature is not supported according to this information, the control loop will execute a set of reconfiguration actions that are implemented by the vendor-agnostic API that can be supported by all vendors.

An overview of the design challenges for this extension to the declarative configuration management approach is outside the scope of this paper. But even in a more manual setting, application managers and cluster administrators would become more knowledgeable about what optional or alternative features of the open-source k8s distribution are supported by a default cluster configuration of a particular Kubernetes vendor. Moreover, having available an array of tactics for reconfiguring vendors with a desired feature in

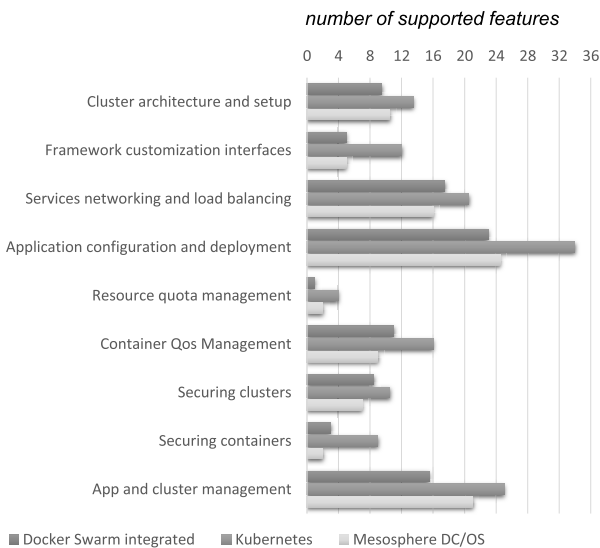
a uniform manner improves productivity and quality of the cloud-native application engineering process.

These considerations raise three concerns:

- 1) How to identify the external (or customer-oriented) features of the open-source k8s distribution?
- 2) How to map features to information assets that are amendable for automated processing to semi-automatically detect if a feature is supported by a k8s vendor?
- 3) How to quantify the ease-of-migration of existing reconfiguration actions for installing a particular feature in a target cluster and whether these actions can be supported through a vendor-agnostic API?

#### D. CONTAINER ORCHESTRATION FEATURE TAXONOMY

For the first concern, we have already conducted a comprehensive study of the documentation of existing open-source container orchestration platforms, including Docker Swarm, Mesos and Kubernetes [19]. In this study, we have distinguished between common features (shared by at least two orchestration platforms) and 54 unique features. We have organized these features into nine functional aspects that are presented in Fig. 4.



**FIGURE 4. Nine functional aspects of container orchestration platforms. The number of common and unique features supported by Docker Swarm, Kubernetes and Mesos is shown for each functional aspect. When a common feature is partially supported, it is counted as 0.5 [19].**

For Kubernetes v1.11, released in July 2018, we identified in total 148 features [19]. For Kubernetes v1.13 – the default k8s release of the major cloud providers at the time of our research – we identified 162 features in total. The small number of 14 additional features is in line with the emerging trend that the number of feature additions has significantly decreased in all mainstream orchestration platforms [19]. This trend can be explained by the fact that k8s already appeared as a de-facto standard [21]–[23] and

the open-source development efforts refocused on improving the security, performance and robustness of existing k8s features [24]–[26].

The existing feature taxonomy also includes the framework customization aspect that identified 12 different types of customization interfaces (cfr. Fig. 4).

We distinguish between three types of customization interfaces in Kubernetes:

- feature gates, which are toggles for (de)activating the alpha and beta features of a particular k8s release
- admission controllers that validate and mutate API requests to enforce features
- extension APIs that allow extending Kubernetes with various functionalities such as networking plugins, volume plugins, IAM plugins, container runtimes and new REST APIs.

Additionally, we also account the separate instantiations of feature gates and admission controllers as independent features in their own right. This point of view enables a finer-grained level of configurability that Kubernetes vendors can decide to expose in their customization interfaces.

Kubernetes release v1.13 itself offers 66 feature gates and 28 admission controllers resulting in a total set of k8s 256 features (162 external k8s feature, 66 feature gates and 28 admission controllers).

#### E. MAPPING FEATURES TO INFORMATION ASSETS

To address the second concern, we will consult three complementary information assets that are amendable for automated processing. In particular, we have considered the following assets in order of their ease for automated processing: (1) end-to-end testing of features by running Kubernetes’ end-to-end testing suite, (2) inspection of the configuration of running k8s clusters, (3) study and analysis of the vendor documentation.

##### 1) END-TO-END TESTING SUITE

The test modules of the e2e testing suite are organized in different packages according to the k8s SIGs. Test modules are also labelled with a String-based descriptor that consists of multiple labels. It can be selected at run-time, which test modules to execute, employing a regular expression over these labels [27]. A test module itself consists of a dozen of tests. Each is labelled with two other String-based descriptors that specify what the test checks and how the test performs the check.

These String-based descriptors make it easy to mine the source code of the e2e testing suite for relevant tests on a per-feature basis by using utilities such as grep. For k8s release 1.13, we were able to map 1730 out of the 1940 tests of the entire e2e testing suite to a part of the documented k8 features, feature gates and admission controllers. Note that, on average, just 1 out of 10 tests of the e2e testing suite has been labelled as a CNCF conformance test.

Each test is also labelled with a list of the k8s vendors supported by that test. As such, we could easily find all

tests that are supported by a particular subset of k8s vendors. The e2e testing suite will only run vendor-specific tests if a specific flag and associated configuration parameters for that vendor are set [10].

## 2) CONFIGURATION STATE

Secondly, we also studied the configuration state of running k8s clusters of specific vendors to establish configuration proof for particular features. The total number of features with configuration proof that can be found in this way depends on the k8s vendor itself. This analysis can be somewhat limited for hosted k8s products when these offer no visibility into the master control plane of the cluster.

## 3) VENDOR DOCUMENTATION

Thirdly, we relied on vendor-specific documentation to determine what features are installed by default and which features could be activated via a customization interface.

## 4) AUTOMATED PROCESSING OF TEST RESULTS

To demonstrate the feasibility of automated processing of test results, we have created a thin layer of automation to process e2e test results. Firstly we aggregated into a CSV file the mappings from features to zero or more relevant regular e2e test expressions. Then, given a subset of k8s vendors, each row of the CSV file contains a tuple of documentation states for each vendor (the feature is supported, not supported or optional/alternative) and a tuple of found configuration proof for each vendor. Secondly, this CSV file is processed by a parser that generates for each e2e test expression the total number of tests and the number of failed and succeeded tests for each vendor. All e2e tests are run once beforehand in multiple batches where each batch corresponds with the tests of one k8s SIG packages. This grouping into batches is to prevent interferences between SIG packages due to residual effects in the cluster state.

## F. QUANTIFYING EASE-OF-MIGRATION

With respect to the third concern, we distinguish between different levels of ease and automation with which a k8s feature can be consistently activated when transferring a cluster from a source vendor to a target vendor:

- At the highest level of ease, *automated migration* appears for a particular k8s feature when the k8s feature is already supported by the target vendor or the native customization interfaces of Kubernetes can be used to activate the k8s feature.
- At the medium level, *uniform reconfiguration* of the target vendor is possible by executing a sequence of generic reconfiguration primitives that are offered by the vendor-agnostic API.
- *Custom translation of input or output data* is needed when the target vendor does not support activating the open-source k8s feature. Instead, the vendor offers an alternative feature implementation that is not interoperable with the source vendor's implementation

(e.g. different data formats for logging, monitoring, auditing).

- *Migration is not possible* because the target vendor does not provide support for activating the relevant k8s feature, and there is no alternative feature implementation.

## III. VENDOR COMPARISON AND ANALYSIS

For this article, we have compared the three leading vendors of the hosted type: Azure Kubernetes Service (AKS), the AWS Elastic Container Service for Kubernetes (EKS) and Google Kubernetes Engine (GKE). In the following, we investigate the following five questions:

- 1) What are the differences between vendors in terms of cluster architecture?
- 2) What are the differences between vendors in terms of their approach towards encapsulating the customization interfaces of Kubernetes?
- 3) Which feature incompatibilities between similar default cluster configurations of the vendors ensue from the above differences?
- 4) What is the completeness of the e2e testing suite and visible cluster configuration state in terms of feature coverage? What is the completeness and accuracy of vendor documentation?
- 5) What are generic reconfiguration actions for installing missing system components for a missing feature and activating it uniformly?

Tables 1 to 3 summarize the main findings for these questions for k8s release v1.13 – the default supported release by all studied vendors at the time of our research.

Concerning the 1<sup>st</sup> question (see Table 1), it is observed that AKS does not support a highly-available master plane that is replicated across multiple availability zones. Moreover, GKE offers the most scalable platform.

Concerning the 2<sup>nd</sup> question (see Table 2), EKS locks the fewest customization interfaces and should therefore allow to activate or deactivate a larger subset of optional and alternative features than AKS and GKE.

For the 3<sup>rd</sup> question (see Table 3), we found significant differences between at least two vendors for 30 out of 162 k8s features, 28 out of 66 feature gates, and 7 out of 28 admission controllers (see 1<sup>st</sup> and 2<sup>nd</sup> row of Table 3).

With respect to 4<sup>th</sup> question (see Table 3), matching e2e tests exist only for 64.1% of the 162 k8s features (see 3<sup>rd</sup> row and 1<sup>st</sup> column of Table 3). Worse, 17.8% of the features have matching tests that are difficult to configure or are not supported for all studied vendors. As such, we were able to produce valid test results for only 46.3% of the documented k8s features (see 4<sup>th</sup> row of Table 3). Available configuration state was even more superficially available because all studied vendors hide the master control plane configuration (see 5<sup>th</sup> row of Table 3). Vendor documentation is the most comprehensive source of information (see 6<sup>th</sup> row). Finally, e2e test results revealed 12 errors in vendors or at least these

**TABLE 1. differences concerning the cluster architecture.**

Features	AKS	EKS	GKE
Declarative configuration management	✓	✓	✓
Versioned API	✓	✓	✓
Simple policy-based scheduler	✓	✓	✓
Master plane HA [28]	n/a	Multi-AZ	Multi-AZ
Node restarts [28]	✓	✓	✓
Maximum pods per worker node	Default 110, user-configurable	Auto-configured range of [4,737] depending on the EC2 instance, user-configurable via <code>eksctl</code> [29]	Default 110, user-configurable
Average observed cluster creation time for 3-node cluster	5 minutes (using console shell)	8 minutes (using <code>eksctl</code> )	3 minutes (using the console)
Maximum nodes per cluster	100 [30]	3000 [31]	5000 [32]
Infrastructure-as-code support	Powershell templates [33]	ClusterConfig file in <code>eksctl</code> [29]	
Common installation methods	Dockerized CO software, Kubelet native Linux, CLI and GUI for cluster setup, Microsoft Windows or Windows Server		

The e2e test suite validates 3 out of 15 features for all vendors (cfr. the dark shaded features). Matching tests exist for two other features, but these tests are not developed well for all features (cfr. the light shaded features), Configuration proof validates 2 other features (cfr. the medium shaded features). Vendor documentation validates all other features.

results were inconsistent with documentation or configuration proof (see 7th row).

