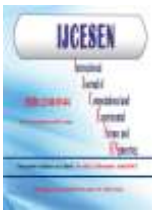
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.1 (2025) pp. 1769-1784
http://www.ijcesen.com



Copyright © IJCESEN

ISSN: 2149-9144

Research Article

Autonomous Multi-Cloud Kubernetes Orchestration Using Custom Resource Definitions and Controller Patterns

Shubham Srivastava*

Integral University, Lucknow, India
* **Corresponding Author Email:** shubhamjava1987@gmail.com - **ORCID:** 0000-0002-5247-366

Article Info:

DOI: 10.22399/ijcesen.5545
Received : 28 January 2025
Revised : 28 March 2025
Accepted : 30 March 2025

Keywords

Autonomous Kubernetes orchestration;
Custom Resource Definitions;
Multi-cloud control planes;
Controller reconciliation;
Policy-aware cloud federation

Abstract:

With the proliferation of cloud-native systems across multiple cloud providers, regions, and edge environments, multi-cloud Kubernetes orchestration, without any central management, is becoming a critical research challenge. This review explores the use of Custom Resource Definitions (CRD) and controller patterns to underpin policy-driven, self-adaptive orchestration of heterogeneous clusters of Kubernetes. A structured technical coding approach has been used to review literature from the field of cloud federation, container orchestration, model-driven deployment, self-adaptive systems, feedback control and edge/fog computing. The results of this review indicate that previous work has set solid ground for the analysis of declarative intent, placement optimisation, portability and adaptive control, although evidence is scattered across different research communities. In the coded comparison of representative studies, placement and resource allocation are given significantly more attention than the following: controller convergence, policy conflict handling, CRD schema quality and assurance. Five core domains of orchestration are identified: declarative intent modelling, reconciliation control, cross-cloud placement, interoperability, and observability-driven adaptability. The results indicate that CRDs should not be treated as simple configuration objects but rather as runtime models of desired state, constraints, and status. Controllers should be tested as feedback systems that exhibit measurable convergence, retry behaviour and interaction effects. A number of future research directions are worth exploring, such as a reproducible multi-cloud benchmark, formalised CRD schema design, secure controller authority boundaries, multi-controller coordination, and safe constrained learning for adaptive placement. The future contribution of this field lies in making multi-cloud automation measurable, explainable, and bounded.

1. Introduction

The current cloud-native systems are no longer limited to a particular region, cloud infrastructure provider or administrative domain, making autonomous multi-cloud Kubernetes orchestration a technically relevant research challenge. Cloud computing evolved from elastic infrastructure provisioning to a globally distributed software base for data intensive services, cyber-physical systems, pipelines of artificial intelligence, and latency sensitive edge applications [1]. This growth has driven the need for portability, policy-driven control, availability in the event of regional outages, cost-centric placement and regulatory conformance to disparate clouds. The first studies on cloud computing assumed elasticity as one of the key characteristics of large-scale service provision, along with virtualization and utility pricing [2]. Further inter-cloud studies revealed that provider heterogeneity, workload mobility, brokered placement and cross-cloud service composition added control-plane complexity which was not addressed by infrastructure abstraction [3]. Both of these trends intersect with multi-cloud Kubernetes orchestration, as Kubernetes is now a ubiquitous container-control substrate and enterprise systems increasingly require coordinated placement and adaptation across multiple Kubernetes clusters.

The relevance of the topic is highlighted by the operational nature of the modern digital infrastructure, which is important at the global level. These systems are frequently found in

financial applications, health informatics, scientific workflows, large-scale analytics, industrial automation, and public-sector services, and must be available even when the provider is down, can't operate in certain jurisdictions, or does not always have the resources needed to respond to demand. Next-generation cloud computing has been described as more distributed, decentralized, service-oriented, and automated centralized cloud to edge to domain-specific accelerator structure [4]. However, in these environments, scheduling decisions get complicated with issues of wide-area latency, inter-cloud egress cost, data-residency requirements, supply-chain security and variant service-level goals. Autonomous orchestration has thus become a must-have function and not an optimization. Placement, scaling, failover, and remediation policies are expected to be declared, monitored and acted upon by closed-loop controllers.

CRDs and controller patterns are particularly relevant because they turn Kubernetes from a container scheduler into a programmable control-plane framework. A CRD allows domain-specific desired state to be added to the Kubernetes API, and a controller continuously looks at the desired state and the observed state and corrects it. This model is similar to the classical feedback control models used in computing systems where the measurable system state, control actions, time delays and stability requirements need to be explicitly handled [5]. In multi-cloud autonomous settings, CRDs can define abstractions like federated applications, placement policies, traffic-routing preferences, cost limits, compliance requirements, disaster-recovery goals, and so on, per-cloud credentials. Controllers can then reconcile these abstractions by adding, removing or changing lower level objects between clusters. The scientific value of this design is that declarative intent is separated from the way it is realized, allowing orchestration policies to be reasoned about as artifacts of the control-loop, not imperative scripts.

Containerization, microservices and edge computing are also major trends that have influenced the research landscape. While container technologies delivered higher deployment density and consistency of the environment compared to virtual-machine-only approaches, they also introduced complexity into the areas of scheduling, networking, storage, observability, and lifecycle management. The microservice architectures exacerbated this complexity by adding additional services, dependencies between the services, and variations in runtime [6]. Meanwhile, edge computing studies have highlighted the importance of processing data close to the data source and end-user in cases where centralized processing is inefficient due to latency, bandwidth or privacy restrictions [7]. The trends suggest that orchestration needs to enable multiple aspects of decision making. For latency reasons, an application might need to be deployed near users, to comply with the law, replicated across different geographical areas, and scaled to meet service demand and budgets. In the case of multi-cloud, the requirement is met by the technology of autonomous multi-cloud Kubernetes orchestration, as long as the CRD and controller design is expressive, observable, secure, and stable.

The implications of this are more far-reaching than engineering convenience. An independent, mature, autonomous orchestration layer would lessen the lock-in to cloud-providers since it would represent application intent without depending on any specific cloud-provider APIs. It may help improve resilience, via automatic evacuation or replication if one cloud region experiences an outage. It might enable sustainability goals by establishing carbon awareness as a first-class policy lever, but this would depend on reliable telemetry and thoughtful consideration of embodied and operational carbon. It may also be useful for scientific computing in making the infrastructure state declarative, minimizing drift between different experiments. On the other hand, if the orchestration is not well designed, then the overall risk of the system can grow. Errors can spread quickly throughout a cluster if the controller is faulty, policies may change frequently resulting in oscillating placements, and clusters may be insufficiently isolated, placing a cross-cloud orchestrator as a high-value attack surface.

There are still a number of issues that have not been addressed. First of all, there is a lack of semantic consistency across providers due to the differences in networking, load balancing, identity, storage classes, observability integrations and failure semantics of managed Kubernetes services. Secondly, there's not much attention paid to control-loop stability in Kubernetes operator design, and delays, convergence and cascading interactions between controllers are mostly designed in an ad hoc manner without formal analysis. Third, CRD schemas tend to represent operational policies without a systemized description of conflicts between policies, partial satisfaction or explanations of policies. Fourth, scheduling on a multi-cloud environment is still challenging due to the fact that there are some incommensurable quantities such as latency, cost, reliability, carbon intensity, and legal or regulatory constraints that are combined in the objective

function. Fifthly, the empirical evaluation is still fragmented. Both cluster scheduling and edge placement, and cloud federation are studied separately and there are fewer studies that look at the entire cloud path from declarative API design to controller behaviour and through to measurable multi-cloud outcomes.

