Lab Orchestrator

Marco Schlicht Mohamed El Jemai

June 10, 2021

Abstract

Explanation of tools that we are using.

Contents

1 Introduction

1.1 Motivation

At the university, lecturers can simply provide their students with a VM in which the students can complete their assignments. In these VMs, software is pre-installed and pre-configured so that the students can, for example, directly start programming their microcontroller with the IDE provided. This has the advantage for the instructors that all students use the same system and therefore they only have to provide support for this system and do not have to worry about problems that vary from system to system. Also, the VM forms a sandbox and thus changes can be made to the system in this environment and if something breaks, a snapshot is taken or the original image is reinstalled.

However, the options here are limited to one VM and local deployment. It is not possible to simply start a whole network of VMs, nor is it possible to open these VMs in the browser.

1.2 Description

The Lab Orchestrator shall allow to start a network of virtualised systems (i.e. VMs and different types of containers) and make them accessible over the network. In the network of virtualised systems several virtualised systems shall run simultaneously and if a user has access to one of the systems it shall be possible to access others. Several such networks should be able to be started, so that users can work independently of each other. A user has a user session and in this session a network is started for this user, in which the user can work.

The access to the virtualised systems should be possible via an integration in the web. The user should be able to start a network on a web page and then get access to the graphical user interface or the terminal of the virtualised systems in a frame or HTML canvas on the web page.

Furthermore, in addition to the integration as a frame or canvas, it should be possible to optionally integrate a tutorial. These instructions should be able to contain several pages and several steps per page and teach the person working in the virtualised system. The instructions contain various features, such as steps that can be checked to see the progress and tutorial boxes that provide knowledge and explain certain parts. These tutorials are meant to extend the mere sandbox to be able to teach people something.

As virtualised systems, we want to support docker containers and classic VMs.

1.3 Target Groups

Target Groups:

- Universities
- Computer Science Clubs
- Companies
- IT Security Personal

- Learning Platforms
- Moodle

The software can be used in universities by lecturers to provide students with an environment in which they can learn and try out. On the one hand, it can be used parallel to lectures or practice sessions as a pure sandbox of a network in which students can do their exercises, and on the other hand, the tutorials can be integrated directly into the application.

Computer science clubs like the CCC often do Capture the Flag competitions. The program can make it easier to scale scenarios for competitions and learning.

Companies and private individuals can use the tool to map their internal IT environment and safely check their environment for security vulnerabilities in a sandbox. There is no need to consider any consequences for the live operation of the company.

IT security personnel also benefit from the sandbox environment and can be trained here or acquire their own knowledge. Although solutions such as Hack The Box already exist for this purpose, they cannot be hosted by the company itself and no instructions are available for them either.

Learning platforms can build on the program to create tutorials. Also an extension for Moodle, which is used by many universities, is conceivable, in which the courses of the application are integrated. One could use such a Moodle addon to work within Moodle in VMs and store tasks for students there.

1.4 Project Planning

The Lab Orchestrator is divided into several parts:

- Orchestrator of virtualised systems
- Accessible from the Web Base Images
- Lab Orchestrator Library
- REST-API

1.4.1 Orchestrator

Kubernetes is suitable as an orchestrator. Kubernetes allows you to launch a predefined set of Docker images. The images in a namespace can be connected to each other and ports can be opened to the outside. In Kubernetes' declarative YAML config, it is possible to define a set of Docker images, in addition to defining the ports opened to the outside and creating an internal network. This allows to access one container from another container. With such a config, it is easy to start and stop this set of containers as often as you want.

Kubernetes out of the box can only start containers. Here it is unclear whether the containers are sufficient to start graphical interfaces of Windows. The KubeVirt extension adds the function to use VMs instead of containers for this. KubeVirt can also be used to start a Windows VM with a graphical user interface. Kubernetes with KubeVirt therefore seems suitable as an orchestrator for the Lab Orchestrator .

1.4.2 Accessible from the Web Base Images

In order to be able to access the virtualised systems from the web and to make it as easy as possible to create your own virtualised systems, a technology must be found that makes it possible to access the terminal or the graphical interface of the virtualised system. This technology should then be provided in a template for example a docker base image or an virtual machine image both with the tool preinstalled.

For terminal access, there are already various tools, such as Gotty, wetty and ttyd. For desktop access, there is Apache Guacamole and noVNC. It is necessary to evaluate which of these tools are the most suitable and then install and configure these tools in the base image.

The goal of this step is to have an runnable image where the graphical interface and the terminal of the VM or Docker container can be accessed via the web.

Then, this image must be included in a Kubernetes template so that a network of such virtualised systems can be launched in Kubernetes.

