POLITECNICO DI TORINO
Repository ISTITUZIONALE

CrownLabs - A Collaborative Environment to Deliver Remote Computing Laboratories

*Original*
CrownLabs - A Collaborative Environment to Deliver Remote Computing Laboratories / Iorio, M.; Palesandro, A.; Risso, F.. - In: IEEE ACCESS. - ISSN 2169-3536. - ELETTRONICO. - 8:(2020), pp. 126428-126442. [10.1109/ACCESS.2020.3007961]

*Availability:*
This version is available at: 11583/2842768 since: 2020-08-19T19:10:49Z

*Publisher:*
Institute of Electrical and Electronics Engineers Inc.

*Published*
DOI:10.1109/ACCESS.2020.3007961

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*
GENERICO -- per es. Nature : semplice rinvio dal preprint/submitted, o postprint/AAM [ex default]

The original publication is available at https://ieeexplore.ieee.org/document/9136697 / http://dx.doi.org/10.1109/ACCESS.2020.3007961.

(Article begins on next page)

17 April 2024

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 29, 2020, accepted July 2, 2020, date of publication July 8, 2020, date of current version July 21, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3007961*

# CrownLabs–A Collaborative Environment to Deliver Remote Computing Laboratories

**MARCO IORIO, (Graduate Student Member, IEEE), ALEX PALESANDRO, AND FULVIO RISSO, (Member, IEEE)**
Department of Computer and Control Engineering, Politecnico di Torino, 10129 Turin, Italy
Corresponding author: Marco Iorio (marco.iorio@polito.it)

**ABSTRACT** The coronavirus pandemic hit the entire education sector hard. All students were sent home and lectures started to be delivered through video-conferencing systems. *CrownLabs* is an open-source project providing an answer to the problem of delivering remote computing laboratories. Simplicity is one of its main characteristics, requiring nothing but a simple web browser to interact with the system and being all heavyweight computations performed at the university premises. Cooperation and mentoring are also encouraged through parallel access to the same remote desktop. The entire system is built up using components from the Kubernetes ecosystem, to replicate a "cloud grade" infrastructure, coupled with custom software implementing the core business logic. To this end, most of the complexity has been delegated to the infrastructure, to speed up the development process and reduce the maintenance burden. An extensive evaluation has been performed in both real and simulated scenarios to validate the overall performance: the results are encouraging, as well as the feedback from the early adopters of the system.

**INDEX TERMS** Collaborative software, computer science education, kubernetes, laboratories, learning systems, platform virtualization, software architecture.

## I. INTRODUCTION

Practical learning is widely recognized as being an essential aspect while assimilating a new subject [1], [2]. It grants students the possibility to experience the practical applications of theoretical knowledge. It involves attempting to perform complex tasks, making mistakes, arguing with the teammates on how to proceed. It immediately unveils whether a concept is clear or not, as well as it fosters interactions with the teachers. In a nutshell, it allows students to learn more, to learn better. As for computer science courses, our students are typically required to attend computer laboratories. There, they have the opportunity to login to workstations already providing all the different pieces of software to perform the exercises assigned, as well as to interact with their classmates and instructors.

Yet, physical attendance may not be always possible, nor desirable. This aspect rose abruptly to prominence during the coronavirus pandemic. The virus hit the entire country hard [3]. All students were sent home, universities set up video conferencing servers in a few days, and classes were transformed into remote lectures. However, remote laboratories needed to be established too: how could students be enabled to practice with their coding exercises, simulations and more?

Students may be asked to install the entire set of applications, tools and simulators required for each laboratory on their own computers. Yet, the IT requirements, the different dependencies and incompatibilities, the number of parallel courses to follow during each semester, as well as the infinite series of problems that may arise from dozens or even hundreds of different environments, push to consider this solution as infeasible. Pre-built, ready-to-use virtual machines (VMs) made available for download to the students are a no-go too. Although considerably simplifying the initial setup, while guaranteeing at the same time uniform and isolated environments, VMs would strain even further the students' devices. Hence, discouraging their usage. Wouldn't it be better if the students could simply access their laboratory environment using their own browser?

To this end, *CrownLabs* is an experimental project developed by a group of volunteers from Politecnico di Torino to enable the delivery of computing laboratories through remote per-user virtual machines. Each student gets her own private

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


This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

environment without requiring to download and install any additional software but a simple web browser. Indeed, all heavyweight computations are performed at the university premises, by a Kubernetes cluster executed in a dedicated data center. Most importantly, *CrownLabs* has been designed from the outset to account for the typical educational requirements, group work and easy tutoring support among all. In a nutshell, the main advantages provided by the system include: (a) adaptability, enabling remote desktop sharing to complete the laboratory duties in a team and to seek for help from an instructor or other classmates; (b) flexibility, allowing the students to complete their tasks at any time, with full control of the life-cycle of their VMs; (c) versatility, granting the instructors the possibility to provision multiple laboratories, each one with tailored setups to account for the specific requirements; (d) compliance, providing access to licensed software allowed to be executed only at the university premises; (e) security, adopting encrypted sessions and enforcing the access to the resources through strict but configurable authorization mechanisms (e.g. to enable a restricted examination mode).

In the remainder of this paper, we present the most relevant characteristics of *CrownLabs*, both from the architectural and the infrastructural point of view. The main aim is to highlight the features mostly presenting aspects of novelty, as well as to share the different lessons we got while facing the challenges brought in by this project. Specifically, in Section II we review existing solutions concerning remote laboratories, focusing both on the research world and commercial solutions. Section III deepens the remote laboratories use-case, providing a general overview about the services made available by *CrownLabs*. In Section IV, we present the main choices that have driven the design and the development of *CrownLabs*. Section V details the key architectural components of the system, while Section VI draws a parallelism between the *CrownLabs* infrastructure and the typical cloud services. Then, Section VII presents an experimental evaluation of the overall *CrownLabs* performance in different scenarios. Finally, Section VIII draws the main conclusions and proposes directions for further research. For a more in-depth technical description, as well as the complete source code, please refer to the project's website[1] and the GitHub repository.[2] Indeed, the entire *CrownLabs* project is open-source and it is built on top of open-source components only: anyone who is interested in deploying the system in-house can freely create her own infrastructural setup or rely on a public cloud provider (granted it features the necessary virtualization functionalities), as well as deploy and customize the complete application logic.

## II. RELATED WORK

Remote laboratories are by no means a recent idea. Back in 2006, Ma and Nickerson [4] already debated about the advantages and the drawbacks associated with both hands-on and remote engineering laboratories. Specifically, they observed the growth of virtual solutions to reduce the cost pressures on universities. Yet, at the same time, they raised concerns about the potential isolation of students engaged in remote learning due to the lack of interactions with both peers and instructors.

On the other side, as mentioned in the introduction, virtual machines locally executed on the students' devices (e.g. with `VirtualBox`[3]) are not deemed a viable approach. First, they would impose a significant burden in terms of both demanded resources and initial setup, the latter being particularly critical for non-IT students. Second, local VMs would not be suitable for team work, other than heavily weighing on mentoring. Indeed, instructors could not investigate first-hand possible problems, needing to mostly rely on inaccurate students' descriptions. Even the screen sharing and remote control functionalities featured by tools such as Zoom[4] and TeamViewer,[5] besides possibly involving paid licenses, would not enable flawless collaboration, preventing simultaneous interactions and requiring the hosting student to allow the other team members and the instructors to remotely control her own personal PC. Conversely, virtual machines running in cloud are nowadays considered a commodity. Indeed, cloud providers (e.g. Amazon Web Services, Microsoft Azure, Google Cloud, just to name a few) offer the possibility to easily create and access remote VMs, while selecting the resources best suited to the final user's needs. Yet, these platforms are meant for general use-cases, thus providing no supports for learning purposes.

Over the past decade, multiple projects featured different solutions characterized by custom application logic to serve remote computing, networking and security laboratories on top of on-premise and cloud-based infrastructures. Among them, the StarHPC project [5] leveraged the Amazon's EC2 service to host the VMs used to support teaching parallel computing programming at MIT, while adopting custom scripts and VM snapshots to setup the laboratories. V-Lab [6], on the other hand, aimed to provide a network security experimental environment based on dedicated virtual machines and virtual networks. In short, it featured a graphical front-end to manage the virtual resources (i.e. create new VMs and configure the network topology), while the back-end was powered by Xen Cloud Platform (XCP), OpenStack and Open Virtual Switches (OVS). In 2016, Caminero *et al.* presented TUTORES [7], a solution to create virtual remote laboratories built on top of VMWare ESXi and OpenNebula. Yet, to the best of our knowledge, none of the above projects is open-source, hence providing no possibility to deploy and evaluate them in different environments. Additionally, these solutions are based on classical virtualization platforms and focus mainly on the evaluation of students and faculty

[1] https://crownlabs.polito.it
[2] https://github.com/netgroup-polito/CrownLabs
[3] https://www.virtualbox.org
[4] https://zoom.us
[5] https://www.teamviewer.com

acceptance of the system. Conversely, *CrownLabs* stems from a completely different approach, and it is strongly oriented towards both cooperation and mentoring. Additionally, from the technological point of view, it is powered by a Kubernetes cluster, hence differentiating from past solutions.