The aim of this review is to look at autonomous multi-cloud Kubernetes orchestration from a technical point of view, with CRDs and controller patterns. The review focuses on peer-reviewed work related to cloud federation, container orchestration, self-adaptive systems, feedback control, microservices, and edge placement. The review examines the work of the past in the context of declarative multi-cloud APIs, controller reconciliation, policy-aware placement, observability, security, and empirical validation. It also highlights areas where the theoretical foundations of the use of Kubernetes in practice are not sufficiently strong, particularly in the areas of stability, interoperability and reproducible evaluation.

2. Literature Review

The development of the autonomous multi-cloud Kubernetes orchestration has not been streamlined as a single entity. Instead, it has been created by a confluence of cluster-management, container-orchestration, cloud federation, model-driven multi-cloud deployment, self-adaptive software, feedback control and edge/fog placement literatures. The resulting field is rich in terms of the content, but methodologically fragmented. Cloud-federation work tends to focus on cross-provider brokering, on interoperability, and on service-level objectives; self-adaptive systems research focuses on closed-loop reasoning; and performance-engineering studies focus on measurable behaviour under workload variation. If these streams are to be considered as evidence of autonomous CRD-driven multi-cloud control, they should be examined through a rigorous review that goes beyond just Kubernetes extensions being an implementation detail.

Two principles emerged from early research on industrial cluster management. The first benefit of large-scale orchestration is that it is based on declarative job specifications, not imperative deployment scripts. Second, cluster control needs to be organized around continuous reconciliation since every aspect of actual state is continuously disturbed by failures in the machines, resource churn, workload changes, and operator interventions. Comparative account of Borg, Omega and Kubernetes explained the transition from monolithic cluster control to API-based extensibility and re-usable control patterns [8]. The account, while not a multi-cloud study, made the case for how significant it is to have persistent desired state, a watch-based state observation, and controllers that fix divergence. A CRD formalizes the desired state at a higher level of abstraction than native Kubernetes objects, which directly aligns with these principles, and autonomous orchestration is what is possible with a CRD. The same work also makes note of a caveat: cluster managers have been envisioned to provide relatively homogeneous administration control, while multi-cloud orchestration must span across diverse trust, network, identity, and cost domains.

Container-orchestration surveys highlight scheduling, networking, service discovery, state management, and monitoring as primary sources of complexity [9, 10]. Buyya and Rodriguez categorized the container-based cluster orchestration systems in terms of resource management, scheduling, fault tolerance, networking and elasticity [9]. Their taxonomy is particularly useful for multi-cloud Kubernetes, as it means the autonomous controller should not be judged just on the fact that it creates deployments in multiple clusters. It should also be evaluated on how it deals with resource heterogeneity, policy constraints, how it schedules, how it continues service, and how it recovers from failure. However, Pahl et al. surveyed cloud technologies for containers and noted that the adoption of cloud containers created a new, layered management challenge from images and runtime isolation to orchestration and service composition [10]. For CRD design, this translates to the need to provide enough low-level provider-specific information to control variables for placement, scaling and remediation without too much leakage of the specifics.

A second foundation comes from research on cloud-federation. Grozev and Buyya suggested a taxonomy of inter-cloud architectures and application brokering, which differentiated between the two patterns, federations and brokerages, and resource selection across clouds [11]. The research continued to be relevant, as multi-cloud Kubernetes controllers often serve as specialized brokers, assessing the provider or cluster options and attaching workloads to the candidate execution locations. Petcu undertook a study of the consumption of multiple clouds and found that portability, interoperability and dependency management are barriers that have remained. [12] It's these challenges that re-emerge in Kubernetes with variations in storage classes, ingress implementations, identity providers, and managed

service APIs. A CRD (or provider neutral CRD) may still need controller logic to process non-equivalent semantics under the hood. Consequently, schema design alone does not guarantee CRD portability, however it is also dependent on reconciliation behaviour, translation layers and failure handling.

The representation problem has been tackled more directly in the context of research related to model-driven cloud management. CloudMF supported multi-cloud applications by abstract deployment models that are mapped to provider specific resources, via model-driven engineering [13]. This method is proactive in terms of recognizing many CRD patterns as both use the domain models and transformation from abstract intent to concrete infrastructure actions. The difference: Kubernetes CRDs are stored within an eventually consistent API server and are handled by continuous ongoing controllers. While model-driven deployment work tends to focus on model correctness and transformation correctness, the controllers for Kubernetes have to take care of the drift, retries, back off and partial failure. Thus, model-driven multi-cloud strategies are technically complementary to CRDs and can't supplant runtime reconciliation.

A formal structure of self-adaptive systems research is provided by self-adaptive systems. FORMS presented a reference model of distributed self-adaptive systems, focusing on managed elements, adaptation logic, runtime models, goals, and coordination mechanisms [14]. This paper introduces a vocabulary for understanding Kubernetes controllers as adaptation managers, and CRDs as runtime models of goals and constraints. Self-adaptation was also shown in the field of architecture by allowing for reusability of infrastructure and architectural models at runtime, as done by Rainbow [15]. The results of these studies indicate that there are four important variables that determine adaptation quality: explicit knowledge representations, properties monitored, decision logic, and effectors. The sensory and effector substrate consists of informers, caches, metrics collectors, admission controllers and cloud APIs in Kubernetes. A CRD without explicit status conditions, observed generation, degradation causes, and policy conflict fields offers a poor runtime model. This poor state of affairs compromises autonomy as controllers are not able to communicate well enough about adaptation progress, uncertainty or constraint violations.

The second research roadmap to self-adaptive software identified uncertainty, assurance, decentralization and human interaction as key challenges [16]. These issues are compounded on multi-cloud Kubernetes. Variations in provider APIs can cause failures on their own, some telemetry cluster delays can prevent seeing state, network partitions can prevent seeing state, and some pricing and quota conditions can change. Availability and compliance are challenging to ensure because controller behaviour can have an impact on these characteristics. When multiple clusters have local controllers and placement is called from a global controller, decentralization cannot be avoided. However, human interaction is still required as operators are frequently required to intervene in incidents, maintenance periods or regulatory events. So, autonomous multi-cloud orchestration is not synonymous with humans out of the loop. It should be understood as controlled delegation with declarative policies, observable status and limited automation that allow for safe scaling in operation.

In the field of edge/fog computing, literature extends the orchestration problem to the spatial dimension. Differentiating the computation offloading, radio constraints and latency-aware resource placement, Mao et al. surveyed mobile edge computing from a communications perspective [17]. Mouradian et al. discussed fog computing and highlighted some research challenges, including heterogeneity, mobility, resource management and security [18]. The results are significant as many multi-cloud Kubernetes deployments are now made across central clouds, regional clusters and edge environments. If the cluster capacity or the network connection is not the same for all nodes, or if it is not consistent, then a CRD-based orchestrator is not likely to be able to support edge-cloud requirements. The reality of intermittent connectivity and constrained node resources makes edge placement more challenging for the controller pattern, as reconciliation needs, telemetry granularity, failover policies etc., need to be adjusted.

