

# ClusterSlice: A Zero-touch Deployment Platform for the Edge Cloud Continuum

Lefteris Mamatras (Member, IEEE), Sotiris Skaperas (Member, IEEE), Ilias Sakellariou (Member, IEEE)  
Department of Applied Informatics, University of Macedonia, GR-54636 Thessaloniki, Greece  
{emamatras, sotskap, iliass}@uom.edu.gr

**Abstract**—We demonstrate ClusterSlice, an open-source solution for automated Kubernetes-center deployments for the edge continuum. ClusterSlice is an infrastructure-as-a-service, platform-as-a-service, and application-as-a-service solution, supporting: (i) declarative deployment slice definitions; (ii) infrastructure-on-demand capabilities over multiple heterogeneous domains; (iii) composable Kubernetes deployments, supporting multi-clustering as well as various Kubernetes flavors and intra-cluster/inter-cluster network plugins; (iv) configurable application deployment; and (v) experimentation automation.

**Index Terms**—Kubernetes, Edge Computing, Edge Cloud System, Zero-touch Deployment, Cloud-Network Slicing.

## I. INTRODUCTION

ClusterSlice is an open-source Kubernetes-centered Edge Cloud solution, able to convert compute resources from bare-metal or hypervisor-level to fully-operatable cloud-network slices. One of its stronger features is the introduction of well-designed abstractions, that reduce deployment complexity with improved reliability and reproducibility, configuring the corresponding slice parts in a parallel fashion and zero-touch manner. ClusterSlice is cloud-native and fully integrated to Kubernetes, a widely used orchestration facility with impressive automation, fault-tolerance, scalability, resource optimization capabilities and, most importantly for the context of this work, a highly modular architecture. The latter allows Clusterslice to implement its functionality by harvesting on the concepts of Custom Resource Definitions (CRDs) and the Operator pattern, and to be fully managed via the `kubectl` tool. Due to this tight integration, it benefits from all Kubernetes reliability features. ClusterSlice supports:

- *Declarative automated operation*: Slice definition is done at a higher level in the form of YAML files, and includes compute resources to be utilized, Kubernetes configuration, and the application modules to be installed.
- *Infrastructure-as-a-service capabilities*: ClusterSlice supports the utilization of heterogeneous physical and virtual resources through a common abstraction, including open testbed infrastructures (e.g., CloudLab), as well as Virtual Machines (VMs) in XCP-ng and VirtualBox virtualization systems.
- *Platform-as-a-service features*: It supports multiple Kubernetes flavors, such as vanilla Kubernetes (k8s), k0s, k3s, and microk8s, as well as network plugins for both intra-cluster (e.g., Flannel, Calico, Kube-ovn, etc.) and inter-cluster communication (e.g., Submariner).

- *Application-as-a-service attributes*: The slice definition supports deployment of configurable application modules, i.e., the definition of applications to deploy, k8s extensions and modular OS configurations, ensuring maximum compatibility with heterogeneous systems.
- *Multi-cluster and multi-domain capabilities*: ClusterSlice operation can span across multiple heterogeneous deployment environments through technology-specific infrastructure managers, establishing multi-cluster operation and communication, such as Ligo, OCM, or Submariner.
- *Experimentation automation*: ClusterSlice is enabled advanced, automated deployment and management of experiments, based on its characteristics of reproducibility in the deployment of slices, and actively supported automation capabilities, such as comparable evaluations of network plugins and anomaly detection workflows.

ClusterSlice provides infrastructure-, platform- and application-as-a-service capabilities over the edge continuum. Relevant infrastructure control solutions include SLICES (<https://portal.slices-sc.eu>) and FABRIC [1] test-beds, which enable multiple infrastructure federations to inter-operate (e.g., Emulab, CloudLab, and others). Their novel resource control features and APIs (e.g., jFed, python APIs, AMs, etc.) make them re-usable for higher-level deployment abstractions. For example, CloudNativeLab (<https://practicum.cloudnativelab.ilabt.imec.be>) enables the creation of a Kubernetes cluster on SLICES testbeds, using a defined resource configuration for the cluster. Furthermore, 5G-CDN [2] and NECOS [3] are slicing infrastructures capable of allocating resources across multi-domain infrastructures.

The platform- or application-oriented solutions, include EdgeNet [4], SLATE [5] and Kubernetes-native testbed (<https://github.com/kubernetes-native-testbed>). EdgeNet [4] is a platform-as-a-service solution over multiple, crowd-sourced k8s worker nodes, implementing an edge continuum through CRDs and operators. SLATE federates science platforms, allowing sites to delegate service deployment and configuration to designated application administrators. The Kubernetes-native testbed integrates various microservices-based applications, CI/CD environments, and monitoring tools.

ClusterSlice supports multiple infrastructure managers under a common design abstraction, so all above mentioned solutions could be jointly utilized and federated. Furthermore,

it also supports both platform-as-a-service capabilities (e.g., composable Kubernetes clusters or multi-cluster deployments) and application-as-a-service capabilities, defining the application or Kubernetes extensions to use: all at the level of a declarative slice definition.