Talking about online but commercial solutions, it is possible to mention ThoTh Lab [8]. It provides a virtualized hands-on laboratory for computer science education based on cloud computing. Most notably, its landing page advertises a feature-rich, web browser-based UI allowing to interact with the system, set-up the topology for each laboratory (i.e. create new VMs and interconnect them through custom networks) and access the remote desktops. Additionally, collaboration and mentoring appear to be provided through real-time project viewing and an in-browser video chat system. Katacoda [9], on the other hand, advertises itself as an interactive learning and training platform for software developers. At a first glance, it features a set of free and ready-to-use scenarios. Each scenario is characterized by a detailed documentation of the tasks the user has to perform, complemented by an in-browser terminal emulator providing access to a personal environment where the assignments can be tested in practice. Yet, this solution targets individual and self-paced learning, thus providing almost no support for the collaborative environment and the interactive support typical of university classes.

## III. THE REMOTE LABORATORIES USE-CASE

In this section, we present how both students and instructors can interact with the *CrownLabs* system, as well as we focus on the extended features made available only to the latter for course administration.

### A. HOW DO THE STUDENTS INTERACT WITH CrownLabs?

Generally speaking, the central access point to *CrownLabs* is represented by an introductory web page, providing access to the login portal. Each authenticated student is presented a personal dashboard, displaying the courses she is currently enrolled in and the list of available laboratories. Indeed, multiple laboratories can be deployed for the same course, in order to account for different requirements (e.g. in terms of software), as well as to avoid the need for one-fits-all VMs that would be overly demanding resource wise.

Selecting a specific laboratory, each student can independently manage the life-cycle of the corresponding VM. Hence, she is allowed to start practising at her own will, as well as to restart from a clean environment in case of necessity (e.g. to recover from a corrupted setup). Once a new VM has been spawned and the operating system is ready, the student can connect to the remote desktop by being redirected to a new web page. There, she can interact with the VM as if it were executed on her own device, although being totally unrelated from the local computational resources and without requiring any preliminary setup. Cooperation is encouraged: the access to the same VM can be extended to multiple students, to allow for synchronous collaboration on the same tasks, facilitate group works and enable peer support. Finally, web-based personal storage is assigned to each student. Hence, the artifacts of the different laboratories can be persisted, as well as easy file exchange between local and remote machines is enabled. Course-wide shared folders are also envisioned, to make exercises and preliminary material available to the entire class and to simplify the delivery of assignments.

### B. WHAT CAN INSTRUCTORS DO MORE?

Instructors are presented an extended dashboard. In addition to possibly spawning their own VMs, e.g. to test in advance a new laboratory, they are enabled to perform the set of administrative tasks required to setup a course. Specifically, they can (a) create new courses; (b) enable new students to access a specific course: student accounts are automatically created by the system if not already present, and the set of authorizations is updated; (c) create new laboratories, by uploading the corresponding disk image and configuring the VM characteristics. Most importantly, instructors are also presented the entire list of VMs launched by their students. Similarly to being in a physical laboratory, they can then access each remote desktop whenever necessary, e.g. to answer the questions posed by the students. Advantageously, the teacher can see exactly what the student is doing (and vice versa) as well as concurrently interact with the VM itself to thoroughly investigate possible problems and provide practical suggestions on how to solve them.

## IV. DESIGN CHOICES

This section details the main requirements associated with the *CrownLabs* project, together with the design choices that have driven its development from the early stages. Specifically, we justify the adoption of Kubernetes as an orchestration platform, introduce a brief comparison between VM-based and Docker-based services and finally discuss about the implementation of the custom back-end, the exploitation of code generation and the usage of additional computation resources. Although the presentation is targeted to the *CrownLabs* project (i.e. the provisioning of remote laboratories), the design requirements and principles summarized in Table 1 stem from a more general consideration. Hence, their applicability can be extended to a wide range of applications.

### A. LIMITING THE COMPLEXITY

Generally speaking, the key aspect enabling complex web-based application workflows is the back-end, the server side component implementing the business logic and exposing an interface to allow the interaction with the clients. Yet, a custom back-end is typically associated with a great complexity, being a single point of failure as well as an attractive target for the attackers. Indeed, besides the core functionalities, a monolithic solution needs to manage aspects concerning user authentication and authorization, as well as API input data validation.

**TABLE 1. High-Level project's design requirements and principles.**

| | |
|---|---|
| **DR1:** | Build a "production-ready" system as fast as possible. |
| **DR2:** | Implement custom APIs to manage application workflows. |
| **DR3:** | Limit the amount of custom code to be maintained. |
| **DR4:** | Prevent application and libraries obsolescence. |
| **DP1:** | Delegate as much tasks as possible to the infrastructure. |
| **DP2:** | Keep the components simple and leverage modularity. |
| **DP3:** | Exploit automatic code generation. |
| **DP4:** | Reuse infrastructural components for the application. |

When dealing with custom software developed by universities and industries, one additional aspect to keep in mind is the typical lack of maintenance. In these contexts, applications are usually developed to solve a specific need and, then, kept in production for years without further modifications. As new security issues are discovered every day, each application should be periodically rebuilt against updated libraries and bugs should be fixed. Yet, the core team starts new projects, the developers move to different jobs and the students graduate: in the end, nobody takes care of application maintenance.

These factors, coupled with the requirement for an experimental, but *"production-ready"* system built from the grounds up in just few weeks, made us immediately discard the development of a monolithic solution. Conversely, we leveraged modularity to keep the tasks executed by each component as simple as possible, while delegating most of the burden to the infrastructure itself. Indeed, widely adopted open-source platforms such as Openstack [10] and Kubernetes [11] benefit from a large community of developers, ensuring a continuous support. Additionally, the management of on-premise clusters is typically delegated to a dedicated team, taking care of the periodic updates as well as of the application of security patches to face newly discovered threats. Similarly, managed solutions incur in automatic rollouts performed by hosting providers.

Deriving from these considerations, the delegation of most of the complexity to the infrastructure represented a fundamental design principle of *CrownLabs*. Specifically, besides operational aspects concerning availability, replication and self-healing, this process has involved also some core parts of the application, API management among all (cf. Sections IV-D, and IV-E). Additionally, in order to limit the amount of custom code to be developed and maintained, we leveraged another universal design principle: the use of source code automatically generated from higher level artifacts (cf. Section IV-F).

### B. ADOPTING KUBERNETES AND ITS ECOSYSTEM

Following the philosophy adopted by our university in delivering remote lectures, we decided to deploy the entire system on-premise, leveraging a dedicated data center. As for the orchestration of virtual machines, Openstack is probably one of the most well-known, open-source solutions today available. Yet, for the *CrownLabs* project we selected a completely different approach, leading to the adoption of Kubernetes and its ecosystem since the beginning. All in all, Kubernetes is an open-source and easily extensible platform originally developed by Google to orchestrate containerized applications and manage the resources enabling the interaction between different micro-services. Although apparently counterintuitive, being designed for containers instead of VMs, multiple project's architectural goals pushed in this direction. First, besides being easy to setup, extend as well as scale, Kubernetes already features a large ecosystem of off-the-shelf companion services. Second, it represents a key enabler for different core aspects of the *CrownLabs* project, as detailed in the following.

### C. VIRTUAL MACHINES VS CONTAINERIZED APPLICATIONS

As for now, the *CrownLabs* project mainly focused on the deployment of virtual machines. Indeed, VMs are characterized by an interaction workflow closer to what a typical user would expect when connecting to a remote desktop. Specifically, complete access to a standard operating system as well as the possibility to run multiple applications in parallel. That is to say, VMs are essentially general-purpose, introducing no constraints on the set of operations that can be performed and being suitable for a wide audience of final users. Yet, all these characteristics come at a high price in terms of resources, start-up time and disk footprint. Hence imposing serious limitations on the number of concurrent users that can be supported.

The adoption of the Kubernetes ecosystem allows to seamlessly introduce the support for containerized applications. In other words, instead of providing the final users access to a full-blown VM, *CrownLabs* could be easily extended to expose ready-to-use applications (e.g. word processors, IDEs, simulators, etc.) executed within lightweight containers on top of a very thin graphical layer. Remote desktop access would then be guaranteed by means of a sidecar container, in charge of relaying the inputs and the video connection. In our opinion, the combination of these two approaches would allow to take the best from each technology: the agility of the containers when dealing with single applications, complemented by the generality of VMs to suit more complex workloads.