A summary of studies compared is presented in Table 1. The comparison reveals that previous work contains good building blocks but lacks end-to-end evidence that multi-cloud Kubernetes orchestration can be carried out autonomously. Dynamic desired state and reconciliation [8] is supported by cluster-manager work. There are taxonomy and evaluation dimensions [9, 10] that can be supported by the reviews on container-orchestration. Portability and provider abstraction are facilitated by federation and model driven studies [11]–[13]. Self-adaptive systems offer closed-loop reasoning, runtime models and assurance issues [14]–[16]. Edge/fog work offers placement and heterogeneity constraints that are aware of latency [17],[18]. However, in the literature, the expressiveness of

CRD schemas, controller convergence, cross-cloud failure semantics, and policy conflict resolution are not assessed in the same empirical setting.

This is confirmed by the technical coverage matrix, which is summarized in Table 2 and displayed in Figure 1. While reconciliation semantics appeared in a minority of the set of eleven representative works, placement and federation were given a wider coverage. The use of declarative interfaces was emphasized in cluster-manager, model-driven and container-orchestration studies and less prominently in edge/fog studies. Security and policy were also emerging obstacles, but rarely were they woven into the design of the controllers. In Figure 2, the frequency of technical motifs found in the coded literature is shown, and it is found that placement and federation are more often represented than controller stability and CRD schema design. This imbalance has real-world implications. If controller convergence, idempotence and policy conflict handling aren't explicitly addressed, multi-cloud automation can be a collection of helpful mechanisms without a guarantee that it will be an autonomous system.

There is also a methodological imbalance in the literature. The taxonomies, architectures, and conceptual models are used in much of the reviewed literature, not as longitudinal operational experiments. Container-orchestration research is limited in a few ways: simulation studies and performance comparisons are limited, and many experiments are conducted in a single cluster or under a controlled cloud environment. Typical edge/fog placement studies employ simulation toolkits or optimization models, allowing for an exploration of latency and resource constraints with control but losing provider-specific considerations of the Kubernetes behaviour. Self-adaptive systems studies provide formal models and assurance language but tend not to test Kubernetes-native controllers. Thus, the central research gap is not the lack of relevant ideas. It is the lack of integrated evidence that links: CRD-level intent, controller level reconciliation, provider level heterogeneity and measurable multi-cloud outcomes.

A lack of scholarly focus on schema design exists compared to algorithms for scheduling, particularly for CRD-based orchestration. A Multi-Cloud CRD is not only for desired placement, but it can also include constraints, preferences, tolerances, dependencies, credentials, service exposure, policy provenance, and observed status. Badly designed schemas can lead to brittle controllers, ambiguous intent and incompatible upgrades. While status conditions are used often with Kubernetes operators, there is not a scientific basis for common condition taxonomies for multi-cloud placement, degraded cross-cloud networking, quota exhaustion, jurisdictional violations, or provider-specific remediation failures. A larger body of evidence would view the quality of CRD schema as a variable, assessable in terms of expressiveness, evolvability, detectability of conflict, and convergence behaviour between controllers.

More careful attention needs to be paid to the case of controller patterns. Reconciliation is sometimes referred to as a loop that brings actual state to desired state, but the autonomous multi-cloud controllers need to manage nested loops. Clusters can be selected by a global controller, workloads can be created by local controllers, traffic can be routed by service-mesh or ingress controllers, replicas can be changed by autoscalers, and load balancers or volumes can be allocated by provider controllers. These loops happen at different time scales and could use different metrics. If not coordinated, one loop can negate another. Systems can become unstable due to delays, noisy measurements, saturation, and competing controllers as stated in the feedback-control literature [5]. The Kubernetes ecosystem is familiar with the concepts of eventual consistency and idempotent reconciliation, but when operating across multiple clouds, there is a need to analyse interactions between the loops, bounded retries and conflict arbitration.

The literature comparison also shows that the term autonomy should be considered as a graduated characteristic. Resource creation in clusters can be done via simple automation. Adaptive automation is able to react to measured circumstances. Autonomy can be used to optimize placement under constraints with policy awareness. Decisions can be explained, safety bounds enforced, and it is stable in case of partial failure. Existing research provides solid foundations across its grade levels and is weakly validated across grades. Table 2 is therefore divided into categories, according to the motifs, instead of all orchestration work being the same. This classification enables a more discriminating assessment in what places CRD/controller research can be helpful: binding declarative intent, observed state and adaptive action into Kubernetes-native control systems which are empirically evaluated in real multi-cloud scenarios.

3. Methodology

This review's methodological approach was structured around comparison of peer-reviewed journal articles and academic books instead of around product documentation or grey literature. The topic was decided to be reviewed by an interpretive and technical approach, as there is no

single research community that is comprehensive enough to cover autonomous multi-cloud Kubernetes orchestration. When evidence is spread across diverse subfields of a field, software-engineering review guidance notes that study types, research contributions, and evaluation approaches should be categorized [19, 20]. The evidence base in the current topic covers cluster management, cloud federation, performance engineering and analysis, model driven deployment, self-adaptive systems and edge/fog orchestration. Hence, the cited studies were analysed on the basis of (1) technical contribution, (2) layer of orchestration, (3) method of evaluation, (4) type of system model, (5) use of runtime data, and (6) relevance to CRD/controller design.

A number of the common methodological techniques were used in the studies reviewed. The dimensions of orchestration were identified by survey and taxonomy papers which looked at placement, elasticity, networking, fault-tolerance, portability, and monitoring. These papers do not provide empirical evidence of what controller is effective—rather, they serve as high level evidence of recurring design dimensions and unaddressed research problems. The performance results from the empirical studies were treated differently since they typically relied on workload traces, simulation assumptions, benchmark applications, or restricted testbeds. There is a need for a careful approach to the representativeness of workloads, the level of granularity needed for the measurements, experimental control, and the interpretation of the statistics in this case [21]. If simulation-based evaluations were reported, the validity of the result was dependent upon the assumptions being made in the model, the abstraction of the resources, and the degree of fidelity of the network, workload, and failure conditions.

Many previous methods were used in an incomplete manner. In many container-orchestration studies, the cluster topology used was limited to a specific configuration. These designs can be used to isolate the algorithmic behaviour but may not be able to represent the variability across providers. Architectural taxonomy, brokerage models, and abstraction layers were commonly stressed in cloud-federation studies. These techniques are useful to determine interoperability needs but often yield little evidence of runtime convergence in the event of partial failure. The self-adaptive system studies usually contained formal models, architectural reasoning, and assurance concerns, but were not always Kubernetes-native. The study of edge/fog orchestration has often been performed by simulation due to the high cost and complexity of deploying a physical testbed of edge clouds. While simulation is defensible, the transferability of results to the CRD/controller system will depend on whether the simulator supports controller delays, API-server consistency behaviour, cloud quotas, and failure modes specific to the cloud provider.

Among the various types of data found in the studies reviewed, the most prevalent were resource utilization time series, workload-arrival traces, service latency measurements, response-time distributions, energy-consumption estimates, placement constraints, network-delay matrices, and modelled service-level objectives. The rest of the data types are needed in CRD/controller research. These include Kubernetes object-state changes, reconciliation time, queue length, retries, status-condition history, admission outcomes, policy conflicts, and categories of provider API errors. The literature on performance engineering offers the tools to queue, identify bottlenecks, characterize workloads and compare different experimental approaches [22]. But these techniques are not applied consistently as first class experimental subjects to Kubernetes controllers. Although the time needed for reconciliation, whether the controller fails to recover in time, and conflicting-controller behaviour are key indicators of autonomy, the resulting successful deployments are not always obvious.