Finally, with respect to the 5th question, the following relevant reconfiguration primitives for a vendor-agnostic API can be distilled:

- wrap commonly offered (proprietary) customization interfaces
- install add-ons using `kubectl` or the vendor CLI,
- bootstrap worker nodes with a custom VM image, a cloud-init script or a privileged `DaemonSet`,
- apply the Operator pattern [71] to re-introduce missing features through CRDs.

In the following, we present a more detailed account of these five questions.

### A. CLUSTER ARCHITECTURE AND SETUP

All three studied vendors adopt the declarative configuration management approach of Kubernetes that has been introduced in Section II.A. As a reminder, the following components run within the master control plane: an API server for submitting and querying k8s resources, an `etcd` database for storing the resources, various controller managers that each implement the control loop for particular types of resource and a scheduler for placing containers on worker nodes.

**TABLE 2. Differences between vendors with respect to their approach of encapsulating the customization interfaces of Kubernetes.**

Customization interfaces	Part of the Master REST API	AKS	EKS	GKE
Feature gates	No	Locked	Proprietary for worker nodes	Locked
Admission controllers	No	Locked	Locked	Locked
Scheduler plugins	Yes	As-is	As-is	As-is
Networking plugin architecture	No	Proprietary	Proprietary	Proprietary
Container runtime interface	No	Locked	Proprietary	Locked
Storage volume plugins	Yes	As-is	As-is	As-is
IAM modules	No	Locked	Locked	Locked
Annotations	Yes	As-is	As-is	As-is
CRD	Yes	As-is	As-is	As-is
Aggregation of new APIs	Yes	As-is	As-is	As-is
Management of extended node resources	Partially (Device Plugins)	Partially Proprietary (device libraries)	Partially Proprietary (device libraries)	Partially Proprietary (device libraries)
CloudController Manager plugin	No	Locked	Locked	Locked
Kubelet Configuration API	Partially (no control loop)	Locked	Proprietary	Locked

**TABLE 3. Feature coverage of the e2e tests, configuration state and vendor documentation.**

	Core k8 features and features of extension plugins	Feature gates	Admission controllers	Total
#features of k8s release v1.13	162	66	28	256
#features with significant incompatibilities between a pair of vendors	30	28	7	67
#features with matching e2e tests	104	33	17	154
# features with valid e2e tests for studied vendors (AKS, EKS, GKE)	75	16	13	104
#features with configuration proof for studied vendors	24	66	2	92
#features with no vendor-specific documentation available	5	0	0	5
#features with inconsistencies between e2e test results and documentation or configuration state	6	2	4	12

For extensibility, the REST API is hierarchically divided into different API groups that can be versioned within the `alpha/beta/stable` stages [34].

For all studied vendors, the master control plane is out of direct control of the user. However, it is possible to query the API server with the currently activated API groups. It is also

possible to inspect the logs of the master control plane in all vendors.

This analysis shows that all vendors are by default configured with the same API groups but only from the beta and stable stage. Only GKE includes in its cluster catalogue the so-called alpha cluster where several API-groups from the alpha stage are additionally included. However, GKE does not commit to an SLA for these type of clusters.

Note that configuring the API server with an additional API group does not guarantee that this API group will work properly. After all, many API groups require extra bolts and nuts of Kubernetes to be appropriately configured as well (see Section III.B).

Worker nodes can be accessed via SSH in all vendors allowing us to inspect the configuration of the Kubernetes platform at these worker nodes. All vendors install the k8s platform similarly: the kubelet agent runs as a regular Linux process, whereas all other k8s components run as Kubernetes Pods (see Section III.C). There is also support for running Kubernetes clusters on Windows Server nodes in all vendors.

Although the basic architectural approach and installation methods share the common core of Kubernetes, architectural properties related to scalability and availability do differ considerably among the studied vendors (see Table 1) [28]. These differences are presumably due to their underlying IaaS infrastructure.

## B. FRAMEWORK CUSTOMIZATION INTERFACES

The ease in migrating Kubernetes resources between clusters depends on the way these clusters and their underlying k8s platforms are configured and whether these configurations can be ported. Kubernetes can be configured using the following customization interfaces [35]–[37]:

- feature gates
- admission controllers
- scheduler plugins
- networking plugins
- plugin-architecture for other container runtimes
- storage volume plugins
- Identity and Access Management (IAM) modules
- annotations
- extending the main k8s API with custom resource definitions (CRDs)
- aggregation of new APIs
- management of extended node resources such as GPU
- CloudControllerManager plugin
- the KubeletConfiguration API

### 1) FEATURE GATES

In the open-source k8s distribution, alpha and beta features are by default deactivated and activated, respectively. This default setting can be changed by setting specific feature gates to true or false. Also, alpha features may be promoted to the beta stage in later releases of the open-source k8s distribution and beta feature gates may be removed as they promote to stable features [38].

The available configuration proof for these feature gates shows that the studied vendors differ slightly concerning the adjustments of this default setting (see Table 4).

**TABLE 4. Adjustments to the default feature gate settings of k8s.**

	Activated alpha features (v1.13)	Deactivated beta features (v1.13)
AKS	<i>RotateKubeletServerCertificate (already Beta feature in 1.13)</i>	
	<i>PodPriority (already Beta feature in 1.13)</i>	
Initializers		
EKS	<i>RotateKubeletServerCertificate (already Beta feature in 1.13)</i>	
	Feature gates for the kubelet agent [39] can be set using <code>eksctl</code> [40]	
GKE	ExperimentalCriticalPodAnnotation (deprecated in 1.13)	TaintBasedEvictions DynamicKubeletConfig
	<i>RotateKubeletServerCertificate (already Beta feature in 1.13)</i>	<i>NodeLease (still alpha feature in 1.13)</i>

. Feature gate settings in italic font do not differ from k8s v1.13 defaults.

Using the KubeletConfiguration API of EKS, cluster administrators can make further adjustments to those feature gates that affect the functionality of worker nodes [40].

In GKE alpha clusters, all alpha features for master and worker nodes are activated by default, but it is not possible to further customize on a per feature-basis.

However, alpha features correspond with immature functionality. Eventually, they either promote to the beta stage, or they are removed in a subsequent Kubernetes release. It is, therefore, better to select a later Kubernetes release where the desired alpha feature has become beta and thus is activated by default. For these reasons, we ignore alpha feature gates in the remainder of this article.

### 2) ADMISSION CONTROLLERS

An admission controller is a modular piece of code that intercepts requests to the master API server. Multiple admission controllers can be run in sequence [41]. After an API request has been successfully authenticated and authorized, admission controllers either accept, reject or mutate the request. They can also update the state of other stored k8s API objects. Admission controllers are used for implementing various functionalities such as resource quota management and enforcement of container security isolation policies (cfr. Section III.C). Unfortunately, different vendors enable a different set of admission controllers. All vendors also differ from the default settings of the open-source k8s distribution (see Table 5). Although cluster administrators cannot change these settings, there exists a run-time pluggable admission



controller, named Webhook, which can be used in all vendors to inject custom code into a running cluster [19].

Some e2e test results for the k8s release v1.13 are inconsistent with the existing documentation of the vendors (see Table 5). So, parts of the current vendor documentation [42]–[44] is outdated or recent additions are insufficiently tested.

**TABLE 5. An overview of which admission controllers are enabled by default in the open-source distribution of k8s and the three studied vendors.**

Non-deprecated admission controllers in Kubernetes release 1.13	Open-source k8s [45]	Azure AKS [42]	AWS EKS [43]	Google GKE [44]
NamespaceLifecycle	✓	✓	✓	✓
LimitRanger	✓	✓	✓	✓
ServiceAccount	✓	✓	✓	✓
DefaultStorageClass	✓	✓	✓	✓
DefaultTolerationSeconds	✓	✓	✓	✓
MutatingAdmissionWebhook	✓	✓	✓	✓
ValidatingAdmissionWebhook	✓	✓	✓	✓
ResourceQuota	✓	✓	✓	✓
Priority	✓	✓	✓	✓
PersistentVolumeClaimResize	✓			✓
Initializers	Opt	✓		
NodeRestriction	Opt		✓	✓
AlwaysPullImages	Opt.	✓		
ExtendedResourceToleration	Opt.			✓
StorageObjectInUseProtection	Opt.	✓	✓	✓
PodPreset	Opt.			✓
PodSecurityPolicy	Opt.	Opt.	✗	Opt.
EventRateLimit	Opt.			
ImagePolicyWebHook	Opt.			
LimitPodHardAntiAffinityTopology	Opt.			
NamespaceAutoProvision	Opt.			
NamespaceExists	Opt.			
PodNodeSelector	Opt.			
PodTolerationRestriction	Opt			
RuntimeClass	Opt			
TaintNodesByCondition	Opt.			
SecurityContextDeny	Opt.	✓	✓	✓
OwnerReferencesPermissionEnforcement	Opt.			

The e2e test suite validates 13 out of 28 controllers for all vendors (cfr. the dark shaded controllers). Matching tests exist for 4 other controllers, but these tests are skipped because of several reasons (see the light shaded controllers, cfr. Table VIII). Configuration proof validates 1 additional controller (cfr the medium shaded controllers). Finally, there are several inconsistencies between vendor documentation and e2e test results. Cell legend: ✓ =enabled by default; Opt. =can be optionally enabled, ✗=outdated vendor documentation, ✗ = the admission controller is not correctly configured

### 3) SCHEDULER PLUGINS

The default scheduler of the open-source k8s distribution can be replaced, or multiple schedulers can run at the same

time [46] by setting the schedulerName field of the Pod REST resource. All vendors support this feature by default because the feature is part of the core API group of k8s. No e2e tests exist that validate a custom scheduler implementation.

### 4) NETWORKING PLUGINS

Two different types of network plugin architectures – kubenet and CNI – exist [47]. Although these two plugin architectures have been incepted at the beginning of the open-source Kubernetes project, they are still labelled with the alpha stage:

- CNI is a specification for a network plugin architecture that is jointly developed by multiple companies. It allows for multiple network plugins to be installed simultaneously and inter-composed together. Some CNI plugins also allow attaching multiple network interface to a single Pod [19].
- Kubenet has been developed in-tree within the Kubernetes open-source project to support well-performing container networks on top of cloud providers.

Different vendors employ both types of plugins for different cluster configurations (see Table 6). Standard cluster configurations tend to run on kubenet while clusters with more advanced features rely on CNI. Moreover, the CNI-based plugin Calico is the leading plugin for network policies. However, this latter feature does not function appropriately in AKS (see Section III.C).