With this template, it must also be tested how it is possible to access it with multiple users. This will probably require a proxy that authorizes certain requests and forwards them to the respective containers. It must be evaluated how the authentication and the routing works. One possibility would be to include a token in the URL. This may work with Kubernetes out of the box, but if that is not enough there are other possibilities. Traefik for example is a dynamic proxy that automatically detects new services in Docker and integrates them for routing. Consul is another tool for discovering services. Traefik can interact with Consul and Consul can report the new routes to Traefik. The best solution here would be one where Traefik directly detects the routes from Kubernetes, similar to how it already works with Docker in Traefik. If that is not possible we either need to be able to add new routes via Traefik's API or include Consul. Anyway, with Consul it is possible to insert a dynamic configuration of routes afterwards. The insertion of new routes should only be tested manually in this step and then automated in the Lab Orchestrator Library. If this concept does not work with Traefik and Consul, we have to find another proxy possibility, program a new one or extend an already existing one with this function.

1.4.3 Lab Orchestrator Library

The library is the core of the project and will be provided as a Python library. A network of virtualised systems base images is called a lab. The library should be able to manage the virtualised systems base images in Kubernetes and provide an interface to manage labs.

In the requirements list, "must" stands for that it has to be implemented, "should" is an optional requirement that should be implemented and "may" is an optional requirement that may be implemented.

Requirements for the library are:

- start and stop labs (must)
- pause and continue labs (should)

- add and remove labs (must)
- configuration of routing labs (must)
- authentication during routing (must)
- show authentication details in labs, e.g. login credentials (must)
- link users to their labs (must)
- add instructions (must)
- link labs and instructions (must)

Requirements for the instructions:

- Markdown or HTML syntax (should)
- pages with text (must)
- controller to select a virtualised system (must)
- steps per page (may)
- tick of steps (may)
- progress bar (may)
- progress bar for ticked steps (may)
- embed images and other media (should)
- present knowledge texts (must)
- interactions with virtualised systems, e.g. copy a text into the clipboard of a system (may)
- variables (may)

There is a Kubernetes client library for Python. This library can be used in the Lab Orchestrator to get access to the Kubernetes API and interact with Kubernetes.

Core functionalities of the library are start, stop, add and remove labs. To start and stop, the previously created template must be mapped into the Kubernetes Client Library and some settings such as the namespace must be kept variable. The Client Library can then be used to start and stop the templates. The configuration of the templates must be stored in a database and contain among other things the access token, the user ID and the specific template configuration like the namespace which is used.

For adding new labs, it is intended that one provides a path to the images of the VMs or, in the case of Docker, optionally a link to the image in a container registry. The specified VMs or containers are then added to the template and must have the respective terminal/desktop web solution integrated and properly configured. Additional Kubernetes configuration can also be entered here. The configuration is then stored in the database. To remove a lab, only the configuration then needs to be deleted from the database.

Pause and continue labs would be a useful extension, which, however, is not mandatory for the first version.

Depending on the routing and authentication solution from the previous step, the proxy must still be told how to use the ports of the VMs or containers when a lab is started. If the Kubernetes native solutions doesn't work, either the Traefik API or Consul must be used. These routing settings must be included in the response at startup so that users know which URL they can use to access it. There must also be a possibility to select the different containers in the URL.

An authorization who can start labs is not provided here and comes in the web interface with a proper user management. That means, everyone who uses this library can do everything in the code and must add an authorization layer, if certain labs are to be started only by certain users.

To link the users with their labs it is sufficient to store a user ID and optionally a name for the user, which can optionally be included in the instructions via variables.

The instructions are only texts, which have to be stored in the database. Several pages can be stored in different database entries or the complete instruction of a course can be stored in one database entry. These are then linked to the respective lab template. All but four features of the instructions are requirements to be implemented in the web interface and are simply different representations of the text. The controller for selecting the virtualised system can include the URLs from the response at startup, as links for the frame. So that individual steps can be checked off, another database table must be added under circumstances, which stores the status. Variables could be queried via an extra function and then also composed by the web interface. For the interaction with the virtualised system, existing solutions can be searched for or an interface for this must be integrated in the virtualised system. The simplest possibility for this would be to offer a service via an internal web interface in the virtualised system, which copies texts into the clipboard.

It must also be evaluated whether the library should write data to a database on its own or only return the data that needs to be saved as a response. The former would be more trivial to use and a SQLite database would be a good choice. The second would provide more flexibility and it would be easier to integrate into Django.

1.4.4 REST-API

The library alone offers the advantage that you can easily write programs that use this concept. For example, you can include the library in a Django app or in desktop software. Another use case for the library would be a web interface to control the library. This way you can include the library in a microservices system or you can access it from other programming languages and thus include it in many other non-Python projects. The web interface will be a REST-API and will be implemented with Django or Flask.

The library will not yet have authentication to start labs, only authorization when accessing the labs. In the web interface the library will be extended by a permission system and user management. Otherwise, the web interface only has to offer the functionalities of the library via REST.

1.5 Milestones

1.5.1 Prototype

First we need to develop a prototype that proves that the idea of labs is working with Kubernetes.