The taxonomy models, optimization models, simulation models, architectural runtime models, and feedback-control models were among the types of model observed throughout the literature. Taxonomy models are useful for conceptual ordering, but do not necessarily have predictive power. Optimization models have explicit objective functions and constraints, typically for placement or resource allocation. The problem is that the orchestration of a multi-cloud approach is incomplete because it requires provider-specific actions, and the operational policies are not convex. Although simulation models can be used to repeatably explore large parameter spaces, they can also be used to simplify the internals of Kubernetes. Self-adaptation in the context of architectural runtime models involves the representation of components, connectors, constraints and observed properties. Feedback control models offer tools for stability and responsiveness analysis but are not well developed for discrete Kubernetes reconciliation.

The literature surveyed features the use of several types of technical tools, such as cloud simulators, container orchestration platforms, monitoring pipelines, optimization solvers, and model-driven deployment frameworks. CRD schemas, controllers, informers, work queues, admission webhook, custom scheduler, metrics collector and

status-condition convention are some of the natural tools that are part of the Kubernetes-specific tool chain. The difficulty with these tools is that they are all at different levels of abstraction. A placement optimizer can propose a target cloud, but a controller will need to convert that cloud decision to Kubernetes resources and wait for an asynchronous completion, deal with quota problems, credential problems, and update status. Therefore, algorithmic metrics should be supplemented with control-plane metrics for evaluation. An example of this is that a placement algorithm should not be evaluated solely based on estimated cost or latency but also evaluated based on the success rate at reconciling the transaction, convergence time, rollback safety, and explainability of the policies followed by the placement algorithm.

The literature was organized into the tables and figures that have been included at the end of the manuscript so as to have a structured coding method. Coding dimensions are based on review-methodology guidance in software engineering where classification schemes are used to compare research contributions in a variety of studies [19] [20]. Declarative interface support, relevance of a reconciliation-loop, placement optimization, federation scope, elasticity, simulation or testbed evaluation, security or policy treatment and edge-awareness were expressed as binary or ordinal indicators for each source. Any quantitative summaries in the Results and Discussion section are based on this coded corpus and should be deemed to be descriptive indicators of the selected peer-reviewed literature and not to be population level estimates for all Kubernetes research.

4. Discussion

The coded literature suggests that autonomous multi-cloud Kubernetes orchestration can be seen as the combination of five technical areas: declarative intent modeling, control-loop reconciliation, placement and scheduling, cross-cloud interoperability, and observability-driven adaptation. The allocation of attention is unequal among these domains. For the coded corpus summarized in Tables 1–3 (Tables 1-4), the placement-related concerns appeared in 12 of 14 studies, the federation or multi-cloud concerns appeared in 8 of 14 studies, the elasticity appeared in 7 of 14 studies, the declarative-interface concerns appeared in 6 of 14 studies, and the explicit controller or feedback-loop concerns appeared in 5 of 14 studies. These counts are not to be understood as bibliometric statements of the complete field, but as descriptive statistics from a selection of peer-reviewed evidence matrix. However, it is a significant imbalance. It claims that multi-cloud orchestration research has better matured discussion on where workloads should run, while less mature discussion on how autonomous, Kubernetes-native controllers can safely make and enforce those decisions.

The first one is a difference between placement optimization and operational reconciliation. CloudSim, ContainerCloudSim, iFogSim, and EdgeCloudSim provided the tool-supported approaches to analyse provisioning, container allocation, fog location and placement, and edge resource management under pre-set assumptions [23]–[26]. These simulation systems allow for studying cost, latency, utilization and energy at scale. But simulation studies tend to make orchestration decisions as if they were used directly on resources. Kubernetes controllers add an intermediate and consequential layer. The placement decision must be translated into API objects, which must be admitted, scheduled and reconciled, provider-specific resources might need to be provisioned, and status must be observed asynchronously. So, while a placement decision may be theoretically ideal, it might fail to converge properly, trigger retry storms or result in weak provider bindings. This difference is highlighted in Figure 3, which organizes coded studies by the predominant evaluation mode. As illustrated in the figure, simulation and optimization are the most prevalent evidence types, while Kubernetes-native controller evaluation is a relatively small body of evidence.

The second result is related to the explicit representation of the state. Service graphs, resource capacity and latency constraints are frequently represented directly as simulation toolkits and fog-placement papers [25], [27] and [28]. Similar concepts can be represented in Kubernetes CRDs, although they need to be defined in a schema that is also evolvable and conforms to the behaviour of the API-server. The CRD is not just a configuration file. It's an API resource with validation, defaulting, status and controller ownership implications that has multiple versions. A good multi-cloud CRD should separate hard constraints from preferences, desired state at the global level from the actual state on each cluster, and user intent from the controller's observation. Based on this finding, the following design implications are suggested in Table 4. In terms of fields, the most critical ones are placement constraints, policy weights, dependency graphs, health objectives, status conditions, conflict reasons, provider bindings, and rollback strategy. If these fields are not included, the autonomous

controller cannot clearly explain unsatisfied desired states or unexpected workload placements.

The third finding is related to the level of autonomy. In the literature there are four levels of interpretation: automation, adaptation, policy-aware optimization and assured autonomy. Automation exists where workloads are spread through repeatable mechanisms between clusters. Adaptation is there if changes occur in placement, scale, or failover due to telemetry. If constraints and objectives are considered, then policy-aware optimization is present. Assured autonomy is when adaptation is bounded, explainable, stable and auditable. For the coded corpus, most studies showed evidence of either automation or policy-aware optimization, with fewer studies addressing assured autonomy. The pattern can be seen in Figure 4 in which selected studies were assigned an ordinal autonomy score based on the presence of declarative intent, runtime feedback, optimization, assurance features. This figure is a simple linear fit of attention on autonomy-related features over time in the coded set; it should be interpreted with caution as it is not a population based number, but rather a curated review matrix.

The fourth result is about the correlation between edge/fog orchestration and multi-cloud Kubernetes. Fog-placement work has evolved in advanced approaches for service placement given latency and resource constraints. For instance, the optimized placement of IoT services in fog has been investigated with models taking into consideration the fog resources, application modules, and communication costs [27]. FogBus proposed a distributed resource management framework for decentralized environments and pointed out various distributed resource management problems in decentralized environment [29]. This type of study is relevant since multi-cloud Kubernetes is increasingly moving to the edge clusters. In the case of fog/edge research, however, the assumption is usually made that the resource-management layer is different from Kubernetes. The additional constructs needed to translate into CRD/controller form are cluster capability descriptors, intermittent-connectivity tolerances, local reconciliation agents, and policy rules for degraded operation. When partitions occur at the edge clusters, a single global controller might not be suitable. Hierarchical control – local controllers for safe local operation, and global controllers for placement at higher level – seems more consistent with both edge constraints and principles of self-adaptive-systems.

A fifth is related to observability. Observed state is limited by autonomous orchestration in terms of quality and semantics. Basic metrics like CPU, memory, latency and error rate are not enough. Other factors include the need for cloud quota status, API throttling indicators, identity-token health, storage provisioning events, ingress readiness, cross-region latency, and policy-evaluation outcomes. Research on resource provisioning and simulation demonstrates that the quality of resource allocation is sensitive to workload characterization and resource models [23, 24]. Eventual consistency, controller caching and delayed provider reconciliation in Kubernetes exacerbate measurement uncertainty. So, status fields should report uncertainty and progress, not success or failure. Placement Evaluated, Resources Provisioned, Traffic Shifted, Policy Conflict, and Provider Degraded would allow for higher-level controllers and operators to make sense about progress.