**TABLE 6. A mapping between cluster configurations of vendors and type of network plugins.**

High-level k8s platform variants	AKS	EKS	GKE
Standard cluster	kubenet	CNI [48]	kubenet
Cluster auto-scaling	CNI [49]	CNI [48]	Kubenet
NetworkPolicy support	Calico [50]	Calico [51]	Calico [52]
Alpha cluster with all alpha features and APIs activated	n/a	n/a	CNI [53]
Multiple network interfaces per container	n/a	n/a	n/a

The e2e test suite validates only the NetworkPolicy feature for all vendors (cfr. the dark shaded feature) and configuration proof validates kubenet or CNI for all vendors (cfr the medium shaded features).

### 5) PLUGIN ARCHITECTURE FOR CONTAINER RUNTIMES

Kubernetes supports the Container Runtime Interface (CRI) for pluggable container runtimes. It also supports the OCI standard, provided that cri-o is installed [54]. Therefore, in theory, it should be possible to use a wide range of container runtime implementations in any certified k8s vendor. However, it is only possible in EKS to boot worker nodes from customized VM images that have been amended with an installation of cri-o [55].

GKE boots worker nodes by default from a VM image with a container-optimized OS that supports the containerd engine [56] next to Docker. In EKS, as already noted

above, it is possible to boot worker nodes from customized VM images that support containerd [57].

No e2e tests or configuration proof validates the above.

### 6) VOLUME PLUGINS

Kubernetes supports a wide range of persistent volume drivers and an API for dynamically provisioning volumes using storage classes and persistent volume claims [58]. Kubernetes also supports dynamic installation of new types of volume plugins using the Container Storage Interface (CSI) specification [59].

All studied vendors offer a very similar set of standard Kubernetes volumes. Commonly supported volume drivers are local volumes and external persistent volumes that rely on vendor-specific storage services. All vendors also support dynamic provisioning of volumes. Finally, at least one operational CSI-based volume driver exists in each vendor [60].

Note that all vendors support many operational management features according to the default configured admission controllers (see Table 5): (i) prevention of deletion of persistent volumes that are still in use by Pods, (ii) dynamic resizing of existing, unattached volumes [61] and (iii) automated capacity management of a node in terms of the maximum number of simultaneously attached volumes [62].

There is one incompatibility, though, namely for the sub-Path feature [63]. AKS and GKE currently support this feature for a substantially larger number of “in-tree” developed volume drivers (e.g. iscsi, rdb, ceph, nfs) than EKS.

Cluster administrators can also install other storage solutions themselves in the cluster if these solutions have a CSI-based or in-tree developed volume driver. For example, the e2e tests for installing an NFS server and attaching NFS volumes to containers all succeeded for all studied vendors.

### 7) IAM MODULES

Kubernetes offers a wide range of modules for authentication of human users, worker nodes and non-human Pods. Inspection of the vendor documentation [64]–[66] and configuration state demonstrates that all studied vendors all rely on X509 certificates, bearer tokens and service account tokens for respectively human users, worker nodes and Pods. Cluster administrators cannot use other existing in-tree developed authentication modules of the open-source k8s distribution (see Table 7).

In opposition to service accounts for Pods, there is no Kubernetes API for representing human users and bearer tokens. The in-tree developed authentication modules simply attempt to associate specific attributes, such as UserName, UID and Groups to every HTTP request to the Kubernetes API.

Kubernetes allows specifying access control policies based on these attributes using role-based access control (RBAC) [67] or attribute-based access control (ABAC) [68]. While all vendors support RBAC, ABAC is only supported by GKE. The e2e tests further show that node authorization

**TABLE 7. Compatibility of vendors with in-tree developed AAA features of Kubernetes.**

Type of IAM feature	IAM features of the k8s distribution	AKS	EKS	GKE
Authent. to master	X509 cert. service account	✓	✓	✓
	bearer tokens	✓	✓	✓
	no anonymous	✓	✓	✓
Authent. to kubelet at port 10250	RBAC	✓	✓	✓
	ABAC			✓
Author. to master	worker node authorization	✗	✓	✓
	Webhook	✓	✓	✓
Author. to kubelet	no AlwaysAllow	✓	✓	✓
		Opt.	Opt.	✓
Audit		✓	✓	✓
Certificate API for application workloads		✓	✓	✓

The e2e test suite validates 2 out of 13 features for all vendors (cfr. the dark shaded features) and configuration proof validates 5 additional features for all vendors (cfr the medium shaded features). Matching tests exist for 3 other features, but these tests are not developed for the studied vendors (see the light shaded features). Vendor documentation validates all other features. Cell legend: ✓ =enabled by default; ✗ = the feature is not correctly configured, Opt=Optional.

(access control of requests to the master API from worker nodes) does not function correctly in AKS.

Kubernetes also supports authentication and authorization of requests to the kubelet API [69]. According to the configuration state of the worker nodes, all vendors support it, although AKS only supports X509 certificates.

Finally, Kubernetes offers stable support for non-repudiation by means of audit [70] and all vendors support it.

### 8) ANNOTATIONS AND CUSTOM RESOURCE DEFINITIONS

Annotations allow attaching additional information to existing API resources, whereas Custom Resource Definitions (CRDs) allow extending the Kubernetes API with new types of resources. CRDs are used for adding new functionality to running Kubernetes clusters and offer higher-level abstractions for managing popular Kubernetes applications by means of the Operator pattern [71], which is a user-defined control loop for managing the life cycle of the CRDs.

Each studied vendor allows customers to extend the API with their own annotations and CRDs. The e2e test suite validates support for these two features for all vendors.

### 9) API AGGREGATION

Kubernetes allows also adding new APIs to orthogonally extend the Kubernetes API of running clusters with new functionality. For example, Istio, a popular service management layer, relies on the API aggregation feature. Each vendor supports this feature. Matching tests exist for this feature, but these tests have not been developed for the studied vendors.

## 10) MANAGEMENT OF EXTENDED NODE RESOURCES

Management of GPU resources is an optional feature in all studied vendors. To install the feature, worker nodes need to be booted from prefabricated VM images with GPU support, and the GPU device plugin needs to be installed on every node using the DaemonSet API of Kubernetes.

It is therefore also possible to install any type of device plugin, provided that nodes can be bootstrapped from custom VM images with the appropriate software installed. As already noted above, it is only possible in EKS to bootstrap worker nodes from custom VM images. However, it is also possible to bootstrap worker nodes with missing software using a DaemonSet that has privileges for modifying the host OS [72], [73]. Alternatively, EKS [74] and GKE [75] also offer support for the cloud-init standard for bootstrapping the worker nodes with the appropriate software. However, the use of a single cloud-init script is more trustworthy than installing software with a privileged DaemonSet; the cluster security configuration must be temporarily modified to allow for privileged DaemonSets, but this leaves the cluster more vulnerable to malicious attacks.

Matching e2e tests exist for the DevicePlugin feature, but the tests produced false negatives because these tests could not be correctly configured using the sonobuoy tool.

## 11) CLOUD CONTROLLER MANAGER PLUGIN

Besides the core controller managers, the cloud controller manager allows k8s vendors to implement additional control loops for managing the cloud infrastructure that is utilized or owned by the vendors. Examples of these cloud-specific control loops include provisioning of new nodes, creating a persistent volume and attaching it to a node, configuring in-bound and out-bound routes such as the cloud-provisioned load-balancing service and the back-end.

A legacy “in-tree” cloud provider package of the open-source k8s distribution has been replaced by an “out-of-tree” approach where the code of the cloud-specific controller managers can evolve independently from the k8s core [76]. However, an inspection of the configuration state of the running platforms shows that all vendors still use the “in-tree” code for k8s release v1.13.

## 12) KUBELET CONFIGURATION API

As stated in Section II.A, the kubelet is the local agent of Kubernetes on every node of the cluster. It is responsible for integrating the container runtime and networking plugins into a coherent fashion for a particular combination of activated feature gates. Moreover, it also configures the container runtime to enforce resource isolation for CPU, memory, ephemeral storage, as well as isolation of file system and network isolation between co-located containers.

The kubelet command in the reference manual of Kubernetes has gradually evolved from accepting a long list of parameters to taking a single configuration file that contains a large part of these options [77].

Such a configuration file enables infrastructure-as-code practices [78] and is therefore recommended in any development context where feature compatibility between different Kubernetes clusters must be managed.

Since Kubernetes release 1.11, this file can also be managed on a per-node basis, using the KubeletConfiguration API that belongs to the core API group. If a vendor offers (proprietary) access to this API, this can be regarded as a feature on itself.

EKS allows setting some fields of the KubeletConfiguration API by setting the kubeletExtraConfig field of EC2 node groups [40]. These parameters include feature gates that only affect worker nodes, resources reserved for the Kubernetes platform, and CPU management policies.

For the other vendors, the aforementioned privileged DaemonSet can update kubelet configuration parameters, but this requires restarting the kubelet. Unfortunately, dynamic kubelet reconfiguration is not activated in the initial kubelet configuration of the vendors and therefore restarting the kubelet will affect all running containers on the node. As a result, each node needs to be drained before restarting the kubelet, which is a costly operation for clusters with a large number of nodes. We refer to Section III.C.6 for more information about dynamic kubelet reconfiguration.

## C. FEATURE INCOMPATIBILITIES

We explain the found feature incompatibilities for the other seven functional aspects of the feature taxonomy. A large part of the found feature incompatibilities emerge from the cluster architecture and customization interfaces of the vendor, but not all of them. For example, some core k8s features do not work out-of-the-box but require a manual reconfiguration of the underlying cloud infrastructure.

### 1) APPLICATION CONFIGURATION AND DEPLOYMENT.

Pods are the unit of application deployment in Kubernetes. Different API resources exist for configuring and deploying Pods in different types of workload configurations: ReplicaSet, StatefulSet and DaemonSet respectively manage web application tiers, master-slave databases and daemons on every node of the cluster. All these workload configurations can be auto-scaled in a generic manner by the Horizontal or Vertical Pod Autoscaler APIs.

Secondly, Kubernetes also supports the reuse of modular Pod configuration fragments by means of PodPreset and ConfigMap resources.

Thirdly, there is support for different types of (rolling) update strategies for Pods such as rolling upgrades and canary testing, and (non)-disruptive Pod updates [19].

Most of these features are commonly supported by all three vendors. There are, however, a few differences. Firstly, the Horizontal Pod Autoscaler API is only functioning well out-of-the-box in AKS and GKE but not in EKS due to a missing kube-system Pod called metrics-server. However, cluster administrators can install this missing metrics-server themselves [79]. Secondly, the Vertical Pod Autoscaler add-on is

only supported in GKE [28]. Thirdly, alpha clusters of GKE support the PodPresets API, which allows injecting a single piece of configuration across multiple Pods.

## 2) SERVICES NETWORKING AND LOAD BALANCING

Kubernetes supports various approaches for exposing the services of Pods through a stable network address that survives failures and migration of Pods [80]. Key-value labels attached to Pods enable application managers to select the subset of Pods that must be exposed by means of the same network address.

Kubernetes supports different types of stable network addresses. Firstly, it supports exposing replicated Pods by means of a stable ClusterIP address, NodePort or cloud-provided external LoadBalancer. The kube-proxy, a Layer 4 load balancer, runs on every node of the cluster to serve all ClusterIP and NodePort services.