### D. A KUBERNETES-POWERED APPLICATION BACK-END

Concerning the application back-end, in an effort to limit the complexity (Section IV-A), we adopted a modular approach, while delegating as much as possible of the burden directly to the infrastructure itself. To this end, Fig. 1 presents a high-level comparison between the typical, ''application server''-oriented approach and the strategy adopted in *CrownLabs*, while drawing a parallelism with the well-established Model-View-Controller (MVC) design pattern [12]. Specifically, let consider as a use-case the creation and setup of a new laboratory instance through the web-based

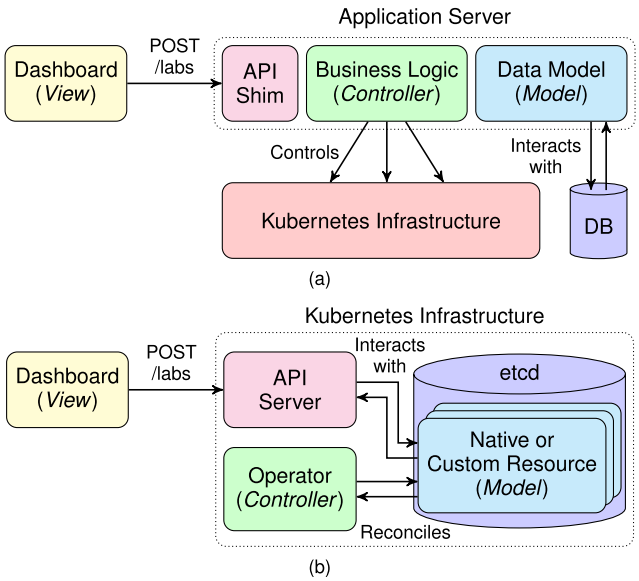


**FIGURE 1.** A high-level architectural comparison between (a) the traditional approach, characterized by an application server driving the infrastructure, and (b) the *CrownLabs* approach, where the constituting blocks are integrated directly into the Kubernetes infrastructure to offload most of the burden.

dashboard, i.e. the view the user interacts with, which then interfaces with the back-end by means of `REST` APIs.

Focusing on Fig. 1a, the back-end would typically encompass an application server that, besides the actual API endpoints automatically generated by the adopted framework, could be divided into two main components, according to the MVC pattern. First, the model, taking care of data representation and interacting with an external database to guarantee persistence. Second, the controller, which includes the core business logic in charge of reacting to the external input to update the model, as well as to drive the underlying infrastructure (e.g. to create the resources required for the laboratory) whenever necessary.

Fig. 1b, on the other hand, outlines the approach adopted in *CrownLabs* and characterized by the integration of the application components directly into the Kubernetes infrastructure. To this extent, the data model is represented by standard Kubernetes resources, both native and externally defined by means of Custom Resource Definitions (CRDs). Instead, the business logic is implemented through ad-hoc operators, created in accordance with the standard Kubernetes workflow paradigm. Operators embed the human knowledge about the resources, to reconcile the current status to the desired one expressed by means of the resources themselves. Decoupling, one of the main advantages associated with the MVC design pattern is preserved, as well as the possibility to deploy parallel controllers operating on the same model. Yet, our approach is also characterized by a greater degree of simplicity, reducing the number of components to be developed and maintained. Additionally, data persistence is facilitated, removing the need for an external database by reusing all the high availability functionalities (e.g. replication, backup strategies, ...) already provided by the `etcd`[6] cluster. Although this approach lacks in part the flexibility provided by a traditional query language such as `SQL`, we still found it perfectly suited to our use-case, while benefiting greatly from its associated simplicity.

### E. OFFLOADING THE API MANAGEMENT

Stemming from the approach outlined in Fig. 1b, all API management functionalities are then exposed through the Kubernetes API server itself, which is directly accessed by the web-based front-end. All in all, this choice involved the main advantages detailed in the following.

1) *The ease of business logic definition:* Kubernetes enables the declarative description of custom and versioned APIs by means of CRDs. Once installed, Custom Resources (CRs) are served and handled by the API server in the same manner as the native resources, thus benefiting from the exact same features. In addition to defining REST APIs by means of CRDs, Kubernetes also provides support for different semantics through the so-called "Aggregation Layer". In this way, the API server allows to register specific handlers to answer for totally custom requests, hence enabling applications with more complex workflows. Hence, the high degree of API customization provided by Kubernetes, complemented by an identity provider to manage the authentication aspects, represents a key enabler to the future extension of *CrownLabs*.
2) *The reuse of existing features:* the Kubernetes API Server is designed to support thousands of worker nodes out of the box, hence tolerating high workloads in terms of requests per second. Additionally, it already integrates a state-of-the-art processing pipeline automatically executed by the API gateway whenever a new request arrives. In a nutshell, it handles authentication and authorization aspects, input validation as well as more advanced security features, including rate limiting, to prevent denial of service attacks. Advantageously, the validation pipeline can be further enriched by means of custom "admission webhooks" defined through declarative policies (cf. `OpenPolicyAgent`[7]), hence offloading semantic validation from the business logic.
3) *The reduction of the operational costs:* being actively used and maintained by a world-wide community, the Kubernetes API server features continuous development, bug fixes and security auditing. Additionally, as a core part of the infrastructure itself, it also benefits from the periodic updates performed by the cluster operators, as well as it avoids the need for yet another component to maintain. Finally, since the same solution can be adopted for multiple applications hosted by the same cluster, the overall operational cost can be further spread.

[6]https://etcd.io
[7]https://www.openpolicyagent.org

While postponing the in-depth technical description to Section V-A, it is worth briefly anticipating here the discussion about the limitations associated with this approach and the possible mitigation that can be adopted.

1) *The security concerns:* according to our solution, the API server is exposed on the Internet to untrusted users. Yet, personal accounts managed through a central identity provider, coupled with the native permission management provided by Kubernetes, allow to strictly limit the set of operations permitted to each user to the bare minimum. Additionally, stemming from the previous considerations on the update procedures, it is possible to assume the timely installation of security patches. DDoS attacks are also deemed not to introduce excessive concerns, thanks to the intrinsic protections integrated in the API server and the degree of isolation provided by the reverse proxy sitting in front the server itself. Finally, although for the sake of the discussion we mentioned exposing the entire API server, it is possible to leverage its hierarchical organization to disclose only the custom APIs necessary for the end-user applications. All in all, we believe these countermeasures to be sufficient to mitigate the security concerns associated with the public API server.
2) *The support for transactions:* the Kubernetes API server does not support the concept of transaction (i.e. to specify a set of operations to be atomically executed together). Instead, each resource is characterized by a current state, that will eventually converge to the desired one. Yet, transaction-like mechanisms can be implemented by custom operators, to spawn/delete, e.g., a set of companion components upon the creation/deletion of a given CR.

### F. EXPLOITING AUTOMATIC CODE GENERATION

Continuing with the overall goal of limiting the development effort, and the consequent risk of introducing both unwanted bugs and security issues, to the bare minimum, another key design choice of the project consisted in leveraging as much as possible automatic code generation features.

Considering the back-end, we adopted `Kubebuilder`,[8] a well-known tool simplifying the development of the CRDs as well as of the associated operator implementing the control loop. Essentially, this tool takes care of all the initial scaffolding tasks, automates the generation of boilerplate code and provides high-level abstractions to enable developers to focus only on the implementation of the actual business logic. Most notably, it also supports the generation of `OpenAPI V3` schemas,[9] hence enabling the syntactic validation performed by the API server to screen the requests and discard non-compliant objects.

Similarly, regarding the front-end, the web-based UI is implemented using `ReactJS`, a widely adopted library providing a declarative approach to describe the different components, automatically translated into `HTML` and `CSS` code. The interaction with the Kubernetes API, on the other hand, is realized through a patched version of the official Kubernetes JavaScript client.[10] Indeed, the official version was designed to be coupled with `node.js`, hence being suitable for server-side interaction only. Yet, one of its main advantages concerns the support for both Kubernetes native resources as well as for custom types (i.e. those implemented by means of CRDs). Finally, the integration of the resulting modules is provided by `WebPack`, in charge of building the dependency graph and exporting the different `JavaScript` modules as a bundle.

### G. JOINING UNDERUTILIZED RESOURCES TO THE CrownLabs CLUSTER

Universities typically encompass multiple laboratories characterized by hundreds of workstations. Each computer usually remains continuously operative, although being effectively utilized only when practical lectures take place. All in all, they represent a stock of computational capacities that for most of the time may be better leveraged for different purposes. To this end, the flexibility of Kubernetes, coupled with its ease of installation could represent an effective solution. Indeed, focusing on the remote laboratories use-case, let suppose to run *CrownLabs* in a dedicated data center. Then, assuming a sufficiently fast campus backbone network, the computational resources of different laboratories could be configured and joined to the central cluster. Hence, the cluster would scale horizontally, allowing to increase the number of parallel users by deploying the VMs even on these additional workstations. Yet, this scalability would come almost for free, better exploiting already available resources and preventing the need for purchasing new ad-hoc servers. This concept of dynamic and transparent resource federation is further explored and addressed by the *Liqo*[11] project.