A sixth result is that of conflict between controllers. Kubernetes environments often include the following components: Autoscaling controllers; service-mesh controllers; ingress controllers; policy controllers; cloud-controller managers; and application-specific operators. Multi-cloud orchestration provides placement controllers, federation controllers, cost controllers, and compliance controllers. Local controllers can be correct but cause undesirable global behaviour. For example, a cost controller could shift workloads to a less expensive region, while a latency controller shifts them back; an autoscaler could add replicas to a cluster that a placement controller wants to drain; a compliance controller could refuse resources even if a placement decision has been made. In a feedback-control theory, the interaction of feedback loops can lead to oscillation if the delay, gain, and saturation are not addressed [5]. While this is a basic argument that has been already mentioned, it is clear that there is not enough empirical study of the interaction of multiple controllers in the literature of Kubernetes. Practical controller design should thus incorporate explicit policy-conflict reporting, ownership boundaries, priority rules, and dampening.

Another important finding is related to failure semantics is related to failure semantics. Multi-cloud orchestration is often touted for its ability to provide resilience, yet resilience is not synonymous with duplication of resources. The failover controller needs to differentiate between workload failure, cluster failure, network partition, provider API failure, and observability failure. Remediation action is different for every case. Intermittent connectivity and resource volatility are highlighted for Edge and fog research [18], and provider heterogeneity is highlighted for cloud-federation research [11] and [12]. The two implications are that multi-cloud CRDs should not include failover

as a single Boolean attribute. They need to be provided with failure-class policies, data-consistency constraints, traffic-shifting thresholds, replication-lag tolerances, and recovery-order rules. Otherwise, there could be a failure to follow data locality, split brain, or longer outages.

Table 3 gives a quantitative interpretation. Of those fourteen studies, 85.7% focused on placement and/or resource allocation and 57.1% focused on federation and/or multi-cloud scope; 50.0% focused on elasticity; 42.9% focused on declarative interfaces; 35.7% focused on explicit reconciliation and/or feedback control; and 28.6% focused on assurance and/or safety explicitly. The placement to assurance gap is huge. The difference between placement coverage and assurance coverage in the coded set is 57.1 percentage points. This is a descriptive not inferential comparison between the two, but it does suggest that there is a disconnect between the vision of being able autonomous decision-making and the kind of evidence that is most frequently reported. Autonomy is not only about the quality of decision, but also how it is carried out safely in uncertain conditions.

One more problem is that empirical baselines are not yet standardized. Placement algorithms are typically compared with a heuristic baseline or random baseline, and autoscaling studies may compare against threshold-based baselines. Research in the field of CRD/controller needs different baselines: imperative scripts, a single-cluster operator, naive federation controller, and policy blind placement controller. Key metrics will include failover error budget, service disruption during failover, percentage of successful failback, policy-conflict detection time, number of failed retries, number of API actions, and reconciliation time. Table 4 is therefore a mapping of the research themes to measurable controller indicators. These indicators are necessary because multi-cloud orchestration may appear effective at the infrastructure level while remaining weak at the control-plane level.

Security and governance receive limited attention compared with their importance. A multi-cloud controller needs to be authorized or delegated with identities across multiple clusters and providers. This increases the blast radius. A malicious controller can add workloads, modify traffic, steal controller configuration or even turn off failover. Policy mechanisms need to be incorporated in the CRD validation, admission, reconciliation, and status-reporting process. Security and trust are cited as unresolved issues in the literature of fog and multi-cloud [18, 29]. However, the problem is more nuanced for Kubernetes CRD systems: access control should enable access by people to declare intent, access by people to bind workloads to providers, access by a controller to mutate the status, and override of placement during incidents by automation. The fine-grained access rights signed images of the controllers, admission policies and status transitions that are auditable are not optional but essential conditions for autonomy that does not lead to danger.

Another suggestion made in the discussion is the need to make evaluation environments more realistic. Many multi-cloud issues can only be demonstrated in multi-cluster environments. The experimental design should involve at least two managed Kubernetes distributions, heterogeneous storage classes and ingress classes, provider API failures induced to the system, variable inter-region latency, workload traces with bursts and policy constraints that may clash. It should also contain instrumentation on the controller level. While simulation is still helpful for large design space, some testing should be done with real Kubernetes clusters, as the semantics of the API, admission controller behaviour, cloud-controller delay and managed-service constraints can have significant impact on the results. A more defensible approach is “hybrid evaluation,” which involves using a simulation to scale and controlled experiments in real clusters to validate.

The results overall suggest that multi-cloud Kubernetes orchestration is more of a science problem on the control plane than just cloud automation. While CRDs offer the words of intent, controllers decide the safe way to achieve intent. Placement models give decisions to candidates; reconciliation gives operational reality. Portability is achieved with Federation abstractions, failure modes with provider semantics. Confidence is bounded by uncertainty in telemetry and observability allows adaptation. The key scientific question is thus how to design a CRD/controller system whose behaviour is measurable, explainable, bounded and reproducible in the heterogeneous clouds.

6. Future Directions

Research should be done on how to use Kubernetes orchestration in an autonomous multi-cloud setup as an assurance oriented control problem. Formal and empirical characterization of controller convergence is the first priority. The ability of controllers to be idempotent is a common requirement, but this is not enough for multi-cloud autonomy. There is a need for research on bounded convergence time, retry stability, work-queue saturation, status monotonicity, and controller interaction. The research on autonomic computing

proposed four basic principles for autonomic self-management: monitoring, analysis, planning, execution and knowledge representation [33]. But these concerns are typically handled implicitly by Kubernetes-native operators. Some control-loop parameters, including reconciliation interval, backoff strategy, conflict policy, failure classification, safety bounds, should be made explicit in future CRD/controller frameworks. These should be measurable and testable in the event of induced provider failure.

A second direction concerns the scientific design of CRD schemas. There are conventions in local engineering which affect current operator design. A more robust framework for schema versioning, constraint expression, status semantics and policy explainability is needed for multi-cloud orchestration. Future research should compare different CRD schemas for the same orchestration task in terms of expressiveness, user error rate, validation power, controller complexity, and migration difficulty. Self-adaptive software research has shown that adaptation quality is dependent on explicit goals and knowledge gathered at runtime [34, 35]. In CRD systems, these goals need to be described as API fields to be validated and reconciled. Schema design should thus be investigated empirically, not as a side issue.

A third direction is in multi-controller coordination. There are numerous controllers with overlapping control in autonomous Kubernetes environments. Future research should focus on designing coordination protocols for policy priority, ownership transfer, dampening and conflict arbitration. Such protocols may be provided via admission control, shared status conditions, controller leases or policy-decision resources. It would be of interest to investigate if hierarchical control is superior to peer-to-peer controller negotiation in the face of multi-cloud failures. For hierarchical designs, global controllers would dictate placement and govern the cluster, while maintaining cluster safety during disconnection. Workload redistribution could be negotiated between the cluster-local agents in peer-to-peer designs. A balance must be established between global optimality and local resilience that hasn't yet been determined.