1. Install Kubernetes and KubeVirt
2. Understand basics of Kubernetes and Kubernetes templates

3. Understand how to start and stop Kubernetes templates, base images and VMs
4. Evaluation of web-terminal and web-vnc tools
5. Integration of web-terminal and web-vnc tools into base images and VMs
6. Integrate base images and VMs into Kubernetes templates
7. Evaluate and implement a routing solution
8. Add multi-user support to the routing solution

In this step Kubernetes and maybe KubeVirt will be installed and configured. We will take a look at Kubernetes templates and base images and how they can be started and stopped in Kubernetes. This is the basic knowledge we need to build the application.

After that, we will evaluate which web-terminal tool and which web-vnc tool is the most suitable for using in the base images. And afterwards this tools will be integrated into docker base images or VM base images. These will be the basis of the labs.

The base images will be integrated into a Kubernetes template and combined with a basic routing solution. After that works the routing will be extended to support multi-user labs.

If all that steps work, the prototype will be a success. This proves, that the orchestrator can be implemented with Kubernetes and the given web accessible base images.

1.5.2 Alpha Phase

Then in the alpha phase the Lab Orchestrator library will be implemented.

1. Start and stop labs
2. Automatically add routing and authorization
3. Add and remove labs
4. Add and remove instructions
5. Link users to labs
6. Link instructions to labs

At the end of the alpha phase we have a working solution as library, that fulfills the minimal needed set of requirements. This library can than be used in other project for example the REST-API.

1.5.3 Beta Phase

In the Beta Phase we will add the REST-API and add the remaining optional features.

1. Implement a REST-API that is able to use the library to start and stop labs
2. Add user-management and permission system to the REST-API
3. Add remaining features of the instructions
4. Pause and continue labs

After the beta phase has succeeded, the project is considered finished and can be released to the public.

2 Basics

The Lab Orchestrator application uses different tools that may be explained before the installation of the application. This chapter will give you an introduction into the tools that are used and required in this project, as well as an explanation about Kubernetes that is needed to understand how the Lab Orchestrator application is working on the inside.

2.1 Generating The Documentation

The documentation is written in markdown and converted to a pdf using pandoc. To generate the documentation pandoc and latex are used. Install `pandoc`, `pandoc-citeproc` and a latex environment: [1]

```
$ sudo apt install pandoc pandoc-citeproc make
$ make docs
```

For the replacement of variables there is a lua script installed, so you need to install lua too. [2]

After that, you need to install the pandoc-include-code filter. For this, you need to download the latest release from here: github.com/owickstrom/pandoc-include-code/releases[1]. Then extract the tar file and install it with the command `install pandoc-include-code ~/.local/bin`. Also make sure `~/.local/bin` is included in `$PATH`.

There is a make command to generate the docs: `make docs`.

2.2 Terminal Tools

2.2.1 Make

Make is used to resolve dependencies during a build process. In this project make is used to have some shortcuts for complex build commands. For example there is a make command to generate the documentation: `make docs`.

2.2.2 nohup

If a terminal is closed (for example if you logout), a HUP signal is send to all programs that are running in the terminal. [3] `nohup` is a command that executes a program, with intercepting the HUP signal. That results into the program doesn't exit when you logout. The output of the program is redirected into the file `nohup.out` `nohup` can be used with `&` to start a program in background that continues to run after logout. [4]

2.3 Kubernetes

Kubernetes is an open source container orchestration platform. With Kubernetes it's possible to automate deployments and easily scale containers. It has many features that make it useful for the project. Some of them are explained here. [5]

[1] https://github.com/owickstrom/pandoc-include-code/releases

2.3.1 Control Plane

The control plane controls the Kubernetes cluster. It also has an API that can be used with kubectl or REST calls to deploy stuff. [6]

2.3.2 Custom Resource

In Kubernetes it's possible to extend the Kubernetes API with so called custom resources (CR). A custom resource definition (CRD) defines the CR. [7]

2.3.3 Kubernetes Objects

Kubernetes Objects have specs and a status. The `spec` is the desired state the object should have. `status` is the current state of the object. You have to set the `spec` when you create an object.

Kubernetes objects are often described in yaml files. The required fields for Kubernetes objects are: [8] - `apiVersion`: Which version of the Kubernetes API you are using to create this object - `kind`: What kind of object you want to create - `metadata`: Helps to uniquely identify the object - `spec`: The desired state of the object

2.3.4 Pods

A pod is a group of one or more containers that are deployed to a single node. The containers in a pod share an ip address and a hostname.

2.3.5 Deployment

Deployments define the applications life cycle, for example which images to use, the number of pods and how to update them. [9]

2.3.6 Services

Services allows that service requests are automatically redirected to the correct pod. Services gets their own IP addresses that is used by the service proxy.

Services also allow to add multiple ports to one service. When using multiple ports, you must give all of them a name. For example you can add a port for http and another port for https.

Example of a Service:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

This service has the name `my-service` and listens on the port 80. It forwards the requests to the pods with the selector `app=MyApp` on the port 9376.