Secondly, Headless services bypass the kube-proxy and expose each replicated Pod as a stable DNS name that is registered in the internal DNS server of the cluster. This configuration is typically required for StatefulSets where each replicated Pod must be separately addressable (cfr. subsection Application configuration and deployment).

Thirdly, Ingress resources additionally declare HTTPS load balancer rules for the services that are exposed via a Cluster IP. Fourthly, services of type ExternalName declare a DNS name for a depended service that does not run inside the k8s cluster.

Generally, all studied vendors support these three types of services with a few exceptions:

- The e2e tests show that when creating a NodePort service, only in AKS the SecurityGroup of the nodes is automatically adjusted to allow incoming traffic on that node port [81].
- Moreover, none of the vendors permits cluster administrators to customize the allowed range of node ports.
- The e2e tests further show that ExternalName services do not function properly in EKS.
- Configuration state shows that none of the vendors configures the built-in L4 load balancer (i.e. kube-proxy) with ipvs, a fast Linux kernel feature for load balancing, but instead uses iptables. As noted in Section III.B.10, it is possible to bootstrap worker nodes with Linux kernel libraries for ipvs using a cloud-init script or a privileged DaemonSet [72]–[75].
- Ingress resources for Layer 7 Load balancing are by default, supported in AKS [82] and GKE [83]. For EKS, there exists an optional community-driven HTTPS load balancer [84].

## 3) RESOURCE QUOTA AND CONTAINER QoS MANAGEMENT

All studied vendors support the full feature set of resource quota management of the open-source k8s distribution. All the QoS management features of the open-source k8s distribution are also supported by all studied vendors,

except for the CPU management feature [85]. This feature, which is disabled in all vendors, allows to exclusively reserve CPU cores for containers of the highest QoS class.

However, in EKS it is possible to activate this feature by setting the appropriate CPU-management policy in the KubeletConfiguration API (cfr. Section III.B.12)

We defer the reader to [19], [86] for an overview of resource quota management and container QoS management features of Kubernetes.

## 4) SECURING CLUSTERS

As already discussed in Section III.B.6, all studied vendors use a compatible subset of the authentication modules, but the concrete mechanisms for setting up credentials are tied into their underlying IAM cloud service.

With respect to cluster network security, all vendors allow cluster administrators to activate the network policies feature by relying on the Calico network plugin (see Table 6). A network policy is similar to the notion of a security group in IaaS to constrain in-bound and out-bound network connections between Pods [19]. The e2e tests show however that this feature does not function well in AKS.

All vendors also add some security features such as dividing clusters into private and public networks. Additionally, GKE supports by default encryption of control messages between master and worker nodes [87], while EKS offers this as an additional feature sold on the AWS market place [88]. Encryption of application-level messages is also supported by default in GKE only [89].

## 5) SECURING CONTAINERS

All vendors support improved security isolation of containers using SecurityContexts. Additional verification and enforcement of particular SecurityContexts upon Pods through the PodSecurityPolicy concept is also supported by all vendors, albeit in different ways: in AKS and GKE the feature can be optionally activated, while in EKS the feature is installed by default. However, the e2e tests for PodSecurityPolicies partially failed for EKS. Further, the SecurityContext tests showed that EKS and GKE only correctly support the SELinux access control model, and the runAsGroup primitive does not function properly on any of the studied vendors. As mentioned before, it is possible to install these missing features by bootstrapping worker nodes with missing appropriate Linux packages using a privileged DaemonSet or cloud-init script.

## 6) APPLICATIONS AND CLUSTER MANAGEMENT

This aspect consists of many different sub-aspects (see Table 8). Concerning management tools, the Kubernetes dashboard of the open-source distribution is supported by all vendors, but it is not enabled by default because of security reasons [90].

With respect to central monitoring of container resource usage, Kubernetes distinguishes between the aforementioned core metrics server for auto-scaling of Pods and a full metrics

**TABLE 8. An overview of the feature coverage of three information assets for all aspects of the feature taxonomy.**

Functional aspect and sub-aspects	# k8s features, feature gates and admission controllers	# features with matching tests					# Features with configuration state	# Features where vendor docs and state are consistent with e2e tests	Accumulat. coverage			
		Valid test results	False negatives	Skipped tests		% Features with reliable tests			Additional % of features with vendor documentation	Additional % features with configuration proof	Additional % features with reliable tests	
				Not developed for all vendors	Improper configuration or bug							Added after k8s release 1.13
Cluster archit./setup	16	2	1	0	1	0	5	16	13	25	100	
Framework customization interfaces	Feature gates	66	16	3	5	3	7	66	64	24	100	100
	Admiss. controllers	28	13	0	1	3	1	2	24	46	54	100
	CRI	2	0	0	0	0	0	0	2	0	0	100
	Scheduler plugins	1	0	0	0	0	0	0	1	0	0	100
	Network. Plugins	4	1	0	0	0	0	3	3	25	75	100
	Volume plugins	15	11	0	3	1	0	0	14	73	73	93
	IAM modules	14	2	4	0	3	0	6	13	14	57	100
	Annotations	1	1	0	0	0	0	0	1	100	100	100
	CRDs	1	1	0	0	0	0	0	1	100	100	100
	API aggregation	1	0	0	0	1	0	0	1	0	0	100
	Extended node resources	2	0	1	0	0	0	0	2	0	50	100
	CloudController API	1	0	0	0	0	0	1	1	0	100	100
	KubeletConfiguration API	1	0	0	0	0	0	1	1	0	100	100
App conf.	Workload config.	11	8	1	0	0	0	11	73	73	100	
	Reusable container configuration	6	5	0	0	1	0	0	6	83	83	100
	(Rolling) updates of Pods	6	6	0	0	0	0	0	6	100	100	100
Services networking	16	7	0	1	0	1	3	16	44	63	100	
Resource quota management	4	4	0	0	0	0	0	4	100	100	100	
Container QoS management	12	5	0	0	1	0	0	12	42	42	100	
Cluster network security	4	1	0	0	0	0	1	3	25	25	100	
Securing containers	11	10	0	0	1	0	0	10	91	91	100	
App and cluster management	Client-side management	3	3	1	0	0	0	0	3	100	100	100
	Monitoring resource usage	6	3	0	0	3	0	1	6	50	50	100
	Health checks and events	2	1	0	0	0	0	0	2	50	50	100
	Cluster auto-scaling	2	0	0	0	2	0	0	2	0	0	100
	Logging	6	1	1	0	2	0	0	6	17	17	100
	Debugging	2	1	0	0	1	0	1	2	50	100	100
	Cluster upgrades and maintenance	7	2	0	0	2	0	2	7	28	28	86
	Multi-cloud clusters	4	0	0	0	1	0	0	4	0	0	100
ServiceCatalog	1	0	0	0	0	0	0	1	0	0	100	
<b>Total</b>	<b>256</b>	<b>104</b>	<b>12</b>	<b>10</b>	<b>26</b>	<b>9</b>	<b>92</b>	<b>245</b>	<b>41</b>	<b>68</b>	<b>99</b>	

Dark, light and medium shaded cells correspond respectively with reliable e2e tests for all studied vendors, with other matching tests of which the test results produce false negatives, or the tests are skipped, and with configuration proof that validates specific features for all vendors. The fourth to last column presents the number of features with inconsistencies between test results and vendor documentation. The last three columns show the accumulated feature coverage.

pipeline backed-up by a time-series database. Kube-state-metrics [91], which is supported by all vendors, is an alternative for metrics-server for monitoring status and health of a broad range of application-level resources such as persistent volumes. The default full metrics pipelines are proprietary in each vendor of the hosted type, but it is possible to install the CNCF-supported Prometheus pipeline in all vendors [92].

For central monitoring of resource usage of the k8s platform itself, the e2e tests show that is possible to grab metrics from various control plane components such as the default ControllerManager, the Scheduler and API server on the master nodes and the kubelet on each node.

Logging and debugging at the level of containers and platform is also commonly supported by each vendor, yet different log aggregation systems are used. It is, however, possible to customize all worker nodes with a unified Fluentd DaemonSet [93].

Concerning cluster maintenance, relevant features include cluster software upgrades, dynamic reconfiguration of the kubelet without restarting Pods, draining of nodes, garbage collection of container and images and support for disruption budgets [94]. The implementation of cluster upgrades is proprietary to every Kubernetes vendor. Dynamic reconfiguration of the kubelet is by default disabled in all vendors because the `-dynamic-config-file` parameter is not set, and it cannot be set using the KubeletConfiguration API [95]. The other features are inherited from the open-source k8s-distribution and commonly supported by each vendor.

All vendors support configurations for a single Kubernetes cluster that runs across multiple zones or regions (see Section III.A). Kubernetes’s federation API that allows managing clusters into a federation and that offers federated instantiations of various Kubernetes APIs, such as Deployments and StatefulSets, is an alpha feature in v1.13. As such, we must assume that this API is not supported by any cloud provider except GKE’s alpha clusters. GKE offer proprietary support for a multi-cluster global HTTP Load balancer [96]. Still, in the other vendors, it is possible to set up a similar load balancer using the Istio ingress gateway [97]. Finally, the ServiceCatalog API [98] is supported by all vendors.

#### IV. DISCUSSION

This section builds upon Sections II.B-C, which respectively have presented the infrastructure-agnostic approach for transferring container clusters across cloud providers and a proposal towards extending this approach with feature compatibility management.

The latter extension involves (i) semi-automatic detection of support for desired features by means of the e2e testing suite of Kubernetes and (ii) uniform support for installing or removing features by means of generic configuration actions. We will discuss the potential of Kubernetes to support these two elements as follows.

Section A focuses on the feature coverage of the e2e testing suite and how to improve this coverage. It also looks at the accumulated feature coverage of all three information assets to show that the vendors have been validated against the whole feature taxonomy. After that, Section B looks at the question for which features it is possible to install missing system components uniformly using vendor-agnostic tactics that only rely on Kubernetes-defined APIs or tools.

#### A. FEATURE COVERAGE OF THE INFORMATION ASSETS

Table 8 gives a detailed overview of the feature coverage of the e2e testing suite of Kubernetes. Matching tests with a valid test result cover only 41% of the 256 features. The feature coverage differs substantially across different functional aspects. Valid test results for resource quota management and container security yield a feature coverage of 100% and 90%, respectively. The application configuration and deployment aspect exhibits a feature coverage of 83%. Subsequently, services networking and container QoS management aspects have an average coverage of 44%. The other four aspects have a coverage between 33% and 13%.

The substantial number of matching e2e tests without valid test results in Table 8 demonstrates the potential for improving the feature coverage of the e2e testing suite. If these matching tests would have produced valid test results, however, the feature coverage for the container security aspect could be lifted from 90% to 100%, the framework customization and the management aspects could be raised from 33% to, respectively, 55% and 70%; services networking and container QoS management could be lifted from an average of 43% to 53% and the cluster architecture aspect from 13% to 25%.