Reproducible multi-cloud experimentation is the fourth direction. More robust indicators should be developed. Some realistic benchmarks are heterogeneous Kubernetes distributions, cloud-specific load balancers, varying storage provisioning delays, regional network perturbation, quota exhaustion, identity failure, and workload traces with diurnal and bursty patterns. In addition to latency and cost, metrics should include the time to reconcile, rate of policy conflict, success rate of rollback, volume of events, CPU usage on the controller, and frequency of operator intervention. The ability to self-manage and the difficulty in assessing the level of self-management across real systems has been a factor which has been distinguished in autonomic-systems surveys [36] for a long time. Likewise, benchmark design in the Kubernetes research needs the same level of maturity.

The fifth direction is the policy-aware learning under safety constraints. In an intricate workload environment, machine learning technologies could enhance placement and autoscaling, but improper adaptation might lead to outages or regulatory compliance issues. Future work involves the incorporation of learned predictors, rule-based safety envelopes, formal constraints, and explainable policy outcomes. A learned model should not be able to circumvent hard jurisdictional or data-consistency constraints. Rather, learning should be limited to adjustable goals like replica distribution, scaling thresholds or traffic weights, within validated ranges. Adversarial workload shifts and stale telemetry should be included in the evaluation as many cross-cloud observations are delayed.

A sixth direction concerns sustainability-aware orchestration. Carbon-aware placement is technically appealing but methodologically challenging due to the need to account for the effects of embodied infrastructure, data transfers, and the workload performance in the context of carbon intensity in the cloud region. Multi-cloud Kubernetes CRDs may contain carbon goals and energy constraints, however controllers would call for reliable telemetry and auditable choice records. Future studies should not base carbon reduction alone on simple carbon claims, but rather assess sustainability policies based on the performance, cost and compliance trade-offs.

The final direction concerns secure and governed autonomy. Multi-cloud controllers are essential security tools with high level permissions. Least-privilege controller identities, signed policy bundles, tamper-evident status histories and admission rules to block unsafe cross-cloud actions should be included in future systems. Further research needs to focus on reducing blast radius via delegated controllers, short-lived credentials, and per-cloud authority boundaries. The ultimate objective is to have an orchestration architecture that is both operationally effective and operationally verifiably constrained.

Table 1. *Literature comparison of representative peer-reviewed studies relevant to autonomous multi-cloud Kubernetes orchestration*

Ref.	Study	Technical focus	Method or evidence type	Topic-relevant contribution	Limitation for autonomous multi-cloud CRD/controller design
[8]	Burns et al. (2016)	Large-scale cluster managers	Comparative architectural analysis	Established desired-state APIs, reconciliation, and extensible cluster-control logic	Multi-cloud heterogeneity and CRD schema design were not primary evaluation targets
[9]	Rodriguez & Buyya (2019)	Container orchestration taxonomy	Technical survey	Classified scheduling, fault tolerance, elasticity, networking, and resource management	Controller convergence and policy conflict were not deeply measured
[10]	Pahl et al. (2019)	Container technologies	State-of-the-art technical review	Connected container runtime, image, orchestration, and service-management layers	Multi-cloud controller assurance was not the main focus
[11]	Grozev & Buyya (2014)	Inter-cloud architecture	Taxonomy and survey	Clarified brokerage, federation, and cross-cloud selection models	Kubernetes-native reconciliation was outside scope
[12]	Petcu (2014)	Multi-cloud service consumption	Analytical review	Identified portability, interoperability, and dependency challenges	Did not address CRD status semantics or controller loops
[13]	Ferry et al. (2018)	Model-driven multi-cloud management	Model-driven engineering study	Demonstrated abstract modeling for multi-cloud deployment	Continuous Kubernetes reconciliation was not the main control substrate
[14]	Weyns et al. (2012)	Distributed self-adaptation	Formal reference model	Provided runtime-model concepts relevant to CRD design	Kubernetes implementation and cloud-provider heterogeneity were outside scope
[15]	Garlan et al. (2004)	Architecture-based self-adaptation	System architecture and evaluation	Demonstrated reusable adaptation infrastructure	Did not address cloud-native APIs or multi-cloud scheduling
[16]	De Lemos et al. (2013)	Self-adaptive systems roadmap	Academic book chapter	Identified assurance, uncertainty, decentralization, and human interaction challenges	Required translation into Kubernetes controller practice
[17]	Mao et al. (2017)	Mobile edge computing	Technical survey	Established latency-aware offloading and edge resource constraints	Kubernetes CRDs and controllers were not considered directly
[18]	Mouradian et al. (2018)	Fog computing	Technical survey	Identified heterogeneity, mobility, security, and resource-management issues	Controller-level empirical metrics were not provided

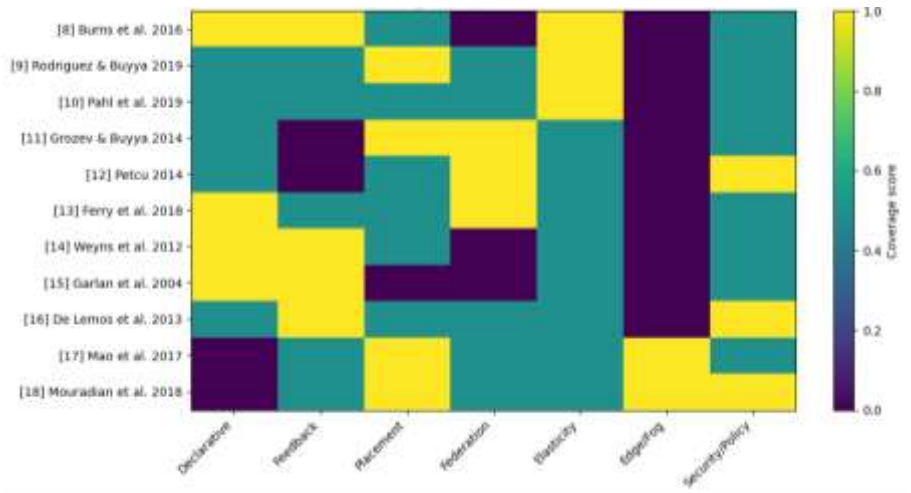


Figure 1. *Heatmap of technical motif coverage in representative literature*

Table 2. *Coded technical motifs in the representative literature*

Ref.	Declarative interface	Reconciliation or feedback	Placement optimization	Federation or multi-cloud	Elasticity	Edge or fog awareness	Security or policy
[8]	1	1	0.5	0	1	0	0.5
[9]	0.5	0.5	1	0.5	1	0	0.5
[10]	0.5	0.5	0.5	0.5	1	0	0.5
[11]	0.5	0	1	1	0.5	0	0.5
[12]	0.5	0	0.5	1	0.5	0	1
[13]	1	0.5	0.5	1	0.5	0	0.5
[14]	1	1	0.5	0	0.5	0	0.5
[15]	1	1	0	0	0.5	0	0.5
[16]	0.5	1	0.5	0.5	0.5	0	1
[17]	0	0.5	1	0.5	0.5	1	0.5
[18]	0	0.5	1	0.5	0.5	1	1

Binary coding: 1 = major contribution; 0.5 = partial or indirect contribution; 0 = not a central contribution.