There is also the ability to publish services. To make use of this, the `ServiceType` must be changed. The default `ServiceType` is `ClusterIP`, which exposes the service on a cluster-internal IP, that makes this service only reachable from within the cluster. One other service type is `ExternalName`, that creates a CNAME record for this service. Other Types are `NodePort` and `LoadBalancer`. [10]

You should create a service before its corresponding deployments or pods. When Kubernetes starts a container, it provides environment variables pointing to all the services which were running when the container was started. These environment variables has the naming schema `servicename_SERVICE_HOST` and `servicename_SERVICE_PORT`, so for example if your service name is foo: [11]

```
FOO_SERVICE_HOST=<the host the Service is running on>
FOO_SERVICE_PORT=<the port the Service is running on>
```

You can also use Ingress to publish services.

2.3.7 Ingress

An ingress allows you to publish services. It acts as entrypoint for a cluster and allows to expose multiple services under the same IP address. [10]

With ingresses it's possible to route traffic from outside of the cluster into services within the cluster. It also provides externally-reachable URLs, load balancing and SSL termination.

Ingresses are made to expose http and https and no other ports. So exposing other than http or https should use services with a service type `NodePort` or `LoadBalancer`.

Ingresses allows to match specific hosts only and you can include multiple services in an ingress by separating them with a path in the URL. [12]

Example:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
```

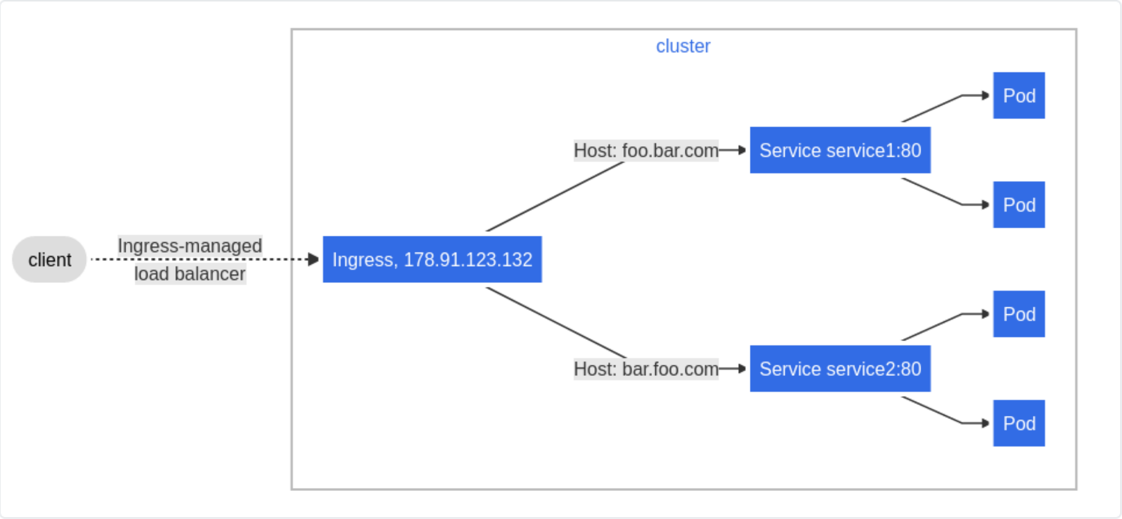

Figure 1: How Ingress interacts with Services and Pods [12]