## V. THE CrownLabs ARCHITECTURE

This section details the main architectural components at the core of the *CrownLabs* project. With reference to the schematic representation shown in Fig. 2, the system can be mainly divided into two parts: the *front-end* and the *back-end*. The former is composed of a set of web pages, carrying out the login process and presenting the management dashboard to the final users (i.e. students and professors). The latter is responsible for the deployment and the execution of the actual VMs. It integrates multiple components of the Kubernetes ecosystem with custom business logic, implemented as Kubernetes extensions (i.e. CRDs and operators). In the following, we describe in greater detail the most relevant aspects of both components, present the technology adopted for remote desktop interaction, as well as characterize the key aspects about the authentication and authorization process.

[8]https://github.com/kubernetes-sigs/kubebuilder
[9]https://www.openapis.org
[10]https://github.com/scality/kubernetes-client-javascript/tree/browser
[11]https://liqo.io

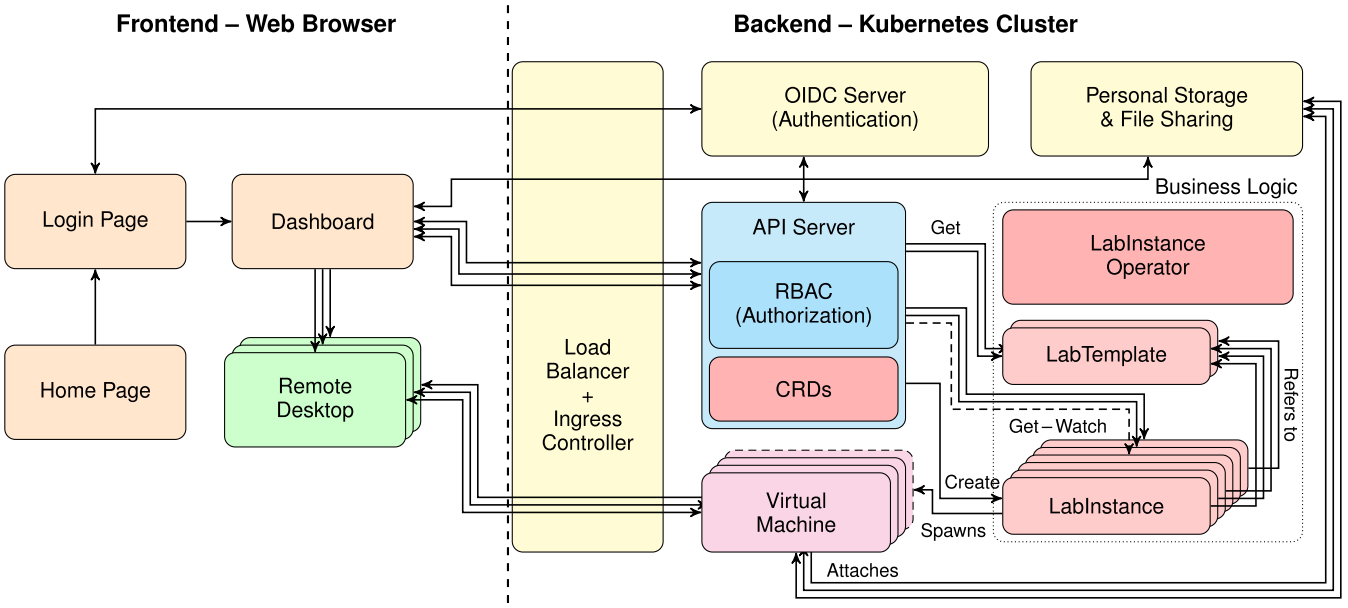


**FIGURE 2.** A high-level representation of the main architectural building blocks composing the *CrownLabs* project. Please notice that, for the sake of clarity, the figure depicts the blocks essential for the provision of the actual *CrownLabs* service, omitting the ones that are dedicated to the cluster operation (e.g monitoring).

### A. THE CrownLabs BACK-END

Stemming from the design choices presented in Section IV-B, the server-side of the *CrownLabs* project is completely powered by a Kubernetes cluster, which is responsible for both the actual execution of the user virtual machines as well as the provisioning of the companion services. As for the externally accessible services, they are exposed outside the cluster through an Ingress Controller. In a nutshell, it is a component responsible for relaying the connections originated from the clients towards the target containers, acting as a TCP and TLS terminator, while transparently managing session securization and high availability aspects. For the sake of reliability, a load balancer is in charge of the management of the public IPs, guaranteeing their reachability even in case one of the servers does no longer work properly (cf. Section VI-B3).

Under the hood, virtual machines are managed through the `KubeVirt` extension.[12] It is a solution allowing to declaratively create and control the life-cycle of `qemu`-based VMs on top of a Kubernetes cluster, as better detailed in Section VI-B1. The external interface, i.e. the one leveraged by the web clients upon users' interaction, is directly provided by the Kubernetes API server. To this end, we leveraged the expression power of CRDs to model the core functionalities made available to the final users. Specifically, focusing on the provisioning of remote laboratories, we expressed two main resources, respectively named `LabTemplate` and `LabInstance`. A `LabTemplate` consists of a wrapper around a KubeVirt's `VirtualMachine` resource. It models the concept of a laboratory, belonging to a specific course, and the associated set of VMs, along with their characteristics. A `LabInstance`, on the other hand, represents a well defined instance of a given `LabTemplate`, linked to a final user (i.e. its owner), by means of a Kubernetes namespace.

The managements of the `LabInstance` resources is delegated to the corresponding `LabInstance Operator`, a custom component developed according to the well-known operator pattern [13]. In a nutshell, it implements a control loop to automate the creation of the entire set of companion components (e.g. services, ingresses, ...) required to allow remote and secure access to the actual VMs. Similarly, it is in charge of their removal upon the deletion of a `LabInstance`.

### B. THE CrownLabs FRONT-END

The core of the *CrownLabs* functionalities is presented to the final users by means of a web-based dashboard, which is accessible once the user is authenticated in the system. The dashboard exposes in an intuitive and attractive format the set of operations available (i.e. creation and destruction of a VM, as well as the connection to its remote desktop). Finally, it provides an easy access to the user's personal storage deployed on premise by means of industry-standard protocols, such as `WebDav`, through open-source tools (i.e. `Nextcloud`[13]). Each user gets her own personal folder automatically attached to her VMs, hence enabling persistent storage and easy file sharing between the local and the remote environment.

Under the hood, the dashboard directly interacts with the Kubernetes API server through the patched JavaScript client and the abstraction automatically generated on top of it

[12]https://github.com/kubevirt/kubevirt

[13]https://nextcloud.com

(cf. Section IV-F). Indeed, a new laboratory, its associated VMs and the companion resources, can be started by simply creating in the user's namespace a new `LabInstance` resource, referencing the `LabTemplate` of interest. Symmetrically, an existing laboratory can be teared down by deleting the corresponding `LabInstance`. The list of own active VMs can be obtained through a `GET` operation on the `LabInstances` present in the personal namespace, while their status can be monitored by `WATCHING` the corresponding resources for the emission of events (e.g. during the initial setup). Additionally, each `LabInstance` resource contains the pointer to the URL where the remote desktop is exposed. Hence, the final users can connect to it through just one mouse click.

### C. ACCESSING THE REMOTE DESKTOP

Remote desktop interaction is provided with `TigerVNC`,[14] a high performance implementation of the Virtual Network Computing (VNC) graphical desktop sharing system. It is composed of a client/server application in charge of transmitting the flow of graphical screen updates from the remote to the local machine, while relaying keyboard and mouse events. Although the hypervisor already features simple VNC functionalities, we nonetheless preferred using an external, easier to configure and more feature-rich solution (e.g. enabling automatic remote desktop resizing), to provide a better user experience. Yet, the former represents a valuable fallback to access the virtual machine in case the main server is no longer reachable. To remove the need for ad-hoc client viewers, we finally leveraged the `noVNC` project[15] to proxy the VNC connection through a websocket and make it accessible from the browser.

Another well-known and open-source solution to access a remote desktop is represented by `SPICE`.[16] Differently from the previous approach, which transmits a flow of video updates already rendered, `SPICE` basically tunnels an X11 session to the client (i.e. the set of events triggered to redraw the different areas of the screen). Although being definitely interesting, an initial high-level comparison suggested that this approach tends to perform worse when moving outside a LAN due to higher latency. This factor, coupled with the availability of a more feature-rich HTML5 client, justified the adoption of the VNC-based solution. Yet, a more complete and in-depth investigation is planned as future work.

### D. SECURING CrownLabs

The secure access to every resource exposed by the *CrownLabs* system is guaranteed by encrypted sessions complemented with personal, single sign-on, accounts. To this aim, centralized authentication is provided by means of `Keycloak`,[17] an open-source identity and management solution that we deployed on top of the Kubernetes cluster. This component takes care of the entire user registration process and of the management of authorization policies (i.e. groups), while exposing a standard OpenID Connect (OIDC) interface.[18] Essentially, it allows different clients to verify the identity of the end users and obtain the corresponding profile information. Hence, it represents a key component to enable modular applications, as well as to simplify the development and the interaction with future extensions and add-ons.