In the rest of sections, a brief presentation of ClusterSlice is covered in Section II, whereas sections III and IV detail the demo run and conclude the paper, respectively.

## II. PROPOSED SYSTEM

The automated transformation of resources from bare-metal or hypervisor setups into fully-operational Kubernetes-based edge cloud deployments, follows a unique approach: it benefits from k8s capabilities based on novel design abstractions.

ClusterSlice design is empowered by the Kubernetes paradigm, and its implementation heavily utilizes Custom Resource Definitions and Kubernetes Operators, thus inheriting the *reliability*, *scalability*, and *resource optimization* capabilities of Kubernetes. The implementation carefully avoids the use of APIs or technology-specific software, relying only on common tools such as *bash*, command-line utilities, and SSH through Ansible. This preserves *simplicity*, *portability*, and *composability* in line with the Unix philosophy.

Furthermore, ClusterSlice introduces *innovative abstractions* on different levels of its architecture:

- *node and cluster level automation abstractions*, i.e., Resource Managers (RMs) and Slice Operators (SOs), respectively. These act like digital twins of compute resources (i.e., physical hosts or VMs) and k8s clusters. ClusterSlice allocates and declaratively configures such containers, which, in turn, represent and actively configure the resources they represent.
- *infrastructure manager abstractions* that interface with diverse cloud/virtualization systems (e.g., VirtualBox or XCP-ng) and even testbed facilities (e.g., CloudLab), maximizing *compatibility* across heterogeneous systems and seamless *future-proof* integration.
- *multi-cluster and multi-domain abstractions*, to create and oversee multiple clusters and deploy the respective software, e.g., Ligo, Karmada or OCM, in the form of application modules.

A more detailed description of the ClusterSlice architecture and its individual components can be found in [6]. One of ClusterSlice’s strongest features lies in its ability to deploy a diverse range of characteristics for the deployment slice through a declarative definition. This is further demonstrated in the following section.

## III. DEMO DESCRIPTION

The demo involves the preparation of various slice deployment descriptors (e.g., as specified by the demo audience), in the form of YAML files. Subsequently, these descriptors are used for deploying corresponding cloud-network slices. Here, we provide a specific example involving three Kubernetes clusters across heterogeneous domains, accompanied by a multi-cluster management system, specifically Ligo. The

scenario considers the following diverse set of deployment environments:

- A local two-server test-bed hosting the XCP-ng cloud environment, known as “swntestbed”.
- A workstation PC hosting VirtualBox, named “lefteris”.
- The CloudLab test-bed, spread across various locations in the USA, referred to as “cloudlab”.

The declarative definition of the deployment is illustrated in Fig. 1. In the YAML file, we present the overarching properties of the experimentation slice, i.e., the name, namespace, server deployment strategy, and user credentials. After that, the specification of the three clusters follows. Each one of these specifications includes information on node, Kubernetes, and application requirements:

- *Cluster node requirements* such as the number of nodes in each cluster, the OS image to be deployed in master and worker nodes, and the node types. It should be noted that each cluster possesses its own deployment domain and unique resource type, for instance `pc3000` physical server at the CloudLab cluster or VM in the XCP-ng or VirtualBox clusters.
- *Kubernetes cluster configuration* supporting the full range of versions, i.e., vanilla, k0s, k3s, etc., and multiple network fabrics, e.g., Flannel, Calico, KubeRouter, etc., allowing for any (valid) combination of the previous, include requesting a specific version of the above, enhancing the reproducibility aspect of the deployment.
- *Application requirements*, that for this scenario include the Ligo Kubernetes extension. It is interesting to note that the Ligo master deployment (cluster named `liqo`), specifies, in a rather declarative manner, the participating clusters in the multi-cluster deployment (`'peers': '[liqo1,liqo2]'`).

The deployment is initiated by issuing a single `kubectl apply` command. The first phase involves allocation of resources for the three clusters, which is handled by the appropriate infrastructure managers depending on their type; for instance in the case of CloudLab by the *Testbed Infrastructure Manager* whereas in the case of VMs, an embedding process might also occur which leads to VMs allocated to specific cloud servers, via the *Cloud Infrastructure Managers*.

During the second phase, the deployment process involves several steps, including: (a) OS image installation and configuration, (b) Kubernetes cluster setup for the three distinct clusters (`liqo`, `liqo1`, and `liqo2`), and (c) the establishment of the Ligo multi-cluster management system, in this specific scenario.

After completing the above phase, Ligo is successfully implemented across three diverse clusters, as illustrated in Fig.2. At any deployment stage, executing the same command provides the status of all involved compute resources. Multiple such examples will be available during the demo.

## IV. CONCLUSIONS

ClusterSlice offers a declarative, effortless, zero-touch solution for transforming testbed resources, even from bare-metal,