```
        backend:
          service:
            name: service1
            port:
              number: 4200
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 8080
```

To use ingresses you need to have an ingress controller.

2.3.8 Ingress Controllers

Ingress controllers are responsible for fulfilling the ingress.

Examples of ingress controllers are: ingress-nginx[2] and Traefik Kubernetes Ingress provider[3].

2.3.9 Namespaces

Namespaces allows you to run multiple virtual clusters backed by the same physical cluster. They can be used when many users across multiple teams or projects use the same cluster.

Namespaces provide a scope for names. Names of resources need to be unique within

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

[3] https://doc.traefik.io/traefik/providers/kubernetes-ingress/

a namespace, but not across namespaces. Namespaces are also a way to divide cluster resources between multiple users.

Namespaces may be useful to separate the networks of individual users.

2.3.10 Network Policies

With Network Policies it is possible to control the traffic flow at the ip address or port level. It allows you to specify how a pod is allowed to communicate with various network entities over the network. This can be useful to separate the networks of individual users. [13]

2.3.11 Config Maps and Secrets

A ConfigMap is an an API object to store configuration in key-value pairs. They can be used in pods as environment variables, command-line arguments or as a configuration file. [14]

Secrets does the same, but for sensitive information. They are by default unencrypted base64-encoded and can be retrieved as plain text by anyone with api access. But it's possible to enable encryption and RBAC (role based access control) rules. [15]

2.4 Kubernetes Tools

2.4.1 kubectl

`kubectl` is a command line tool that lets you control Kubernetes clusters. It can be used to deploy applications, inspect and manage cluster resources and view logs. [16]

2.4.2 kind and minikube

`kind` is used to deploy a local Kubernetes cluster in docker.

`minikube` is used to deploy a local Kubernetes cluster that only runs one node.

Both tools are used to get started with Kubernetes, to try out stuff and for daily development. To run Kubernetes in production you should install other solutions or use cloud infrastructure. [16]

In this project we use minikube for development.

2.4.3 Helm, Krew, KubeVirt Virtctl and Rancher

Helm is a package manager for Kubernetes. **Krew** is a package manager for kubectl plugins. **KubeVirt** enables Kubernetes to use virtual machines instead of containers. And **Virtctl** is a kubectl plugin to use KubeVirt with kubectl. Virtctl adds some commands for example to get access to a VMs console.

There is a tool called Containerized Data Importer (CDI) that is designed to import Virtual Machine images for use with KubeVirt. [17]

Rancher is an Web UI for Kubernetes, that can display all running resources and allows an admin to change them and create new. Maybe this is worth a look.

2.5 Web-Terminal Tools

There are several tools available to get access to a terminal over a website. Gotty, wetty and ttyd are examples of this. These tools start a terminal session and then allows a user to access this session over a website.

2.6 Web-VNC Tools

To connect to a VNC session of a virtualised system, there are also several tools. To name two of them, there are Apache Guacamole and noVNC. These tools start a VNC session and then allows a user to access this session over a website.

3 Installation

3.1 Prerequisites

3.1.1 Kubernetes Development Installation

To run Lab Orchestrator you need an instance of Kubernetes. If you want to use VMs instead of containers you additionally need to install KubeVirt.

For development we use minikube. To install minikube install docker and kvm2[4] or some other driver for VMs and follow this guide[5]. Also install `kubectl` using this guide[6].

After the installation you should be able to start minikube with the command `minikube start --driver kvm2` and get access to the cluster with `kubectl get po -A`. The command `minikube dashboard` starts a dashboard, where you can inspect your cluster on a local website. If you like you can start it with this command in the background: `nohup minikube dashboard >/dev/null 2>/dev/null &`, but then it's only possible to stop the dashboard by stopping minikube with `minikube stop`.

You can start one cluster with docker `minikube start --driver=docker -p docker` and a second cluster with `minikube start -p kubevirt --driver=kvm2`. You should now see both profiles running with `minikube profile list`. This may be helpful for testing. [18]

`kubectl` is now configured to use more than one cluster. There should be two contexts in `kubectl config view`: `docker` and `kubevirt`. You can use the minikube kubectl command like this to specify which cluster you would like to use: `minikube kubectl get pods -p docker` and `minikube kubectl get vms -p kubevirt`. Or you can specify the context in kubectl like this: `kubectl get pods --context docker` and `kubectl get vms --context kubevirt`.

You can stop them with `minikube stop -p docker` and `minikube stop -p kubevirt`. Deleting them works with the commands `minikube delete -p docker` and `minikube delete -p kubevirt`.

It is sufficient to only run one cluster with kvm2 driver, because this can execute docker as well.

3.1.2 Kubernetes Productive Installation

3.1.3 Helm, Krew, KubeVirt and Virtctl Installation

Start minikube with kvm2 driver: `minikube start --driver kvm2`.

Install Helm using this guide[7].

Install Krew using this guide[8].

[4] https://minikube.sigs.k8s.io/docs/drivers/kvm2/

[5] https://minikube.sigs.k8s.io/docs/start/

[6] https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/

[7] https://helm.sh/docs/intro/install/

[8] https://krew.sigs.k8s.io/docs/user-guide/setup/install/

If you are running Minikube, use this installation guide to install KubeVirt and then Virtctl with Krew: KubeVirt quickstart with Minikube[9]. This adds some commands to kubectl for example `kubectl get vms` instead of `kubectl get pods`.

Start kubevirt in the kubevirt cluster: `minikube addon enable`. After that deploy a test VM using this guide: Use KubeVirt[10]

3.2 Lab Orchestrator Installation

[9] https://kubevirt.io/quickstart_minikube/

[10] https://kubevirt.io/labs/kubernetes/lab1

4 Prototype

4.1 KubeVirt and Virtual Machines

You should have installed kubectl and minikube with activated kubevirt addon.

KubeVirt has a tool called Containerized Data Importer (CDI), which is designed to import Virtual Machine images for use with KubeVirt. This needs to be installed from here: Containerized Data Importer (CDI)[11].

The installation of KubeVirt and CDI adds several new CRs, which can be found in the documentation of kubevirt[12].

Resources from Kubernetes:

- PersistentVolume (PV)
  - already included in Kubernetes
- PersistentVolumeClaim (PVC)
  - already included in Kubernetes

CRs from KubeVirt:

- VirtualMachine (VM)
  - An image of an VM, e.g. Fedora 23
  - Can only be started once
- VirtualMachineInstance (VMI)
  - An instance of an VM, e.g. the Lab i'm currently using
- VirtualMachineSnapshot
- VirtualMachineSnapshotContent
- VirtualMachineRestore
- VirtualMachineInstanceMigration
- VirtualMachineInstanceReplicaSet
- VirtualMachineInstancePreset

CRs from CDI:

- StorageProfile
- Containerized Data Importer (CDI)
  - converts an VM image into the correct format to use it as VM in KubeVirt
- CDIConfig
- DataVolume (DV)
- ObjectTransfer

4.1.1 KubeVirt Basics

There is an example vm config in the KubeVirt documentation. [19] Download the vm config `wget https://raw.githubusercontent.com/kubevirt/kubevirt.github.io/master/labs/manife` and apply it: `kubectl apply -f vm.yaml`. Now you should see, that there is a new VM in `kubectl get vms` called testvm. You can start the VM with `kubectl virt start`

[11] https://kubevirt.io/labs/kubernetes/lab2.html

[12] https://kubevirt.io/user-guide/

testvm. This creates a new VM instance (VMI) that you can see in kubectl get vmis. You can then connect to the serial console using kubectl virt console testvm. Exit the console with ctrl+] and stop the VM with kubectl virt stop testvm. Stoping the VM deletes all changes made inside the VM and when you start it again, a new instance is created without the changes. You can start a VM only once.

When a VM gets started, its status.created attibutes becomes true. If the VM instance is ready, status.ready becomes true too. When the VM gets stopped, the attributes gets removed. A VM will never restart a VMI until the current instance is deleted. [20]

After starting the VM you can expose its ssh port with this command: kubectl virt expose vm testvm --name vmiservice --port 27017 --target-port 22. Then you can get the cluster-ip from kubectl get svc. The cluster ip can't be used directly to connect with ssh, but from inside minikube. So to connect to the ssh of the VM execute minikube ssh. This logs you in to the minikube environment. From there you can execute the corresponding ssh command, e.g. ssh -p 27017 cirros@10.102.92.133. [20]

VMIs can be paused and unpaused with the commands kubectl virt pause vm testvm or kubectl virt pause vmi testvm and the commands kubectl virt unpause vm testvm or kubectl virt unpause vmi testvm. This freezes the process of the VMI, that means that the VMI has no longer access to CPU and I/O but the memory will stay allocated. [21]

Example VM (prototype/vm.yaml): [19]

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: testvm
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/size: small
        kubevirt.io/domain: testvm
    spec:
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
            - name: cloudinitdisk
              disk:
                bus: virtio
          interfaces:
          - name: default
```

```
            bridge: {}
        resources:
          requests:
            memory: 64M
      networks:
      - name: default
        pod: {}
      volumes:
        - name: containerdisk
          containerDisk:
            image: quay.io/kubevirt/cirros-container-disk-demo
        - name: cloudinitdisk
          cloudInitNoCloud:
            userDataBase64: SGkuXG4=
```