These tests can be improved to produce valid results in two ways. First, tests for 26 features should be developed for all studied vendors. Secondly, enhanced ease-of-configuration of the testing suite could avoid the occurrences of skipped tests and false negatives for 22 features. These occurrences were due to unspecified configuration parameters (e.g. an SSH key is needed for accessing worker nodes, but this was not documented). The e2e testing suite should be extended with a proper configuration interface for managing all required configuration parameters.

Thus, to achieve full coverage, the testing suite needs to be extended, and its ease-of-configuration improved.

Another avenue is to complement the end-to-end testing suite with the other information assets to improve feature coverage but also to filter false negatives. Unfortunately, an inspection of configuration state in combination with valid test results only adds up towards 68% feature coverage (this accumulative ratio is shown in the second to last column of Table 8). For more recent releases of Kubernetes, all studied vendors have increased the observability of configuration state of the master control plane by exposing its logging files. As such, feature coverage for later k8s releases is expected to be substantially higher.

For k8s release v1.13, however, vendor documentation needs to be added as a third complementary source. Table 8 shows that vendor documentation is quite accurate. By comparing existing vendor documentation with e2e test results, we could detect inconsistencies for 12 features. For seven features, vendor documentation was not up-to-date. For the other five features, vendor documentation and configuration state asserts the features to be supported, but the e2e tests show that the features do not function well.

In total, these information assets do not offer conclusive information for only two k8s features: support for raw volumes in AKS and GKE [99], backup and recovery of cluster state [100].

#### B. EASE OF MIGRATION AND UNIFORM RE-CONFIGURABILITY

As stated in Section II.E, we distinguish between different levels of ease and automation with which a k8s feature can be consistently activated when transferring a cluster from a source vendor to a target vendor: (i) automated migration, (ii) uniform reconfiguration, (iii), custom translation of input or output data (e.g. different data formats for logging, monitoring, auditing), (iv) migration is not possible. Using these levels of ease-of-migration, we can assess for each (sub)-aspect of the feature taxonomy the ease of which feature incompatibilities can be bridged between any pair of Kubernetes vendors.

Table 9 systematically indicates the specific level for the nine functional aspects and their respective sub-aspects. Since a (sub)-aspect can involve multiple features. We need to consider the set of all valid subsets of these features that can be supported by the source vendor. We therefore further distinguish between intermediate ease-of-migration levels that indicate whether all or only some subsets of features of a sub-aspect can be activated or reconfigured. We refer to the detailed score legend of Table 9 for more information.

We can draw conclusions by counting the number of occurrences of a particular score in the grey shaded columns in Table 9). These columns represent migration scenarios from the open-source k8s distribution, where all customization interfaces of Kubernetes are available as is. The scores in these columns represent thus an objective measure for comparing vendors.

Vendors cannot support a common vendor-agnostic API for all sub-aspects with a score  $\leq 2$ . The lower the number of those scores, the more customizable the vendor is.

GKE appears as the most customizable vendor according to the number of scores  $\leq 2$ . This is at odds with the observation that EKS exposes the largest number of customization interfaces (see Table 2). More specifically, EKS allows customizing feature gates and several other kubelet configuration options for worker nodes. EKS also allows booting worker nodes from custom VM images, whereas GKE only allows choosing from an existing catalogue of VM images and cluster workload types.

This odd observation can be explained by the fact that EKS for k8s release v1.13 still contains the most errors according to the failed e2e tests (see the brownish shaded cells of Table 9). These failed e2e tests indicate errors because the test results are inconsistent with vendor documentation or configuration state.

**Score legend for a sub-aspect in Table 9:**  
*Ease of migration from a source vendor to a target vendor (alpha features are not considered):*  
**5:** For all subsets of features supported by the source vendor, automated migration is possible  
**4:** For all subsets of features supported by the source vendor, automated migration or vendor-agnostic reconfiguration is possible  
**3:** For all subsets of features supported by the source vendor, only vendor-agnostic reconfiguration is possible  
**2:** For some subsets of features supported by the source vendor, automated migration or vendor agnostic reconfiguration is possible  
**1:** Only custom, vendor-specific translation of input/output data is possible for some subsets of features supported by the source vendor  
**0:** None of the feature subsets supported by the source vendor are supported by the target vendor  
**-:** None of the feature subsets of the sub-aspect are supported by the source vendor  
**Brownish shaded cells:** The score could be improved by correcting configuration errors. The features are supported by the target vendor according to the vendor documentation, but do not function correctly according to failed e2e tests.

- In particular, the most significant errors are the following:
- Volume plugins: EKS fails for a large part of relevant e2e tests for the subPath feature.
  - IAM modules: AKS fails for a large part of relevant e2e tests concerning NodeAuthorization.
  - Services networking: The ExternalName service does not function properly in EKS according to the e2e tests.
  - Cluster security: AKS fails for a large part of relevant e2e tests with respect to NetworkPolicies.
  - Container security: EKS fails tests for PodSecurity Policies.

If these errors would have been resolved, EKS has the lowest number of scores  $\leq 2$  and therefore can be considered as the most customizable and feature-rich. Note GKE exhibits the highest number of scores  $= 5$ , and therefore GKE offers the highest amount of features out-of-the-box.

Note there are no occurrences of scores  $= 1$  meaning that all studied vendors offer Kubernetes-native support for monitoring, audit and logging.

Finally, the number of scores  $\leq 2$  in the non-shaded columns is much lower than the grey-shaded columns indicating that the studied vendors themselves are quite compatible. As expected, feature lock-in is the strongest for GKE, then followed by EKS and AKS.

Still, even with a vendor-agnostic API in place and the above EKS and AKS errors corrected, some features of the open-source distribution cannot be consistently imposed across all vendors:

- Feature gates: Only in EKS specific subsets of feature gates can be further disabled or enabled by the cluster administrator.

**TABLE 9. Uniform re-configurability analysis and migration analysis.**

Aspects   Sub-aspects	Migration scenarios										
	K8s→AKS	EKS→AKS	GKE→AKS	K8s→EKS	AKS→EKS	GKE→EKS	K8s→GKE	AKS→GKE	EKS→GKE		
Cluster architecture and setup	2	2	2	3	3	2	3	3	3		
CO framework customization	Feature gates master node	2	5	5	2	5	5	2	2	2	
	Feature gates worker node	2	2	5	3	5	5	2	5	2	
	Admission controllers	2	2	2	2	2	2	2	2	5	
	CRI	Containerd	0	-	0	3	-	3	5	-	5
		cri-o	0	0	-	3	-	-	0	-	0
	Network plugins	Kubenet	3	-	3	0	0	0	3	3	-
		CNI	2	3	3	2	3	3	2	3	3
	Volume plugins	3	5	5	2	2	2	3	5	5	
	Scheduler plugins	5	5	5	5	5	5	5	5	5	
	IAM master nodes	Authentication	2	5	5	2	5	5	2	5	5
		Authorization	2	5	2	2	5	2	2	5	5
		Node authoriz.	0	0	0	5	-	5	5	-	5
		Audit	3	3	3	3	3	3	5	5	5
	IAM worker nodes	Authent.	2	2	2	5	5	5	5	5	5
		Authorizat.	5	5	5	5	5	5	5	5	5
CRD and annotations	5	5	5	5	5	5	5	5	5		
Extended node resources	3	3	3	3	3	3	3	3	3		
CloudController API	0	-	-	0	-	-	0	-	-		
KubeletConfiguration API	0	0	-	3	-	-	0	-	0		
Service networking	ClusterIP service	5	5	5	5	5	5	5	5	5	
	NodePort service	2	5	5	2	3	3	2	3	3	
	LoadBalancer service	5	5	5	5	5	5	5	5	5	
	L4 Loadbalancer+ iptables	5	5	5	5	5	5	5	5	5	
	L4 Loadbalancer + ipvs	3	3	3	3	3	3	3	3	3	
	L7 Loadbalancer	5	5	5	3	3	3	5	5	5	
	ExternalName service	5	5	5	0	0	0	5	5	5	
Headless service	5	5	5	5	5	5	5	5	5		
App. Config.	Workload configurations	2	5	2	2	3	2	5	5	5	
	Reusable contain. conf.	5	5	5	5	5	5	5	5	5	
	(Rolling) updates	5	5	5	5	5	5	5	5	5	
Resource quota management	5	5	5	5	5	5	5	5	5		
Container QoS management	2	2	5	3	5	5	2	5	2		
Cluster security	Network policies	0	0	0	5	5	5	5	5	5	
	Encryption of control- and application-level messages	-	0	0	-	-	2	-	-	5	
	Private subnets	-	3	3	-	3	3	-	3	3	
Container security	Secrets	5	5	5	5	5	5	5	5	5	
	Managing private repo.	5	5	5	5	5	5	5	5	5	
	SecurityContext support	3	3	3	3	5	5	3	5	5	
	PodSecurityPolicy	5	5	5	0	0	0	5	5	5	
	Safe sysctl support	5	5	5	5	5	5	5	5	5	
Application and cluster management	K8s dashboard	5	5	5	5	5	5	5	5	5	
	Core metrics server	5	5	5	3	3	3	5	5	5	
	Prometheus pipeline	3	3	3	3	3	3	3	3	3	
	Control plane metrics	5	5	5	5	5	5	5	5	5	
	Log aggregation service	3	3	3	3	3	3	3	3	3	
	Debugging	5	5	5	5	5	5	5	5	5	
	Cluster upgrades	-	3	3	-	3	3	-	3	3	
	Disruption budget	5	5	5	5	5	5	5	5	5	
	Draining of nodes	5	5	5	5	5	5	5	5	5	
	Multi-region clusters	3	3	3	3	3	3	3	3	3	
	Cluster federation	2	2	2	2	2	2	3	3	3	
	Cross-region load bal.	3	3	3	3	3	3	3	3	3	
	ServiceCatalog	5	5	5	5	5	5	5	5	5	
Total occurrences of score $\leq 2$	18	11	10	13	6	10	11	2	5		
Total occurrences of score = 5	23	29	29	22	27	27	29	33	35		

- KubeletConfiguration API: Only in EKS, some fields of the KubeletConfiguration API can be modified statically.
- Container QoS Management: Only in EKS, CPU management policies can be set through the KubeletConfiguration API.

There are also several differences for essential features that are not part of the open-source k8s distribution:

- Cluster setup: Only EKS and GKE offer an automated highly-available master plane.
- Cluster security: Only in GKE, application-level network encryption is possible. Only GKE and EKS support encryption of control plane messages.

In summary, the following selection guideline can be formulated:

- If vanilla Kubernetes cluster configurations are sufficient, GKE is the most complete, scalable and reliable offer.
- If highly customized Kubernetes clusters are required, EKS is the preferred choice as it exposes the Kubelet-Configuration API as-is. As a consequence, not only alpha feature gates but also performance-critical features of the Kubelet can only be activated in EKS without nullifying the goal of vendor-neutrality.
- EKS is also the only vendor that supports the cri-o container runtime for running OCI-compliant container images.
- Concerning dependability and network security, AKS is the weakest offer and GKE the strongest.