Each user is associated with a single account, granting her access to, e.g., the *CrownLabs* dashboard, the remote desktop of her own VMs and her personal storage. Moreover, every user of the system is assigned to a set of groups, stating the exact set of courses she can access, her role for each of them (i.e. student or professor), as well as any additional privilege concerning the cluster operation and maintenance. Focusing on the personal dashboard, the user's data contained in the access token returned by the OIDC server upon authentication (e.g. attributes and groups) is leveraged to customize the view presented to the user herself and show only relevant pieces of information. Furthermore, the Kubernetes API server has been interfaced with Keycloak too. Thus, each request is automatically discarded if not enriched with a valid authentication token. Finally, remote desktop access is protected by configurable group-based authorization policies, to enable team work and peer support. Yet, they are complemented by URL obfuscation to prevent random guessing in case of course-wide access.

Server-side isolation and access control is implemented by means of standard Kubernetes resources, namely namespaces and role bindings.[19] Specifically, each user is assigned a personal namespace where she is granted permission to interact with `LabInstances` only, according to a white listing policy. Similarly, `LabTemplate` resources are grouped in course-wide namespaces. Read access is provided only to the students enrolled in the corresponding course (i.e. belonging to the authorization group). Though, professors are associated with extended privileges, being allowed to manage and access all the virtual machines owned by the students of their own courses.

## VI. A CLOUD GRADE INFRASTRUCTURE

In this section, we present an overview of the main infrastructural components laying the foundations of *CrownLabs*. Specifically, we start with a brief description of the on-premise physical infrastructure hosting the service, given its significance for the experimental evaluation detailed in Section VII. Then, stemming from the high-level services typically exposed by a cloud provider, we characterize the main components of the Kubernetes ecosystem we selected for their implementation. This section demonstrates how *CrownLabs* is suitable to run not only on managed infrastructures, but also in-house, on relatively common hardware

[14] https://tigervnc.org
[15] https://novnc.com
[16] https://spice-space.org
[17] https://www.keycloak.org
[18] https://openid.net/connect
[19] https://kubernetes.io/docs/reference/access-authn-authz/rbac

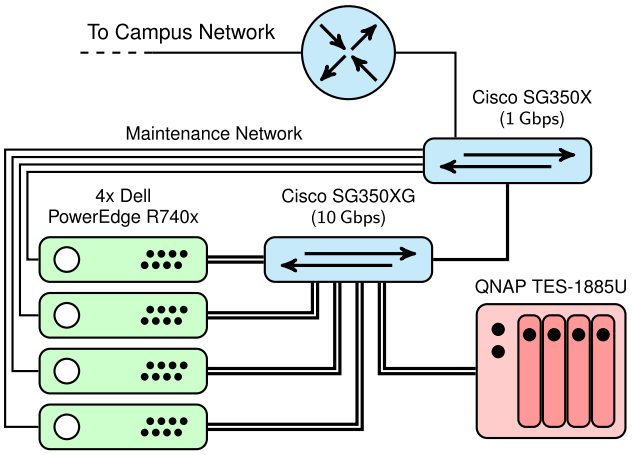


**FIGURE 3.** A representation of the physical infrastructure hosting *CrownLabs*.

available e.g. in schools and universities, while still maintaining its excellent scalability properties.

### A. THE BARE-METAL KUBERNETES CLUSTER

The entire *CrownLabs* service is powered by a Kubernetes cluster hosted at the university premises and pictorially represented in Fig. 3. In detail, it is mainly composed of four Dell PowerEdge R740x servers, each featuring a single Intel Xeon Gold 5120 CPU (with 14 hyper-threaded cores operating at 2.20 GHz) and 64 GB of RAM. The servers are interconnected to a central switch by means of two 10 Gbps links configured in link aggregation mode, while the persistent storage is supplied by local disks and a QNAP TES-1885U NAS attached to the same network. The high-speed intra-cluster network plays an important role to drain the east-west traffic, which is mostly generated by `kube-proxy`, a Kubernetes component possibly distributing the incoming traffic to a different node, and the distributed storage system. Finally, the connection to the rest of the campus network (and the Internet) is provided by a 1 Gbps link.

From the logical point of view, each server plays the role of a Kubernetes worker, while the master node is hosted by a `KVM`-powered virtual machine running on one of the servers. Indeed, given the limited number of resources available, we preferred avoiding to dedicate one entire server just for the Kubernetes master and we leveraged virtualization to achieve an acceptable degree of isolation. For the same reason, we adopted a single master setup, although loosing the advantages associated with high-availability. As for Kubernetes networking, we selected `Project Calico`,[20] being it one of the most popular CNI plugins, limiting the overhead by requiring no overlay and supporting advanced features such as the definition of network policies to isolate the traffic between different containers.

[20]https://www.projectcalico.org

**TABLE 2.** Mapping the typical cloud computing services to the key components selected for their implementation.

| Cloud Provider Service | CrownLabs Component |
|---|---|
| Compute Virtualization | Kubevirt |
| Tenant Virtual Networking | None |
| Firewall as a Service | Network Policies (Project Calico) |
| Load Balancing | MetalLB + NGINX Ingress Controller |
| DNS Synchronisation | external-dns |
| Certificate Provisioning | cert-manager |
| Block and Object Storage | Rook + Ceph |
| Local Docker Registry | Docker Registry 2.0 |
| Monitoring as a Service | Kube-Prometheus |
| Identity Provider | Keycloak + oauth2-proxy |
| File Sharing | Nextcloud |

### B. ADDING TYPICAL "CLOUD" SERVICES ON TOP OF KUBERNETES

Once the bare Kubernetes cluster is up and running, it is necessary to install the entire set of components required to create the fully-fledged platform to host *CrownLabs*. A huge amount of customization and fine-tuning is clearly required in this respect, depending on the characteristics of the physical setup, the desired degree of fault tolerance as well as to work around possible external constraints. Yet, in the following description, we stick to a high-level presentation, motivating the requirement for the different components by means of a parallelism with the services provided by cloud providers, as summarized in Table 2. Indeed, we can show that, picking the right constituting elements from the Kubernetes ecosystem, we succeeded in replicating (even though in a smaller scale) all the functionalities typically available in the cloud world. Nonetheless, the entire configuration files required to reproduce our setup are available on the project's repository.

#### 1) COMPUTE VIRTUALIZATION

Kubernetes is designed for the orchestration of containers. Indeed, the smallest deployable unit of computing that can be managed by Kubernetes is the *pod* (i.e. an atomic entity grouping together one or more containers with shared networking and storage capabilities). Yet, the *CrownLabs* use-case brought in the necessity to handle also classical virtual machines, in order to provide a generic and totally isolated environment where students can play with their own laboratories. Here, `KubeVirt` comes into play, introducing the support for generic VMs on top of a Kubernetes cluster. In short, whenever a `VirtualMachineInstance` is created (i.e. the custom resource describing its characteristics), the `KubeVirt` operator spawns a new pod, representing the space where the VM will live. This pod is composed of a launcher container, mainly responsible for creating the local `libvirtd` instance that actually manages the life-cycle of the VM. A second container, on the other hand, essentially wraps a `qcow2` disk image file, representing the HDD that is attached to the newly created virtual machine. A great

amount of tuning is possible for what regards resource management, in order to specify the amount of CPU and memory reserved to each VM and optionally the desired degree of over-commitment. Finally, as for the HDD images packaged as Docker images, we setup a local Docker Registry, in order to speed-up the VM boot time and prevent saturating the bandwidth of our Internet connection, given their considerable size.

To simplify the setup of the VM images, we developed a set of `bash` and `Ansible`[21] scripts, which automate (on a local machine, e.g. with `VirtualBox`) the entire preparation process, from the installation of the guest O.S. to the upload of the resulting image to the private Docker Registry. Specifically, they take care of downloading and configuring all the tools required for *CrownLabs* itself, as well as removing a set of redundant programs to reduce the disk footprint and speed-up the boot process. Additionally, multiple predefined Ansible playbooks are made available to simplify the installation of common software used in our courses. Yet, additional playbooks can be prepared to support different use-cases, as well as graphical one-shot configurations are possible. Based on our experience, the resulting VMs (all adopting `xubuntu` as base O.S.) were characterized by a virtual file-system usage ranging from 4.1 GB for the stripped down O.S., to 5.0–7.0 GB in the most common scenarios, up to 11.0 GB when MATLAB (complemented by some toolboxes) was installed. Yet, the compressed size when stored within the Docker registry was much lower, fluctuating from 1.5–2.5 GB in all but the most demanding situation (5.4 GB). These values represent the amount of data that needs to be transferred to each node when spawning a VM for the first time.

#### 2) VIRTUAL NETWORKING AND ISOLATION

Differently from most VM-oriented platforms, Kubernetes adopts a flat addressing space. Indeed, each CNI implementation is required to allow every pod to directly communicate with all the other pods, whatever the node they reside on. Back to the *CrownLabs* use-case, this limitation made it impossible to directly impose a strong isolation between the students through the usage of totally separated virtual networks. Yet, isolation can be achieved by means of Kubernetes network policies, which provide a declarative way to express how groups of pods are allowed to communicate between each other and with external network endpoints. We leveraged this approach to prevent the VMs in each student's namespace from being contacted by those of other users. Yet, we allowed internal communication, to support multi-VM setups for networking laboratories.