4.1.2 KubeVirt Run Strategies

VirtualMachines have different so called run strategies. If a VMI crashes it restarts if you set `spec.running: true`, but by defining a `spec.RunStrategy` this behaviour can be changed. You can only use `spec.running` or `spec.RunStrategy` and not both at the same time. There are four run strategies: [22]

- Always: If the VMI crashes, a new one is created. It's the same as setting `spec.running: true`
- RerunOnFailure: VMI restarts, if the previous failed in an error state. It will not be re-created if the guest stopped it.
- Manual: It doesn't restart until someone starts it manually.
- Halted: This means, the VMI is stopped. It's the same as setting `spec.running: false`

4.1.3 KubeVirt Presets

`VirtualMachineInstancePreset` is a resource that can be used to create re-usable settings that can be applied to various machines. These presets work like the `PodPreset` resource from Kubernetes. They are namespaces, so if you need to add these presets to every namespace where you need it. Any domain structure can be added in the `spec` of a preset, for example memory, disks and network interfaces. The presets uses `Labels` and `Selectors` to determine which VMI is affected from the preset. If you don't add any selector, the preset will be applied to all VMIs in the namespace. [23]

You can use presets to define a set of specs with different values and give them labels and then customise VMIs with them. This abstracts some of the specs of VMIs and make it easily customisable to change the specs of a VMI. [23]

Example `VirtualMachineInstancePreset`: [23]

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstancePreset
metadata:
  name: small-qemu
spec:
  selector:
    matchLabels:
      kubevirt.io/size: small
  domain:
    resources:
      requests:
        memory: 64M
```

Example VMI, that matches the correct labels: [23]

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
version: v1
metadata:
  name: myvmi
  labels:
    kubevirt.io/size: small
```

The example shows a preset, which applies 64M of memory to every VMI with the label `kubevirt.io/size: small`. [23]

When a preset and a VMI define the same specs but with different values there is a collision. Collisions are handled in the way that the VMI settings override the presets settings. If there are collisions between two presets that are applied to the same VMI an error occurs. [23]

If you change a preset it is only applied to new created VMIs. Old VMIs doesn't change. [23]

4.1.4 KubeVirt Disks and Volumes

4.1.4.1 Disks

Disks are like virtual disks to the VM. They can for example be mounted from inside /dev. Disks are specified in `spec.domain.devices.disks` and need to reference the name of a volume. [24]