## V. RELATED WORK

This section first describes existing surveys that have defined a taxonomy or comparison of container orchestration platforms and how they are different from our feature taxonomy. After that, we review other works in the area of infrastructure-agnostic management of container clusters across cloud providers.

### A. CONTAINER ORCHESTRATION SURVEYS

Besides our feature comparison studies [5], [19], [20], Rodriguez *et al.* [103] present a taxonomy for classifying container scheduling architectures and multi-tenancy. However, this work is not a complete feature taxonomy as it does not cover the framework customization aspect. Heidari *et al.* [86] present a survey of seven container orchestration frameworks that were identified as most promising: Apache Mesos, Mesos Marathon, Apache Aurora, Kubernetes, Docker Swarm and Fleet. This survey concisely and clearly describes the QoS management architectures of these frameworks.

Costache *et al.* [104] present a classification of resource management techniques in Platform-as-a-Service (PaaS) platforms, including Mesos [105] and Borg [106], the predecessor of Kubernetes. Costache *et al.* also present a list of opportunities for further research, which includes

the use of container orchestration frameworks to support generic resource management for any type of workload and provisioning of resources across multiple IaaS clouds.

### B. INFRASTRUCTURE-AGNOSTIC MANAGEMENT OF CONTAINER CLUSTERS

We already referred to our own work on an infrastructure-agnostic middleware platform to transfer container clusters from one cloud provider to another cloud provider [6], [17]. As the requirements of this middleware platform favor pragmatism over expressiveness [20], the middleware platform supports commonly supported features that are supported by Kubernetes, Docker Swarm and Mesos and therefore ignore many unique features of Kubernetes. Pahl *et al.* [107] analyses required container orchestration functions for facilitating deployment and management of distributed applications across multiple clouds and how these functions can be integrated into existing PaaS platforms and relevant standards for portable orchestration of cloud applications such as TOSCA. Kim *et al.* [102] propose an integration between TOSCA and Kubernetes to deploy a container-based application across multiple federated Kubernetes clusters in different continents. These works indicate the relevance of achieving feature compatibility between different Kubernetes clusters that are potentially installed by different vendors or cloud providers.

## VI. CONCLUSION

We have studied the differences in cluster architecture and framework customization for three leading Kubernetes vendors of the hosted product type and synthesized the feature incompatibilities that ensue from these differences.

In total, we identified incompatibilities for 18% of documented k8s features when comparing default cluster configurations of the studied vendors.

Thereafter we have evaluated the feature coverage of three information assets of Kubernetes that are amendable for automated processing to detect whether a feature is (correctly) implemented by a vendor: (i) tests results from the e2e testing suite of Kubernetes that underlies the CNCF certification programme, (ii) configuration state of running Kubernetes clusters and (iii) vendor documentation. Our analysis has focused on the e2e testing suite, which is the most logical approach for automated feature detection. Our findings show that there is quite some room for improving feature coverage: matching e2e tests only exist for 64% of documented features, and for 17% of features, these matching tests are not developed for all studied vendors or are challenging to configure correctly.

Finally, we have presented an analysis of the ease-of-migration between vendors and the potential of defining a vendor-agnostic API for uniform feature management. This shows that for most beta features of the open-source k8s distribution, the possibility exists to define a standard API for managing feature compatibility among these three vendors. The exceptional features are (i) the cri-o container runtime, (ii) customizations to feature gates for worker nodes, (iii) the



KubeletConfiguration API, and (iv) CPU management policies. The latter three incompatibilities can be resolved if the CNCF conformance program enforces that vendors activate the KubeletConfiguration API.

These insights are beneficial to avoid feature incompatibilities already in cloud-native application engineering processes. Developers, operators and site reliability engineers can use the feature taxonomy to determine what k8s features should be installed or deactivated. Moreover, they can use the presented methodology to identify k8 vendors that offer standardized or infrastructure-agnostic interfaces for configuring these features. Finally, they can efficiently use the end-to-end testing suite of Kubernetes by only running those tests that have been identified by our feature mapping as applicable to the features of their interest.

Cloud k8s vendors can also increase their cross-compatibility with other k8s vendor solutions by applying the presented methodology. Moreover, the k8s SIGS can increase their end-to-end tests in currently unlighted areas to provide better feature coverage.

All can equally contribute to reducing vendor lock-in. The feature taxonomy is, to some degree, comparable to POSIX that harmonized Unix-oid single machine operating systems in the past.

## SUPPLEMENTARY MATERIAL

The end-to-end test results, the code for automated processing of tests results and various excel sheets for quantitative analysis are available at Code Ocean (<https://codeocean.com/capsule/2358221>) or GitHub (<https://github.com/k8-scalar/MigratingKubernetes>).

## REFERENCES

- [1] N. Kratzke and R. Peinl, "ClouNS—a cloud-native application reference model for enterprise architectures," in *Proc. IEEE 20th Int. Enterprise Distrib. Object Comput. Workshop (EDOCW)*, Sep. 2016, pp. 1–10.
- [2] J. Opara-Martins, R. Sahandi, and F. Tian, "Critical analysis of vendor lock-in and its impact on cloud computing migration: A business perspective," *J. Cloud Comput.*, vol. 5, no. 1, p. 4, Dec. 2016.
- [3] D. Petcu, G. Macariu, S. Panica, and C. Craciun, "Portable cloud applications—From theory to practice," *Future Gener. Comput. Syst.*, vol. 29, no. 6, pp. 1417–1430, Aug. 2013.
- [4] N. Kratzke, "A brief history of cloud application architectures," *Appl. Sci.*, vol. 8, no. 8, p. 1368, Aug. 2018.
- [5] N. Kratzke and P.-C. Quint, "Understanding cloud-native applications after 10 years of cloud computing—A systematic mapping study," *J. Syst. Softw.*, vol. 126, pp. 1–16, Apr. 2017.
- [6] N. Kratzke, "About the complexity to transfer cloud applications at runtime and how container platforms can contribute?" in *Proc. Int. Conf. Cloud Comput. Services Sci. (CLOSER)*, 2017, pp. 19–45.
- [7] The Cloud Native Computing Foundation. (2017). *Cloud Native Computing Foundation Launches Certified Kubernetes Program with 32 Conformance Distributions and Platforms*. Accessed: Mar. 27, 2018. [Online]. Available: <https://www.cncf.io/announcement/2017/11/13/cloud-native-computing-foundation-launches-certified-kubernetes-program-32-conformant-distributions-platforms/>
- [8] Cloud Native Computing Foundation. *Community/Sig-List.Md At Master · Kubernetes/Community*. Accessed: Sep. 13, 2019. [Online]. Available: <https://github.com/kubernetes/community/blob/master/sig-list.md>
- [9] S. Sloka and J. Schnake. *Certifying Kubernetes With Sonobuoy—Cloud Native Apps Blog*. Accessed: Nov. 28, 2019. [Online]. Available: <https://blogs.vmware.com/cloudnative/2019/02/21/certifying-kubernetes-with-sonobuoy/>
- [10] J. Schnake. *Simple Approaches to Customizing the Kubernetes E2E Tests*. Accessed: Oct. 17, 2019. [Online]. Available: <https://sonobuoy.io/custom-e2e-image/>
- [11] R. Buyya, R. Ranjan, and R. N. Calheiros, "InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proc. 10th Int. Conf. Algorithms Archit. Parallel Process. (ICA3PP)*, Part 1, in Lecture Notes in Computer Science, vol. 6081. Busan, South Korea: Springer, May 2010, pp. 13–31.
- [12] B. Wadhwa, A. Jaitly, and B. Suri, "Cloud service brokers: An emerging trend in cloud adoption and migration," in *Proc. 20th Asia-Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2013, pp. 140–145.
- [13] Cloud Native Computing Foundation. *Kubernetes Distributions & Platforms—Google Sheets*. Accessed: Mar. 15, 2019. [Online]. Available: [https://docs.google.com/spreadsheets/d/1LxSqBzjOxfGx3cmtZ4Ebb\\_BGCxT\\_wlxW\\_xgHVVa23es/edit#gid=0](https://docs.google.com/spreadsheets/d/1LxSqBzjOxfGx3cmtZ4Ebb_BGCxT_wlxW_xgHVVa23es/edit#gid=0)
- [14] Microsoft Azure. *Azure Kubernetes Service (AKS) Documentation-Tutorials, API Reference | Microsoft Docs*. Accessed: Jan. 24, 2020. [Online]. Available: <https://docs.microsoft.com/en-us/azure/aks/>
- [15] AWS. *Amazon Elastic Kubernetes Service Documentation*. Accessed: Jan. 24, 2020. [Online]. Available: [https://docs.aws.amazon.com/eks/?id=docs\\_gateway](https://docs.aws.amazon.com/eks/?id=docs_gateway)
- [16] Google. *Google Kubernetes Engine Documentation*. Accessed: Jan. 24, 2020. [Online]. Available: <https://cloud.google.com/kubernetes-engine/docs/?hl=nl>
- [17] N. Kratzke, "Smuggling multi-cloud support into cloud-native applications using elastic container platforms," in *Proc. 7th Int. Conf. Cloud Comput. Services Sci. (CLOSER)*. Lagos, Portugal: SCITEPRESS—Science and Technology Publications, 2017, pp. 29–42.
- [18] S. Baset, S. Suneja, N. Bila, O. Tuncer, and C. Isci, "Usable declarative configuration specification and validation for applications, systems, and cloud," in *Proc. 18th ACM/IFIP/USENIX Middleware Conf. Ind. Track*, 2017, pp. 29–35.
- [19] E. Truyen, D. V. Landuyt, D. Preuveneers, B. Lagaisse, and W. Joosen, "A comprehensive feature comparison study of open-source container orchestration frameworks," *Appl. Sci.*, vol. 9, no. 5, p. 931, Mar. 2019.
- [20] N. Kratzke and P.-C. Quint, "Project CloudTRANSIT transfer cloud-native applications at runtime," Lübeck, Germany, Tech. Rep., 2018.
- [21] Amazon Web Services. (2017). *Introducing Amazon Elastic Container Service for Kubernetes (Preview)*. [Online]. Available: <https://aws.amazon.com/about-aws/whats-new/2017/11/introducing-amazon-elastic-container-service-for-kubernetes/>
- [22] Docker Inc. *Docker Enterprise | Docker Documentation*. Accessed: Nov. 16, 2018. [Online]. Available: <https://docs.docker.com/ee/#kubernetes-support>
- [23] Mesosphere. *Mesosphere/Dcos-Kubernetes-Quickstart: Quickstart Guide for Kubernetes on DC/OS*. Accessed: Nov. 16, 2018. [Online]. Available: <https://github.com/mesosphere/dcos-kubernetes-quickstart#install>
- [24] P. Beth. (2018). *Kubernetes Security Vulnerability Reveals Fractured Market*. TechTarget. Accessed: Dec. 11, 2018. [Online]. Available: <https://searchitoperations.techtarget.com/news/252453827/Kubernetes-security-vulnerability-reveals-fractured-market>
- [25] J. Steven Vaughan-Nichols. (2018). *Kubernetes' First Major Security Hole Discovered | ZDNet*. Accessed: Dec. 11, 2018. [Online]. Available: <https://www.zdnet.com/article/kubernetes-first-major-security-hole-discovered/>
- [26] A. Modak, S. D. Chaudhary, P. S. Paygude, and S. R. Ldate, "Techniques to secure data on cloud: Docker swarm or kubernetes?" in *Proc. 2nd Int. Conf. Inventive Commun. Comput. Technol. (ICICCT)*, Apr. 2018, pp. 7–12.
- [27] Cloud Native Computing Foundation. *Kubernetes End-to-End Testing for Everyone-Kubernetes*. Accessed: Sep. 13, 2019. [Online]. Available: <https://kubernetes.io/blog/2019/03/22/kubernetes-end-to-end-testing-for-everyone/>
- [28] *Kubernetes Cloud—Google Sheets*. Accessed: Mar. 27, 2019. [Online]. Available: <https://docs.google.com/spreadsheets/d/1U0x4NQegEPM7eVTKJemhkPy18LWuHW5vX8uZzqYo/edit#gid=0>
- [29] *Config File Schema-Eksctl*. Accessed: May 4, 2020. [Online]. Available: <https://eksctl.io/usage/schema/>
- [30] *Limits for Resources, SKUs, Regions—Azure Kubernetes Service | Microsoft Docs*. Accessed: May 26, 2020. [Online]. Available: <https://docs.microsoft.com/en-us/azure/aks/quotas-skus-regions>
- [31] *Amazon EKS Service Quotas—Amazon EKS*. Accessed: May 26, 2020. [Online]. Available: <https://docs.aws.amazon.com/eks/latest/userguide/service-quotas.html>