#### 3) LOAD BALANCING

One of the key components required by every multi-server setup is the load balancer, the element in charge of distributing the ingress traffic to the correct back-end service. Focusing on the *CrownLabs* stack, this task is fulfilled by two different pieces of software. First, `MetalLB`,[22] that implements a network load balancer for bare metal clusters (i.e. those not running on supported IaaS platforms). Yet, having no control on the border router, we were forced to operate in `Layer2` mode (i.e. only one node in the cluster attracts all the traffic for each virtual IP address). Hence, our solution achieves mostly resiliency, guaranteeing that the virtual IP addresses continue to remain reachable even in case one of the physical nodes is no longer accessible, as each server runs its own `MetalLB` instance. Nonetheless, once the traffic is received by the node, `kube-proxy` takes care of its automatic distribution to all the pods associated with the service, thus effectively achieving load-balancing. Second, the `NGINX` Ingress Controller,[23] which is responsible for providing the actual entry point for all HTTP-based application traffic and is executed in multiple instances. This component, which is exposed outside the cluster through one ''load-balanced'' virtual IP address, selects the target back-end service, depending on the host name and, optionally, the requested URL path. Additionally, it provides TLS termination, to ensure that all traffic exchanged with the end users is secured. Finally, two complementary components, namely `external-dns` and `cert-manager`, are responsible for companion tasks triggered whenever a new `Ingress`[24] resource is created. Specifically, the former takes care of the automatic configuration of DNS records on external DNS providers (e.g. a `bind9` server), to make the specified host names immediately discoverable via public DNS servers. The latter, on the other hand, automates the issuance and renewal of valid TLS certificates with the ''Let's Encrypt'' public Internet service.

#### 4) STORAGE PROVISIONING

Most applications cannot provide any useful service to the users if data cannot be persisted. In Kubernetes, this problem is tackled by means of an abstraction named `PersistentVolumeClaim`. It represents a request for a piece of storage, characterized by a given type, size and access mode, that can be eventually attached to one or more pods. Yet, the actual implementation is provided by external solutions. Specifically, we leveraged `Rook`,[25] an operator that essentially provides a management layer to automate the deployment, configuration and upgrade of the actual storage providers. As for the latter, we selected `Ceph`,[26] thanks to its maturity and the support for both block, file-system, and object storage, through the definition of different storage classes. Indeed, the former provides a raw pool of storage that can be individually formatted by the user, while the file-system solution exposes a typical abstraction characterized by `POSIX` semantics. Finally, object storage provides an

[21] https://www.ansible.com

[22] https://metallb.universe.tf

[23] https://kubernetes.github.io/ingress-nginx

[24] https://kubernetes.io/docs/concepts/services-networking/ingress

[25] https://rook.io

[26] https://ceph.io

S3 compliant abstraction to save unstructured data in a flat address space. In *CrownLabs* we made use of all these three types of storage classes, depending on the peculiarities and constraints of the different applications.

#### 5) CLUSTER MONITORING

When it comes to operating complex systems, one essential requirement is the availability of rich and easy to access metrics to evaluate both the status of the different components and the evolution of the key performance indicators representing the user perception about the overall service. To address this requisite, we leveraged `kube-prometheus`,[27] a solution packaging together `Prometheus` (i.e. the component scraping the actual metrics from multiple endpoints and exposing them through a flexible query language), `Grafana` (i.e. the visualization platform adopted to graphically present the metrics) and `Alertmanager` (i.e. the tool in charge of managing the alerts and routing them to the correct receivers). In addition, `kube-prometheus` provides metrics exporters, ready-to-use dashboards as well as configurations to automate the installation of a complete, highly-available, cluster monitoring solution. The entire stack allows to collect an incredible amount of information about physical nodes, containers as well as most of the other cluster components, while featuring easy-to-read and appealing dashboards to present the most relevant data. Finally, it provides timely notifications, to enable fast reactions when a problem occurs.

#### 6) HIGHER LEVEL APPLICATIONS

On top of the infrastructure built out of the different key components enumerated up to now, we installed the main services, besides the actual *CrownLabs* application, directly accessed by the end users. Specifically, the most relevant ones are `Keycloak` and `Nextcloud`. The former is the identity and management solution in charge of providing a seamless single-sign-on authentication and authorization workflow whatever the target service to access. To this end, we also leveraged `oauth2-proxy`, a reverse proxy that, combined with the ingress controller, allows to provide authentication in front of services that do not natively support it. `Nextcloud`, on the other hand, provides each user a personal folder, shared between all her different VMs and easily accessible also through a web-based interface. Yet, the configuration of these two fundamental elements, taking care of aspects such as high-availability and performance optimization, proved to be challenging, given the high degree of under the hood customization necessary to make them work. For instance, besides the actual service, they both required the configuration of both databases (through the `PostgreSQL` operator[28]) and in memory data structure stores (via the `KubeDB` operator[29]).

[27] https://github.com/coreos/kube-prometheus
[28] https://github.com/zalando/postgres-operator
[29] https://github.com/kubedb/operator

## VII. EXPERIMENTAL EVALUATION

To experimentally evaluate the overall performance associated with *CrownLabs*, we conducted an extensive benchmarking campaign. This section presents and discusses the most relevant results, both considering reference scenarios and observing the behavior during a real laboratory. Specifically, the evaluation initially focused on the most relevant cluster metrics, to show how the system reacts when the number of users (and virtual machines) increases. Then, we assessed the delay experienced by the students before their VMs became up and running. Finally, we measured the amount of network bandwidth demanded to access the remote desktop, hence observing the requirements in terms of users' Internet connection.

### A. CLUSTER METRICS

Focusing on the analysis of the cluster metrics, we start considering the outcome of a real remote laboratory accessed by 25 students in parallel: some were working alone, while others cooperated in groups of two. All in all, this scenario is deemed to better represent the real requirements associated with the target use-case of *CrownLabs* compared to simulated configurations. During the laboratory, the students, attending a networking class, were required to experiment with `GNS3`,[30] a graphical software simulating complex networks through the interconnection and configuration of routers, switches and hosts. Additionally, this software interacts with `Wireshark`,[31] to capture and explore the traffic exchanged between the different nodes. The virtual machine image adopted for the laboratory was based on an `xubuntu 19.10` guest O.S., with all the different pieces of software already installed and configured. Each VM was assigned 2 CPUs and 2.5 GB of RAM. Finally, all the results herein presented derive from the metrics scraped by `Prometheus` from the physical servers.

Fig. 4 presents the results of the evaluation, showing the evolution over time of the cluster CPU and memory usage as the number of *CrownLabs* users changed. The laboratory lasted three hours, while the results span 30 additional minutes before and after to account for early and late students. Focusing first on the CPU (Fig. 4b), three different metrics are evaluated. Two stem from the USE method [14] and measure respectively the percentage of time the CPU was busy servicing work (i.e. Utilization) and the amount of work waiting to be performed as measured by CPU load over one minute (i.e. Saturation). In other words, the former measures how busy the CPU is in a certain moment, while the latter accounts for the amount of work awaiting in a kernel queue to be performed (including both CPU and I/O time) over a period of time. The third line, on the other hand, represents the current amount of *reserved* CPU. Indeed, Kubernetes allows to reserve a certain amount of resources (e.g. CPU and memory) to each pod, in order to guarantee their availability

[30] https://www.gns3.com
[31] https://www.wireshark.org

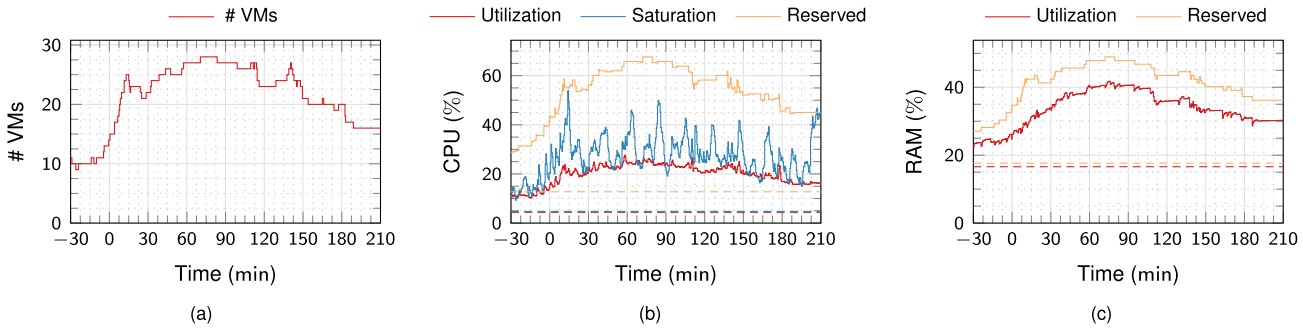


**FIGURE 4.** An evaluation of the most important cluster metrics during a remote networking laboratory accessed by 25 students in parallel, in terms of (a) active VMs, (b) CPU and (c) RAM. The dashed lines represent a baseline computed as the 12 h-average of the corresponding metric when no VMs are present.