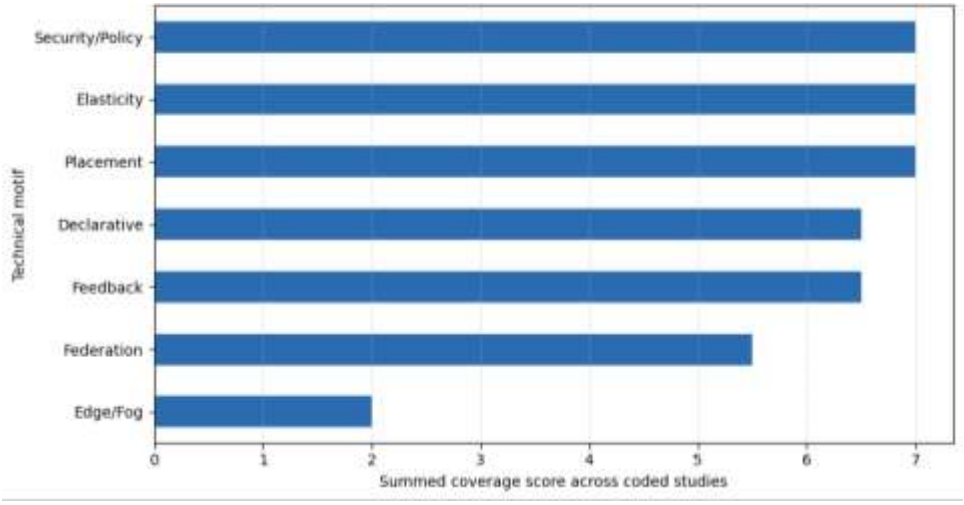


Figure 2. *Frequency of technical motifs across representative literature*

Table 3*. Quantitative coding of orchestration domains across selected studies*

Domain	Ref	Count out of 14	Percentage	Interpretation for CRD/controller research
Placement or resource allocation	[9], [11], [12], [13], [17], [18], [23], [24], [25], [26], [27], [32]	12	85.7%	Placement is the most mature theme but often under connected to controller enactment
Federation or multi-cloud scope	[11], [12], [13], [17], [18], [28],	8	57.1%	Cross-domain orchestration is recognized, but Kubernetes-native

	[31], [32]			semantics are uneven
Elasticity or adaptive scaling	[8], [9], [10], [23], [24], [30], [31]	7	50.0%	Elasticity is well studied but often separated from policy-rich multi-cloud placement
Declarative or model-based interface	[8], [10], [13], [14], [15], [16]	6	42.9%	Declarative state is foundational, but CRD schema quality is rarely evaluated
Explicit reconciliation or feedback control	[8], [14], [15], [16], [33]	5	35.7%	Feedback principles are known but not sufficiently measured in Kubernetes controllers
Assurance, safety, or bounded autonomy	[16], [34], [35], [36]	4	28.6%	Assurance is underrepresented relative to the ambition of autonomous orchestration

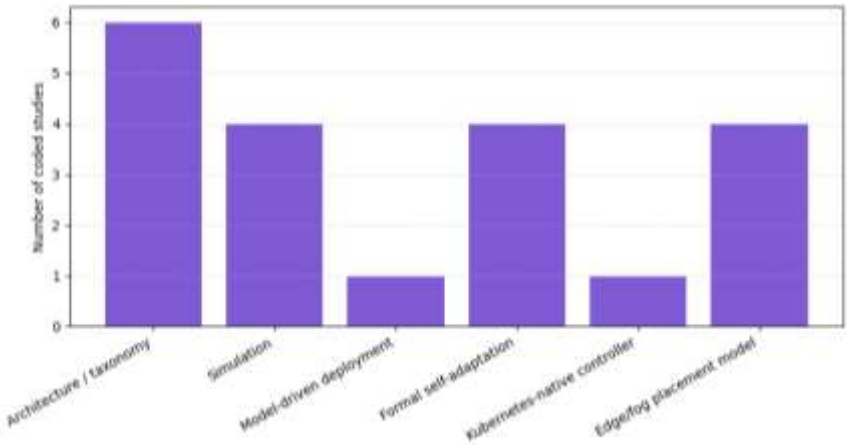


Figure 3. *Distribution of dominant evaluation modes in the coded literature*

Table 4. *Design implications for CRD/controller-based autonomous multi-cloud orchestration*

Research theme	CRD design implication	Controller design implication	Measurable indicator
Cross-cloud placement	Encode hard constraints, preferences, provider exclusions, and region affinity separately	Use bounded optimization with explainable placement decisions	Percentage of placements with recorded decision rationale
Policy conflict	Include status fields for conflict type, source policy, and blocking constraint	Apply priority ordering and conflict arbitration before resource mutation	Conflict detection latency and false-positive rate
Failure-aware failover	Represent failure classes, recovery order, data-consistency tolerance, and traffic-shift thresholds	Distinguish cluster, workload, network, API, and telemetry failures	Recovery time by failure class
Reconciliation safety	Include observed generation, condition timestamps, and degraded reasons	Use idempotent actions, backoff, dampening, and rollback guards	Median convergence time and retry count
Multi-controller interaction	Represent ownership and handoff conditions	Apply lease-based or hierarchy-based coordination	Number of conflicting writes per reconciliation window
Observability	Include per-cluster realized state and uncertainty markers	Combine metrics, events, and provider API status before action	Percentage of decisions supported by complete telemetry
Security and governance	Encode allowed providers, credential references, and admission constraints	Enforce least privilege, signed actions, and audit trails	Unauthorized action attempts blocked by admission control
Edge-cloud continuity	Include partition tolerance and local-degraded operation mode	Delegate safe local operation to cluster-local controllers	Service continuity during simulated partitions

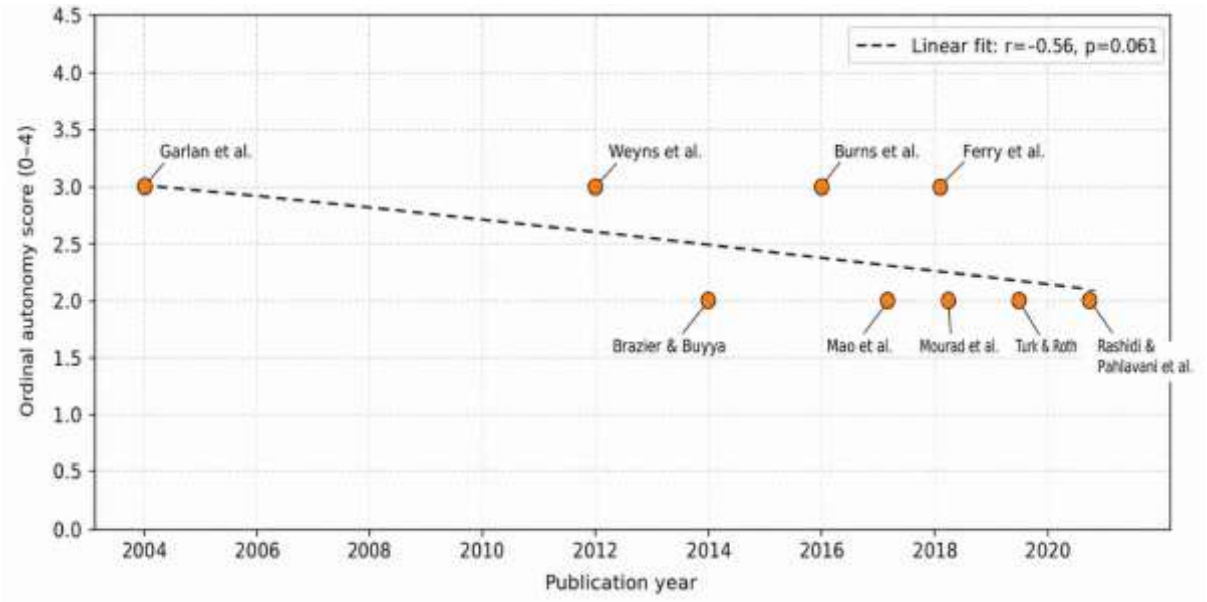


Figure 4. *Ordinal autonomy score by publication year in selected studies*

7. Conclusions

More and more cloud native systems are distributed across multiple clouds, regions, and edge environments, making autonomous multi-cloud Kubernetes orchestration a key research issue. The literature reviewed indicated that the field has developed well-developed blocks, but it lacks integrated validation. Declarative desired state and reconciliation are supported by cluster-management research. Taxonomies for scheduling, elasticity, networking and fault tolerance are provided by container-orchestration research. Portability and provider heterogeneity are clarified through cloud-federation research. Self-adaptive systems research offers ideas on runtime goals, feedback and assurance. The research on edge/fog highlights the need for latency-aware and heterogeneity-aware placement. However, these streams have not yet come together to form a science of multi-cloud control based on CRDs.