- [32] *Quotas and Limits | Kubernetes Engine API | Google Cloud*. Accessed: May 26, 2020. [Online]. Available: <https://cloud.google.com/kubernetes-engine/quotas?hl=en>
- [33] *Azure Quickstart Templates*. Accessed: May 13, 2020. [Online]. Available: <https://azure.microsoft.com/en-us/resources/templates/?term=aks>
- [34] *The Kubernetes API-Kubernetes*. Accessed: May 26, 2020. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/kubernetes-api/#api-groups>
- [35] Cloud Native Computing Foundation. *Feature Gates-Kubernetes*. Accessed: Dec. 5, 2019. [Online]. Available: <https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates/>
- [36] Cloud Native Computing Foundation. *Extending Your Kubernetes Cluster-Kubernetes*. Accessed: Dec. 3, 2019. [Online]. Available: <https://kubernetes.io/docs/concepts/extend-kubernetes/extend-cluster/>
- [37] Cloud Native Computing Foundation. *Kubernetes Cloud Controller Manager-Kubernetes*. Accessed: Dec. 5, 2019. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/running-cloud-controller/>
- [38] Cloud Native Computing Foundation. *Website/Feature-Gates.md at Release-1.13 · Kubernetes/Website*. Accessed: Nov. 16, 2018. [Online]. Available: <https://github.com/kubernetes/website/blob/release-1.13/content/en/docs/reference/command-line-tools-reference/feature-gates.md>
- [39] Cloud Native Computing Foundation. *Website/Kubelet.md at Release-1.13 · Kubernetes/Website*. Accessed: Apr. 29, 2020. [Online]. Available: <https://github.com/kubernetes/website/blob/release-1.13/content/en/docs/reference/command-line-tools-reference/kubelet.md>
- [40] Cloud Native Computing Foundation. *Customizing Kubelet Configuration—Eksctl*. Accessed: Apr. 29, 2020. [Online]. Available: <https://eksctl.io/usage/customizing-the-kubelet/>
- [41] Cloud Native Computing Foundation. *Website/Admission-Controllers.md at Release-1.13 · Kubernetes/Website*. Accessed: Nov. 12, 2018. [Online]. Available: <https://github.com/kubernetes/website/blob/release-1.13/content/en/docs/reference/access-authn-authz/admission-controllers.md>
- [42] Microsoft. *Frequently Asked Questions for Azure Kubernetes Service (AKS) | Microsoft Docs*. Accessed: Mar. 28, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/azure/aks/faq#what-kubernetes-admission-controllers-does-aks-support-can-admission-controllers-be-added-or-removed>
- [43] Amazon Web Services. *Platform Versions—Amazon EKS*. Accessed: Mar. 28, 2019. [Online]. Available: <https://docs.aws.amazon.com/eks/latest/userguide/platform-versions.html>
- [44] A. A. Balkan and Y. Tamura. *Multi-Tenancy Best Practices for Google Kubernetes Engine-Speaker Deck*. Accessed: Mar. 28, 2019. [Online]. Available: <https://speakerdeck.com/alp/multi-tenancy-best-practices-for-google-kubernetes-engine?slide=38>
- [45] Cloud Native Computing Foundation. *Using Admission Controllers-Kubernetes*. Accessed: Mar. 28, 2019. [Online]. Available: <https://github.com/kubernetes/website/blob/release-1.13/content/en/docs/reference/access-authn-authz/admission-controllers.md#which-plugins-are-enabled-by-default>
- [46] Cloud Native Computing Foundation. *Configure Multiple Schedulers-Kubernetes*. Accessed: Mar. 11, 2020. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/>
- [47] Cloud Native Computing Foundation. *Network Plugins-Kubernetes*. Accessed: Mar. 29, 2019. [Online]. Available: <https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/>
- [48] Amazon Web Services. *Aws/Amazon-Vpc-Cni-K8s: Networking Plugin Repository for Pod Networking in Kubernetes Using Elastic Network Interfaces on AWS*. Accessed: Mar. 29, 2019. [Online]. Available: <https://github.com/aws/amazon-vpc-cni-k8s>
- [49] Microsoft Azure. *Azure-Container-Networking/Cni.md at Master · Azure/Azure-Container-Networking*. Accessed: Mar. 29, 2019. [Online]. Available: <https://github.com/Azure/azure-container-networking/blob/master/docs/cni.md>
- [50] Microsoft Azure. *Secure Pods With Network Policies in Azure Kubernetes Service (AKS) | Microsoft Docs*. Accessed: Mar. 29, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/azure/aks/use-network-policies>
- [51] Amazon Web Services. *Installing Calico on Amazon EKS—Amazon EKS*. Accessed: Mar. 29, 2019. [Online]. Available: <https://docs.aws.amazon.com/eks/latest/userguide/calico.html>
- [52] Google. *Creating a Cluster Network Policy | Kubernetes Engine | Google Cloud*. Accessed: Mar. 29, 2019. [Online]. Available: [https://cloud.google.com/kubernetes-engine/docs/how-to/network-policy#creating\\_a\\_network\\_policy](https://cloud.google.com/kubernetes-engine/docs/how-to/network-policy#creating_a_network_policy)
- [53] Google. *GoogleCloudPlatform/Netd: Netd: GKE Networking Daemonset*. Accessed: Mar. 29, 2019. [Online]. Available: <https://github.com/GoogleCloudPlatform/netd>
- [54] Cloud Native Computing Foundation. *Open Container Initiative-based implementation of Kubernetes Container Runtime Interface*. Accessed: Nov. 4, 2019. [Online]. Available: <https://github.com/cri-o/cri-o>
- [55] D. Radetic. (2018). *CentOS 7 With CRI-o on EKS*. Medium. Accessed: Mar. 29, 2019. [Online]. Available: <https://medium.com/ernothxbye/centos-7-with-cri-o-on-eks-ae9684af764>
- [56] Google. *Using Container-Optimized OS with containerd | Kubernetes Engine | Google Cloud*. Accessed: Mar. 28, 2019. [Online]. Available: <https://cloud.google.com/kubernetes-engine/docs/concepts/using-containerd>
- [57] [EKS] [CRI]: *Support for Containerd CRI · Issue #313 · Aws/Containers-Roadmap*. Accessed: May 13, 2020. [Online]. Available: <https://github.com/aws/containers-roadmap/issues/313>
- [58] Cloud Native Computing Foundation. *Persistent Volumes—Kubernetes*. Accessed: Nov. 4, 2019. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>
- [59] Kubernetes. *Introduction-Kubernetes CSI Developer Documentation*. Accessed: Nov. 4, 2019. [Online]. Available: <https://kubernetes-csi.github.io/docs/>
- [60] *Drivers-Kubernetes CSI Developer Documentation*. Accessed: Apr. 1, 2019. [Online]. Available: <https://kubernetes-csi.github.io/docs/drivers.html>
- [61] H. Kumar. *Resizing Persistent Volume Using, Kubernetes-Kubernetes*. Accessed: Apr. 1, 2020. [Online]. Available: <https://kubernetes.io/blog/2018/07/12/resizing-persistent-volumes-using-kubernetes/>
- [62] Cloud Native Computing Foundation. *Node-specific Volume Limits—Kubernetes*. Accessed: Apr. 1, 2020. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/storage-limits/#dynamic-volume-limits>
- [63] Cloud Native Computing Foundation. *Volumes-Using Subpath*. Accessed: Nov. 8, 2019. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/volumes/#using-subpath>
- [64] Microsoft Azure. *Service Principals for Azure Kubernetes Services (AKS) | Microsoft Docs*. Accessed: Jan. 28, 2020. [Online]. Available: <https://docs.microsoft.com/en-us/azure/aks/kubernetes-service-principal>
- [65] AWS. *Identity and Access Management for Amazon EKS—Amazon EKS*. Accessed: Jan. 28, 2020. [Online]. Available: <https://docs.aws.amazon.com/eks/latest/userguide/security-iam.html>
- [66] Google. *Creating Cloud IAM Policies | Kubernetes Engine Documentation*. Accessed: Jan. 28, 2020. [Online]. Available: <https://cloud.google.com/kubernetes-engine/docs/how-to/iam>
- [67] Cloud Native Computing Foundation. *Accessing the API-Kubernetes*. Accessed: Nov. 4, 2019. [Online]. Available: <https://kubernetes.io/docs/reference/access-authn-authz/>
- [68] Cloud Native Computing Foundation. *Using ABAC Authorization-Kubernetes*. Accessed: May 25, 2020. [Online]. Available: <https://kubernetes.io/docs/reference/access-authn-authz/abac/>
- [69] Cloud Native Computing Foundation. *Kubelet Authentication/Authorization-Kubernetes*. Accessed: Nov. 4, 2019. [Online]. Available: <https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet-authentication-authorization/#kubelet-authentication>
- [70] Cloud Native Computing Foundation. *Auditing-Kubernetes*. Accessed: Nov. 19, 2019. [Online]. Available: <https://kubernetes.io/docs/tasks/debug-application-cluster/audit/>
- [71] Cloud Native Computing Foundation. *Operator Pattern—Kubernetes*. Accessed: Apr. 3, 2020. [Online]. Available: <https://kubernetes.io/docs/concepts/extend-kubernetes/operator/>
- [72] S. Patnaik. *Initialize Your AKS Nodes With DaemonSets*. Accessed: Mar. 26, 2020. [Online]. Available: <https://medium.com/@patnaikshekhar/initialize-your-aks-nodes-with-daemonsets-679fa81fd20e>
- [73] Google. *Automatically Bootstrapping GKE Nodes With DaemonSets | Solutions*. Accessed: Mar. 26, 2020. [Online]. Available: <https://cloud.google.com/solutions/automatically-bootstrapping-gke-nodes-with-daemonsets>