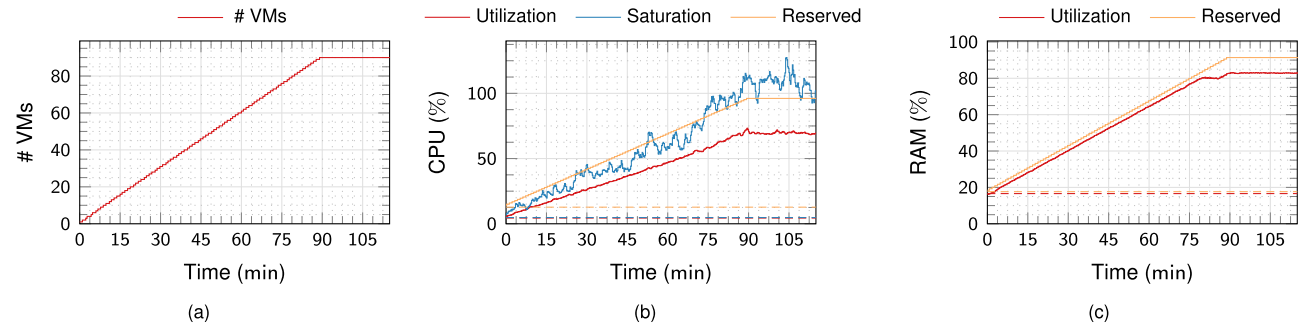


**FIGURE 5.** An evaluation of the most important cluster metrics while simulating an artificial load characterized by an increasing number of VMs, in terms of (a) active VMs, (b) CPU and (c) RAM. The dashed lines represent a baseline computed as the 12 h-average of the corresponding metric when no VMs are present.

and prevent scheduling more tasks than the available computational capacity. During this laboratory, each VM was configured to request exactly the amount of CPU and memory assigned, hence achieving no over-commitment. Finally, the graph is complemented by the dashed baselines corresponding to the three different metrics, measured as the average over 12 h when no virtual machines were running on the cluster (please notice that the average values referring to CPU utilization and saturation graphically overlap). Moving to the RAM evaluation (Fig. 4c), we similarly assessed the amount of memory used and reserved, together with the respective baselines. All in all, the graphs confirmed that the amount of resources consumed by the infrastructural components remained certainly low even in our simple setup, hence leaving much space for the execution of the VMs of the final users. Additionally, given the significant difference between the resources consumed and reserved, it would be possible to increase the number of parallel users through a greater degree of over-commitment.

Starting from this considerations, the evaluation shown in Fig. 5 pushed the infrastructure to its limits, to verify the maximum number of VMs that can be supported concurrently and the overall reaction of the system. Indeed, in this situation we maintained a virtual machine configuration similar to the previous case (same image, 2 CPUs and 2 GB of RAM), while lowering the number of reserved computing resources to 1 CPU only. In other words, we simulated a high degree of over-commitment to better leverage all the available cluster resources (although slightly degrading the user experience in high-load situations). A new virtual machine was started every minute, while simulating for each of them a random CPU load (using the `stress-ng` utility) and RAM usage (with `python`, generating random matrices of the appropriate size). Specifically, the former was characterized by one second long intervals uniformly distributed in the 0–30% band (one CPU only), complemented by high-load spikes (75–100%) on both CPUs and extracted with a 5% probability. The RAM usage, on the other hand, was uniformly drawn between 100–1250 MB, and refreshed every 60 s. According to the above tests, our infrastructure managed to support 90 parallel virtual machines, while still maintaining bearable levels of CPU and RAM utilization. Indeed, also considering the CPU saturation, the load value did not present an exponentially increasing pattern, hence suggesting that all requested work was successfully serviced. All in all, this evaluation confirmed the high degree of consolidation achieved by the system.

### B. VM READY TIME

In this test we assessed the delay experienced by the final users between the creation of a new virtual machine and its availability (i.e. the possibility to access the remote

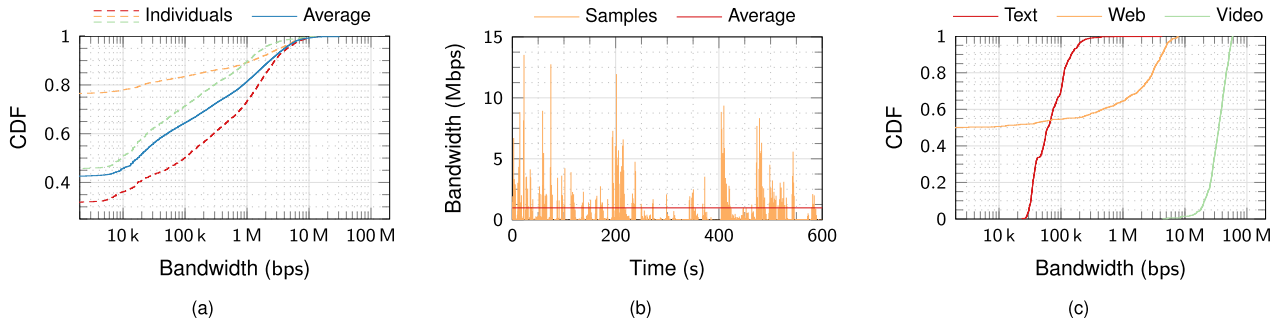


**FIGURE 6.** An evaluation of the downstream bandwidth (from the final user's point of view) consumed by the remote desktop viewer (a) during a networking laboratory, both considering three random individuals — a detailed excerpt of one of them is presented in (b) — and an average over 10 students, and (c) in simulated scenarios: writing a text in LibreOffice Writer, reading news on the Internet and playing a Full HD video (Big Buck Bunny).

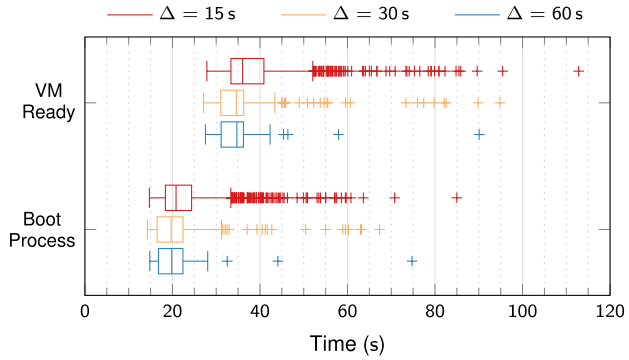


**FIGURE 7.** An evaluation of the time required by *CrownLabs* to spawn a new VM and let the end user connect to the remote desktop. A new instance is created every Δ seconds, to analyze the outcome in different load conditions.

desktop). In other words, this metric includes both the time required by the operator to create the companion resources, the delay introduced by `kubevirt` to schedule the VM and, finally, its boot time. In this scenario, we leveraged a VM image characterized by `xubuntu 20.04 LTS` as guest O.S. (some unnecessary programs and services were removed to speed-up the boot process), 2 CPUs and 2 GB of RAM. The evaluation encompassed the creation of 1000 VMs: a new virtual machine was spawned every Δ seconds, with $\Delta = \{15\,\text{s}, 30\,\text{s}, 60\,\text{s}\}$, while limiting the concurrent number of running VMs to 25.

Fig. 7 summarizes the results of the benchmark, representing the extensions of the quartiles (outliers are shown as individual points). Being the boot times strongly influenced by the guest O.S. configuration, the evaluation is complemented by the time spent completing the boot process. According to the above tests, the user was almost always able to start a new VM and connect to the remote desktop in less than one minute, even when creating a new virtual machine every 15 s. Yet, the main difference emerged between the three situations resides in the degree of dispersion, with the most demanding scenario characterized by a much higher number of outliers. Focusing on the boot process only, it is possible to observe a symmetrical behavior. Hence, showing the important role it played in the ready time variability. Conversely, the impact of the other steps remained mostly constant, requiring on average 15 s up to the start of the boot process.

### C. BANDWIDTH CONSUMPTION

Lastly, our evaluation campaign focused on the network bandwidth demanded by the remote desktop, to evaluate the characteristics of the student's Internet connection necessary to achieve a smooth user experience. To this end, both components related to the remote desktop (i.e. `TigerVNC` and `noVNC`) have been adopted with their out of the box configuration. Yet, a great amount of customization is permitted by both tools in terms of compression algorithms and video parameters. Hence, suggesting the possibility to achieve much better performance in a carefully tuned environment. As for the methodology, all measurements have been obtained directly on the VMs and encompass the remote desktop traffic only. All the results herein presented refer to the downstream bandwidth (from the user point of view), being it dominant compared to the upstream.