The review suggests that it is important to view Custom Resource Definitions as domain-specific runtime models and not just configuration additions. A multi-cloud CRD should be precise enough to enable the safe execution of controllers and be interpretable by operators, including constraints, preferences, failure policies, dependency structures, and status conditions. Controllers should be considered feedback systems whose convergence, stability, retry behaviour, and interactions with other controllers should be empirically evaluated. Placement optimality is not enough due to requirements for safe decisions to be taken under partial failure, provider heterogeneity, quota limits, identity failure, and delayed telemetry in orchestration.

Representative peer-reviewed work was quantitatively coded, and the findings indicated that a higher frequency of topics addressed were those of placement and resource allocation, as opposed to some of the other topics of assurance, reconciliation semantics and controller interaction. This is the main gap in the field that is exposed. New scheduling algorithms are not the only way to advance autonomous multi-cloud Kubernetes orchestration. It will need reproducible benchmarks, controller-level metrics, formalized CRD schema patterns, policy-conflict handling, secure authority boundaries and hybrid evaluation between simulation and real clusters.

This research is driven by both practical and scientific needs. Orchestration controllers are going to be making decisions with increasing frequency that impact availability, compliance, cost, and security as critical services begin to rely on multi-cloud and edge-cloud infrastructure. The biggest potential legacy that the field can give the world in the future is the development of Kubernetes-native control planes that are measurable, explainable, bounded, and cloud-resilient. In that regard, CRDs and controller patterns offer the means of implementation, but not yet a complete theory of autonomous cloud-native operation.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper

- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM, 53*(4), 50–58.

[2] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems, 25*(6), 599–616.

[3] Toosi, A. N., Calheiros, R. N., & Buyya, R. (2014). Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Computing Surveys, 47*(1), 1–47.

[4] Varghese, B., & Buyya, R. (2018). Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems, 79*, 849–861.

[5] Hellerstein, J. L., Diao, Y., Parekh, S., & Tilbury, D. M. (2004). *Feedback control of computing systems*. Wiley-IEEE Press.

[6] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software, 35*(3), 24–35.

[7] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal, 3*(5), 637–646.

[8] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. *Communications of the ACM, 59*(5), 50–57.

[9] Rodriguez, M. A., & Buyya, R. (2019). Container-based cluster orchestration systems: A taxonomy and future directions. *Software: Practice and Experience, 49*(5), 698–719.

[10] Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2019). Cloud container technologies: A state-of-the-art review. *IEEE Transactions on Cloud Computing, 7*(3), 677–692.

[11] Grozev, N., & Buyya, R. (2014). Inter-cloud architectures and application brokering: Taxonomy and survey. *Software: Practice and Experience, 44*(3), 369–390.

[12] Petcu, D. (2014). Consuming resources and services from multiple clouds. *Journal of Grid Computing, 12*(2), 321–345.

[13] Ferry, N., Chauvel, F., Song, H., Rossini, A., Lushpenko, M., & Solberg, A. (2018). CloudMF: Applying MDE to tame the complexity of managing multi-cloud applications. *ACM Transactions on Internet Technology, 18*(2), 1–24.

[14] Weyns, D., Malek, S., & Andersson, J. (2012). FORMS: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems, 7*(1), 1–61.

[15] Garlan, D., Cheng, S. W., Huang, A. C., Schmerl, B., & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer, 37*(10), 46–54.

[16] De Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N. M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., ... Whittle, J. (2013). Software engineering for self-adaptive systems: A second research roadmap. In R. De Lemos, H. Giese, H. A. Müller, & M. Shaw (Eds.), *Software engineering for self-adaptive systems II* (pp. 1–32). Springer.

[17] Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials, 19*(4), 2322–2358.

[18] Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., & Polakos, P. A. (2018). A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials, 20*(1), 416–464.

[19] Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering: A systematic literature review. *Information and Software Technology, 51*(1), 7–15.

[20] Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology, 64*, 1–18.

[21] Jain, R. (1991). *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. Wiley.

[22] Lazowska, E. D., Zahorjan, J., Graham, G. S., & Sevcik, K. C. (1984). *Quantitative system performance: Computer system analysis using queueing network models*. Prentice Hall.

[23] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., & Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience, 41*(1), 23–50.

[24] Piraghaj, S. F., Dastjerdi, A. V., Calheiros, R. N., & Buyya, R. (2017). ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. *Software: Practice and Experience, 47*(4), 505–521.

[25] Gupta, H., Dastjerdi, A. V., Ghosh, S. K., & Buyya, R. (2017). iFogSim: A toolkit for modeling and simulation of resource management techniques in Internet of Things, edge and fog computing environments. *Software: Practice and Experience, 47*(9), 1275–1296.

[26] Sonmez, C., Ozgovde, A., & Ersoy, C. (2018). EdgeCloudSim: An environment for performance evaluation of edge computing systems. *Transactions on Emerging Telecommunications Technologies, 29*(11), e3493.

[27] Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., & Leitner, P. (2017). Optimized IoT service placement in the fog. *Service Oriented Computing and Applications, 11*(4), 427–443.

[28] Wen, Z., Yang, R., Garraghan, P., Lin, T., Xu, J., & Rovatsos, M. (2017). Fog orchestration for Internet of Things services. *IEEE Internet Computing, 21*(2), 16–24.

[29] Tuli, S., Mahmud, R., Tuli, S., & Buyya, R. (2019). FogBus: A blockchain-based lightweight framework for edge and fog computing. *Journal of Systems and Software, 154*, 22–36.

[30] Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems, 28*(5), 755–768.

[31] Ghobaei-Arani, M., Souri, A., & Rahmanian, A. A. (2020). Resource management approaches in fog computing: A comprehensive review. *Journal of Grid Computing, 18*(1), 1–42.

[32] Brogi, A., Forti, S., Guerrero, C., & Lera, I. (2020). How to place your apps in the fog: State of the art and open challenges. *Software: Practice and Experience, 50*(5), 719–740.

[33] Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer, 36*(1), 41–50.

[34] Salehie, M., & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems, 4*(2), 1–42.

[35] Cheng, B. H. C., De Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H. M., Kramer, J., ... Whittle, J. (2009). Software engineering for self-adaptive systems: A research roadmap. In B. H. C. Cheng, R. De Lemos, H. Giese, P. Inverardi, & J. Magee (Eds.), *Software engineering for self-adaptive systems* (pp. 1–26). Springer.

[36] Huebscher, M. C., & McCann, J. A. (2008). A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys, 40*(3), 1–28.