- [74] J. Cowan. *The Problem With Kube-Proxy: Enabling IPVS on EKS*. Accessed: Mar. 26, 2020. [Online]. Available: <https://medium.com/@jeremy.i.cowan/the-problem-with-kube-proxy-enabling-ipvs-on-eks-169ac22e237e>
- [75] Google. *Creating and Configuring Instances | Container-Optimized OS*. Accessed: Apr. 2, 2020. [Online]. Available: [https://cloud.google.com/container-optimized-os/docs/how-to/create-configure-instance#using\\_cloud-init](https://cloud.google.com/container-optimized-os/docs/how-to/create-configure-instance#using_cloud-init)
- [76] Cloud Native Computing Foundation. *Community/Cloud-Provider-Refactoring.md at Master · Kubernetes/Community*. Accessed: Jan. 28, 2020. [Online]. Available: <https://github.com/kubernetes/community/blob/master/contributors/design-proposals/cloud-provider/cloud-provider-refactoring.md>
- [77] Cloud Native Computing Foundation. *Set Kubelet Parameters Via a Config File—Kubernetes*. Accessed: May 13, 2020. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/>
- [78] L. Zhu, L. Bass, and G. Champlin-Scharff, “DevOps and its practices,” *IEEE Softw.*, vol. 33, no. 3, pp. 32–34, May/June 2016.
- [79] B. Chavis. *Introducing Horizontal Pod Autoscaling for Amazon EKS | AWS Open Source Blog*. Accessed: Apr. 4, 2019. [Online]. Available: <https://aws.amazon.com/blogs/opensource/horizontal-pod-autoscaling-eks/>
- [80] Cloud Native Computing Foundation. *Service—Kubernetes*. Accessed: Nov. 22, 2019. [Online]. Available: <https://kubernetes.io/docs/concepts/services-networking/service/>
- [81] Microsoft. *Concepts—Networking in Azure Kubernetes Services (AKS) | Microsoft Docs*. Accessed: Nov. 22, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/azure/aks/concepts-network#services>
- [82] Cloud Native Computing Foundation. *Kubernetes/Ingress-Nginx: NGINX Ingress Controller for Kubernetes*. Accessed: Apr. 4, 2019. [Online]. Available: <https://github.com/kubernetes/ingress-nginx>
- [83] Google. *Setting up HTTP Load Balancing With Ingress|Kubernetes Engine Tutorials|Google Cloud*. Accessed: Nov. 22, 2019. [Online]. Available: <https://cloud.google.com/kubernetes-engine/docs/tutorials/http-balancer>
- [84] Cloud Native Computing Foundation. *Kubernetes-Sigs/Aws-Alb-Ingress-Controller: AWS ALB Ingress Controller for Kubernetes*. Accessed: Apr. 4, 2019. [Online]. Available: <https://github.com/kubernetes-sigs/aws-alb-ingress-controller/>
- [85] *Control CPU Management Policies on the Node—Kubernetes*. Accessed: May 13, 2020. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies/>
- [86] P. Heidari, Y. Lemieux, and A. Shami, “QoS assurance with light virtualization—A survey,” in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Dec. 2016, pp. 558–563.
- [87] Google. *Control Plane Security | Kubernetes Engine | Google Cloud*. Accessed: Apr. 5, 2019. [Online]. Available: <https://cloud.google.com/kubernetes-engine/docs/concepts/control-plane-security>
- [88] C. Puccio. *Turnkey Network Security and Continuous Compliance for Your Amazon EKS Cluster | AWS Partner Network (APN) Blog*. Accessed: Apr. 5, 2019. [Online]. Available: <https://aws.amazon.com/blogs/apn/turnkey-network-security-and-continuous-compliance-for-your-amazon-eks-cluster/>
- [89] Google. *Application Layer Transport Security | Documentation | Google Cloud*. Accessed: Apr. 5, 2019. [Online]. Available: <https://cloud.google.com/security/encryption-in-transit/application-layer-transport-security/>
- [90] Google. *Hardening Your Cluster’s Security | Kubernetes Engine Documentation | Google Cloud*. Accessed: Apr. 5, 2019. [Online]. Available: <https://cloud.google.com/kubernetes-engine/docs/how-to/hardening-your-cluster>
- [91] *Kubernetes/Kube-State-Metrics: Add-on Agent to Generate and Expose Cluster-Level Metrics*. Accessed: Apr. 1, 2020. [Online]. Available: <https://github.com/kubernetes/kube-state-metrics>
- [92] Cloud Native Computing Foundation. *Prometheus—CNCF Cloud Native Interactive Landscape*. Accessed: Apr. 2, 2020. [Online]. Available: <https://landscape.cncf.io/selected=prometheus>
- [93] Cloud Native Computing Foundation. *Kubernetes Fluentd-Fluentd*. Accessed: Apr. 2, 2020. [Online]. Available: <https://docs.fluentd.org/v/0.12/articles/kubernetes-fluentd>
- [94] Cloud Native Computing Foundation. *Disruptions—Kubernetes*. Accessed: Dec. 5, 2019. [Online]. Available: <https://kubernetes.io/docs/concepts/workloads/pods/disruptions/>
- [95] *Reconfigure a Node’s Kubelet in a Live Cluster-Kubernetes*. Accessed: Jun. 8, 2020. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/>
- [96] Google. *Setting up a Multi-Cluster Ingress | Kubernetes Engine Documentation | Google Cloud*. Accessed: Jan. 16, 2020. [Online]. Available: <https://cloud.google.com/kubernetes-engine/docs/how-to/multi-cluster-ingress>
- [97] G. Agarwal. (2020). *Istio Multi Cluster Service Mesh on Kubernetes | Better Programming*. Accessed: Jul. 20, 2020. [Online]. Available: <https://medium.com/better-programming/istio-service-mesh-on-multi-cluster-kubernetes-environment-518094cdcdc4>
- [98] Cloud Native Computing Foundation. *Service Catalog—Kubernetes*. Accessed: Feb. 10, 2020. [Online]. Available: <https://kubernetes.io/docs/concepts/extend-kubernetes/service-catalog/>
- [99] Cloud Native Computing Foundation. *Persistent Volumes—Kubernetes*. Accessed: Nov. 22, 2019. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#raw-block-volume-support>
- [100] Cloud Native Computing Foundation. *Operating EtcD Clusters for Kubernetes—Kubernetes*. Accessed: Apr. 1, 2020. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/>
- [101] P.-C. Quint and N. Kratzke, “Towards a lightweight multi-cloud DSL for elastic and transferable cloud-native applications,” in *Proc. 8th Int. Conf. Cloud Comput. Services Sci.*, 2018, pp. 400–408.
- [102] D. Kim, H. Muhammad, E. Kim, S. Helal, and C. Lee, “TOSCA-based and federation-aware cloud orchestration for Kubernetes container platform,” *Appl. Sci.*, vol. 9, no. 1, p. 191, Jan. 2019.
- [103] M. A. Rodriguez and R. Buyya, “Container-based cluster orchestration systems: A taxonomy and future directions,” *Softw., Pract. Exper.*, vol. 49, no. 5, pp. 698–719, May 2019.
- [104] S. Costache, D. Dib, N. Parlavantzas, and C. Morin, “Resource management in cloud platform as a service systems: Analysis and opportunities,” *J. Syst. Softw.*, vol. 132, pp. 98–118, Oct. 2017.
- [105] B. Hindman, A. Konwinski, A. Platform, F.-G. Resource, and M. Zaharia, “Mesos: A platform for fine-grained resource sharing in the data center,” in *Proc. 8th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2011, p. 22.
- [106] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at Google with Borg,” in *Proc. 10th Eur. Conf. Comput. Syst. (EuroSys)*, 2015, pp. 1–17.
- [107] C. Pahl, “Containerization and the PaaS cloud,” *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 24–31, May 2015.



**EDDY TRUYEN** received the Ph.D. degree from KU Leuven, in 2004.

From 2004 to 2009, he was a Postdoctoral Researcher with the Department of Computer Science, KU Leuven, and a member of research group DistriNet. Since 2009, he has been a Research Expert with Middleware. His research interests include adaptive middleware, dynamic reconfiguration, and engineering of customizable software services. He has been involved in national and international projects on adaptive middleware and cloud computing.



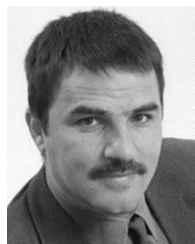
**NANE KRATZKE** received the Ph.D. degree from the University of Potsdam (and Oldenburg), in 2006.

He is currently a (Coding) Computer Scientist and a Professor of computer science with the Lübeck University of Applied Sciences. Before that, he consulted the German Ministry of Defense in questions of network-centric warfare as a Consulting Software Architect. His research interests include cloud-native application and cloud-native service-related software engineering methodologies and corresponding application architectural styles, such as microservices or serverless architectures. Additionally, he is interested in data science, distributed systems, and web-scale elastic systems.



**DIMITRI VAN LANDUYT** received the Ph.D. degree from KU Leuven, Belgium, in 2011, with a dissertation on robust and reusable interfaces for modular software architectures.

He is currently a Research Manager of software engineering with imec-DistriNet (Distributed Systems research group), Department of Computer Science, KU Leuven. His current research interests include applying and validating established software engineering principles to cloud computing, which among many other initiatives includes research on software engineering support for multi-tenant and multi-cloud storage architectures and self-adaptive systems.



**WOUTER JOOSEN** received the Ph.D. degree from KU Leuven, Belgium, in 1996, with a dissertation on load balancing of high performance computing applications.

He is currently a Full Professor of distributed software systems with the Department of Computer Science, KU Leuven. His current research interests include distributed systems and cloud computing, focusing on software architecture and adaptive middleware, as well as in security aspects of software. He has also co-founded spin-off companies of KU Leuven: Luciad, a company specializing in software components for Geographical Information Systems, and Ubizen (now part of Verizon Business Solutions), where he had been the CTO from 1996 till 2000, and the COO from 2000 till 2002.

...



**BERT LAGAISSE** received the Ph.D. degree from KU Leuven, Belgium, in 2009, with a dissertation on customization of enterprise middleware platforms using aspect-oriented and component-based software development techniques.

He is currently employed as an Industrial Research and a Valorization Manager with the imec-DistriNet research group in which he manages a portfolio of applied research projects on cloud technologies and security middleware in close collaboration with industrial partners. He has a strong interest in distributed systems, enterprise middleware, cloud platforms, and security services. He is experienced with industrial valorization of research as well as the cross-fertilization between academic know-how and industrial expertise in multi-partner industrial projects.