Possible disk types are: `lun`, `disk` and `cdrom`. `disk` is an ordinary disk to the VM. `lun` is a disk that uses iCSI commands. And `cdrom` is exposed as a cdrom drive and read-only by default. [24]

Disks have a bus type. A bus type indicates the type of disk device to emulate. Possible types are: `virtio`, `sata`, `scsi`, `ide`. [25]

4.1.4.2 Volumes

Volumes are a Kubernetes Concept. They try to solve the problem of ephemeral disks. Without volumes, if a container restarts, it restarts with a clean state and it's not possible to save any state. Volumes allows to have a disk attached, that is persistent. There are ephemeral and persistent volumes. Ephemeral volumes have the same lifetime as a pod. Persistent volumes aren't deleted. For both of them in a given pod, data is preserved across container restarts. [26]

In the context of KubeVirt, volumes define the KubeVirts type of the disk. For example you can make them persistent in your cluster or even store them in a container image registry. [24]

Possible disk types are: `cloudInitNoCloud`, `cloudInitConfigDrive`, `persistentVolumeClaim`, `persistentVolumeClaim`, `dataVolume`, `ephemeral`, `containerDisk`, `emptyDisk`, `hostDisk`, `configMap`, `secret`, `serviceAccount`, `downwardMetrics`. [24]

4.1.4.3 cloudInitNoCloud

`cloudInitNoCloud` can be used to attach some user-data to the VM, if the VM contains a proper cloud-init setup. The NoCloud data will be added as a disk to the VMI. This can be used for example to automatically put an ssh key into `~/.ssh/authorized_keys`. For more information see the cloudinit nocloud documentation[13] or the KubeVirt cloudInitNoCloud documentation[14]. [24]

4.1.4.4 Persistent Volumes and Persistent Volume Claims

Kubernetes provides some resources for providing persistent storage. The first is a `PersistentVolume`. A `PersistentVolume` is a piece of storage in the cluster that is reserved from a cluster administrator or it is dynamically provisioned using Storage Classes. [27] A `StorageClass` is the second resource and it is a way for administrators to customize the types of storage the offer. [28] You can read more about `StorageClass` and `PersistentVolume` in the Kubernetes documentation about Storage Classes[15] and PersistentVolumes[16].

A `PersistentVolumeClaim` (PVC) is the third resource provided by Kubernetes. It is a request for storage by a user. In KubeVirt it is used, when the VMIs disk needs to persist after the VM terminates. This makes the VM data persistent between restarts. `PersistentVolumes` and `StorageClasses` can be used to customize the Storage that can be provided to PVCs. [24]

Example of VMI with PVC: [24]

```
metadata:
  name: testvmi-pvc
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
spec:
  domain:
    resources:
```

[13] http://cloudinit.readthedocs.io/en/latest/topics/datasources/nocloud.html

[14] https://kubevirt.io/user-guide/virtual_machines/disks_and_volumes/#cloudinitnocloud

[15] https://kubernetes.io/docs/concepts/storage/storage-classes/#local

[16] https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims

```
   requests:
    memory: 64M
  devices:
   disks:
   - name: mypvcdisk
    lun: {}
 volumes:
  - name: mypvcdisk
   persistentVolumeClaim:
    claimName: mypvc
```

This examples creates a VMI and attaches a PVC with the name `mypvc` as a lun disk.

4.1.4.5 Data Volumes

`dataVolume` are part of the Containerized Data Importer (CDI) which need to be installed separately. A data volume is used to automate importing VM disks onto PVCs. Without a `DataVolume`, users have to prepare a PVC with a disk image before assigning it to a VM. DataVolumes are defined in the VM spec by adding the attribute list `dataVolumeTemplates`. The specs of a data volume contain a `source` and `pvc` attribute. `source` describes where to find the disk image. `pvc` describes which specs the PVC that is created should have. An example can be found here[17]. When the VM is deleted, the PVC ist deleted as well. When a VM manifest is posted to the cluster (for example with a yaml config), the PVC is created directly before the VM is even started. That may be used for performance improvements when starting a VM. It is possible to attach a data volume while creating a VMI, but then the data volume is not tied to the life-cycle of the VMI. [24]

4.1.4.6 Container Disks

`containerDisk` is a volume that references a docker image. The disks are pulled from the container registry and reside on the local node. It is an ephemeral storage device and can be used by multiple VMIs. This makes them an ideal tool for users who want to replicate a large number of VMs that do not require persistent data. They are often used in `VirtualMachineInstanceReplicaSet`. They are not a good solution if you need persistent root disks across VM restarts. Container disks are file based and therefore cannot be attached as a lun device. [24]

To use container disks you need to create a docker image which contains the VMI disk. The disk must be placed into the `/disk` directory of the container and must be readable for the user with the UID 107 (qemu). The format of the VMI disk must be raw or qcow2. The base image of the docker image should be based on `scratch` and no other content except the image is required. [24]

Dockerfile example with local qcow2 image: [24]

```
FROM scratch
```

[17] https://kubevirt.io/user-guide/virtual_machines/disks_and_volumes/#datavolume-vm-behavior

```
ADD --chown=107:107 fedora25.qcow2 /disk/
END
```