In this regard, we first assessed the requirements as measured during the real networking laboratory detailed in Section VII-A. The outcome has been normalized depending on the number of students concurrently accessing the same remote desktop — when two users are connected to the same VM, the bandwidth measured at *CrownLabs* roughly doubles, while it remains the same at the student's end — but it possibly includes the interference associated with the interventions of the instructors. Fig. 6a summarizes the results of the evaluation, displaying the values obtained from three random students, as well as the mean over ten individuals. Overall, it is possible to observe that, on average, about two thirds of the total time is associated with an almost negligible bandwidth demand (i.e. $< 100$ kbps). Indeed, this behaviour is explained considering that network traffic is generated by `noVNC` only when the desktop content varies. On the other hand, the 20 % of the samples is associated with demands greater than 1 Mbps, corresponding to the instants when the desktop is completely redrawn. Yet, different students presented

fairly different behaviors, depending on the amount of time spent actively using the system, reading documentation and searching for information on the Internet. For the sake of completeness, Fig. 6b shows an excerpt of the entire samples associated with one of the students. Overall, it displays the alternation between ''silence periods'' and high-demand spikes, as introduced by the differential video transmission. Yet, the average bandwidth sits in the 1 Mbps area.

Fig. 6c, on the other hand, presents the bandwidth consumed in three different simulated situations. Although not being strictly related to the core use-cases of *CrownLabs*, they aim to show a baseline associated with best and worst case scenarios. Specifically, we started measuring the data transmitted while writing a one page long text in LibreOffice Writer ($\approx$ 200 keystrokes per minute). This scenario was characterized by a very low bandwidth usage (30–300 kbps), thanks to the limited amount of desktop content to be redrawn at each instant. Then, we simulated reading news and searching for information on the Internet, obtaining a CDF shape comparable to those associated with the real networking laboratory. Finally, we examined the most extreme scenario, by remotely playing a full-screen, Full HD video. In a nutshell, it consumed on average 35 Mbps, with peaks up to 60 Mbps. Yet, the playback quality was qualitatively excellent, with only limited differences in terms of smoothness compared to viewing the video locally.

All in all, both evaluations showed that, in normal use-cases, the remote desktop can be accessed achieving a very good user experience with a 10 Mbps Internet connection. Yet, the throughput peaks are short and impulsive, while the average bandwidth consumed is much less. Indeed, the system remained still perfectly usable even in case the available bandwidth was as low as 1 Mbps, with only limited differences in terms of fluidity when causing the refresh of the entire desktop.

## VIII. CONCLUSIONS AND FUTURE WORK

Practical laboratories are an essential component in every computing, networking and security course. Although our students are typically required to physically complete their duties at the university premises, the coronavirus pandemic quickly wiped out what was considered routine just a few days earlier. Indeed, besides the human tragedy accounted by the death toll, the lock-down period proved to be an extreme test bench from an IT point of view.

In this paper, we presented *CrownLabs*, an open-source project started in March 2020 by a group of volunteers from Politecnico di Torino to answer the need for hands-on remote laboratories. *CrownLabs* allows students to instantiate and access personal learning environments already configured with all the software necessary to complete their duties, but remotely executed by a Kubernetes cluster located at the university premises. The entire system has been designed with the educational requirements in mind: both group cooperation and mentoring are encouraged by the ability to seamlessly share the same remote desktop among multiple users. No complex setup or special computing resources are required from the students: a simple web browser is enough to interact with the system and access the remote desktop. To see a live preview of the web-based dashboard, as well as of the entire *CrownLabs*, please refer to the introductory video available on YouTube.[32]

Stemming from the specific use-case, this paper analyzed from a more general perspective the different design requirements and principles that have driven *CrownLabs*. First, we focused on the concept of offloading as many tasks as possible to the infrastructure itself, to speed up the initial development and prevent application obsolescence. Then, we presented the high-level architecture of the system, focusing on the most relevant components, as well as shown how we managed to build a small-scale, ''cloud-grade'' infrastructure by installing and configuring the necessary components from the Kubernetes ecosystem. Finally, we performed an extensive benchmarking campaign to evaluate how the system reacted to different loads both in real and simulated scenarios. All in all, the results are encouraging, both in terms of cluster resources and bandwidth demands. Additionally, the feedback from early adopters proved to be positive, confirming the effectiveness of the system.

As for the future work, besides increasing the user base and improving the usability of the system, we currently plan to focus on three main aspects. First, the introduction of container-based laboratories. Indeed, an initial proof of concept has shown all the potential of this approach to achieve much higher consolidation as well as faster start-up times. Yet, many concerns from the isolation point of view still needs to be better investigated. Second, the support for lightweight and text-only VMs, by providing an easy access to the console both through the web-based dashboard and the traditional `SSH` protocol. Finally, the practical implementation of the ad-hoc federation idea envisioned in Section IV-G, to increase the computational resources without actually setting up new hardware.

## ACKNOWLEDGMENT

The CrownLabs project was started by a group of volunteers, mostly students enrolled in the M.Sc. of Computer Engineering at Politecnico di Torino (Italy), under the pressure of the coronavirus, in March 2020.

The authors would like to thank all the people who greatly contributed to this project: Aldo Lacuku, Andrea Cossio, Francesco Borgogni, Gabriele Filaferro, Giuseppe Ognibene, Hamza Rhaouati, Mattia Lavacca, Michele Luigi Greco, Serena Flocco, Simone Magnani and Stefano Galantino.

## REFERENCES

[1] E. Scanlon, E. Morris, T. di Paolo, and M. Cooper, ''Contemporary approaches to learning science: Technologically-mediated practical work,'' *Stud. Sci. Educ.*, vol. 38, no. 1, pp. 73–114, Jan. 2002.

[32] https://youtu.be/i7fqga7xQv0

[2] The Joint Task Force on Computer Engineering Curricula Association for Computing Machinery (ACM) and IEEE Computer Society. (Dec. 2016). *Computer Engineering Curricula 2016, CE2016*. [Online]. Available: https://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf
[3] M. Roser, H. Ritchie, E. Ortiz-Ospina, and J. Hasell, "Coronavirus pandemic (COVID-19)," *Our World Data*, 2020. [Online]. Available: https://ourworldindata.org/coronavirus
[4] J. Ma and J. V. Nickerson, "Hands-on, simulated, and remote laboratories: A comparative literature review," *ACM Comput. Surv.*, vol. 38, no. 3, p. 7-es, Sep. 2006.
[5] C. Ivica, J. T. Riley, and C. Shubert, "StarHPC—Teaching parallel programming within elastic compute cloud," in *Proc. ITI 31st Int. Conf. Inf. Technol. Interface*, Jun. 2009, pp. 353–356.
[6] L. Xu, D. Huang, and W. Tsai, "Cloud-based virtual laboratory for network security education," *IEEE Trans. Educ.*, vol. 57, no. 3, pp. 145–150, Aug. 2014.
[7] A. C. Caminero, S. Ros, R. Hernández, A. Robles-Gómez, L. Tobarra, and P. J. T. Granjo, "VirTUal remoTe labORatories management system (TUTORES): Using cloud computing to acquire university practical skills," *IEEE Trans. Learn. Technol.*, vol. 9, no. 2, pp. 133–145, Apr. 2016.
[8] *ThoTh Lab V3.0*. Accessed: Jul. 9, 2020. [Online]. Available: https://www.thothlab.com
[9] *Katacoda*. Accessed: Jul. 9, 2020. [Online]. Available: https://katacoda.com
[10] K. Pepple, *Deploying Openstack*. Newton, MA, USA: O'Reilly Media, 2011.
[11] K. Hightower, B. Burns, and J. Beda, *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. Newton, MA, USA: O'Reilly Media, 2017.
[12] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," *J. Object Oriented Program.*, vol. 1, no. 3, pp. 26–49, Aug. 1988.
[13] B. Ibryam and R. Huß, *Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications*. Newton, MA, USA: O'Reilly Media, 2019.
[14] B. Gregg, "Thinking methodically about performance," *Commun. ACM*, vol. 56, no. 2, pp. 45–51, Feb. 2013.

**MARCO IORIO** (Graduate Student Member, IEEE) received the M.Sc. degree in computer engineering from the Politecnico di Torino, Italy, in 2018, where he is currently pursuing the Ph.D. degree. His research interests include vehicular networks, cooperative driving, and cybersecurity.

**ALEX PALESANDRO** received the M.Sc. degree in computer engineering from the Politecnico di Torino, Italy, in 2014, and the Ph.D. degree from the University of Lyon, in 2018. He is currently a Postdoctoral Researcher with the Politecnico di Torino. His research interests include cooperative multi-cloud systems, edge/fog computing, and network functions virtualization.

**FULVIO RISSO** (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer engineering from the Politecnico di Torino, Italy, in 1995 and 2000, respectively. He is currently an Associate Professor with the Politecnico di Torino. He has coauthored more than 130 scientific articles. His research interests include high-speed and flexible network processing, edge/fog computing, software-defined networks, and network functions virtualization.