Dockerfile example with remote qcow2 image: [24]

```
FROM scratch
ADD --chown=107:107 https://cloud.centos.org/centos/7/images/CentOS-7-
x86_64-GenericCloud.qcow2 /disk/
END
```

The dockerfiles can then be build with `docker build -t example/example:latest .` and pushed to a remote docker container registry with `docker push example/example:latest`. [24]

4.1.4.7 Empty Disks and Ephemeral Disks

`emptyDisk` is a temporary disk which shares the VMIs lifecycle. The disk lifes as long as the VM, so it will persist between reboots and will be deleted when the VM is deleted. You need to specify the `capacity`. [24]

`ephemeral` is also a temporary disk, but it wraps around `PersistentVolumeClaims`. It is mounted as read-only network volume. An ephemeral volume is never mutated, instead all writes are stored on the ephemeral image which exists locally. The local image is created when a VM starts and it is deleted when the VM stops. They are useful when persistence is not needed. [24]

The difference between `ephemeral` and `emptyDisk` is, that `ephemeral` disks are read only and there is only a small space for application data. Also the application data is deleted, when the VM reboots. This can cause problem to some applications and then it's useful to use `emptyDisks`. [24]

4.1.4.8 Remaining Volumes

`hostDisk`, `configMap`, `secrets` and the other volumes are explained in the KubeVirt Disks and Volumes Documentation[18].

4.1.5 KubeVirt Interfaces and Networks

There are two parts needed to connect a VM to a network. First there is the interface that is a virtual network interface of a virtual machine and second there is the network which connects VMs to logical or physical devices.

Networks need unique names and a type. There are two fields in a network. The first field is `pod`. A pod network is the default `eth0` interface. [29] And the second field is Multus. Multus enables attaching a secondary interface that enables multiple network interfaces in Kubernetes. To be able to use multus it needs to be installed separately. [30]

Interfaces describe the properties of a virtual interface and are seen inside the quest instance. They are defined in `spec.domain.devices.interfaces`. You can specify its

[18] https://kubevirt.io/user-guide/virtual_machines/disks_and_volumes/

type by adding the type with curly brackets (`masquerade: {}`). Available types are `bridge`, `slirp`, `sriov` and `masquerade`. Other properties that you can change are `model`, `macAddress`, `ports` and `pciAddress`. Custom mac addresses are not always supported.

You can read more about the types here[19]

Example network and interface:

```
kind: VM
spec:
  domain:
    devices:
      interfaces:
        - name: default
          masquerade: {}
          ports:
           - name: http
             port: 80
  networks:
  - name: default
    pod: {}
```

The ports field can be used to limit the ports the VM listens to.

If you would like to disable network connectivity, you can use the `autoattachPodInterface` field:

```
kind: VM
spec:
  domain:
    devices:
      autoattachPodInterface: false
```

4.1.6 KubeVirt Network Policy

Maybe needed to separate userspaces or to connect all users machines. https://kubevirt.io/user-guide/virtual_machines/networkpolicy/

4.1.7 KubeVirt Snapshots

KubeVirt has a feature called snapshots. This is currently not documented, but in the near future it may be a good solution for pausing VMs.

4.1.8 KubeVirt ReplicaSets

https://kubevirt.io/user-guide/virtual_machines/replicaset/

[19] https://kubevirt.io/user-guide/virtual_machines/interfaces_and_networks/

4.1.9 KubeVirt Running Windows

https://kubevirt.io/user-guide/virtual_machines/windows_virtio_drivers/

4.1.10 KubeVirt Services

4.1.11 KubeVirt Other Features

There are several other features that we are not going into detail but recommend reading. The most interesting features are the following:

- Virtual Hardware[20], e.g. Resources like CPU, timezone, GPU and memory.
- Liveness and Readiness Probes[21]
- Startup Scripts[22]

4.1.12 KubeVirt CDI

https://kubevirt.io/user-guide/operations/containerized_data_importer/

4.1.13 KubeVirt UIs

There is a comparison about different KubeVirt User Interfaces: KubeVirt user interface options[23].

4.1.14 KubeVirt Additional Plugins

The local persistence volume static provisioner[24] manages the PersistentVolume lifecycle for preallocated disks.

4.2 Base images

4.3 Web access to terminal

4.4 Web access to graphical user interface

https://kubevirt.io/2019/Access-Virtual-Machines-graphic-console-using-noVNC.html ## Integration of terminal and graphical user interface web access to docker base image

[20] https://kubevirt.io/user-guide/virtual_machines/virtual_hardware/

[21] https://kubevirt.io/user-guide/virtual_machines/liveness_and_readiness_probes/

[22] https://kubevirt.io/user-guide/virtual_machines/startup_scripts/

[23] https://kubevirt.io/2019/KubeVirt_UI_options.html

[24] https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner

4.5 Integration of terminal and graphical user interface web access to VM base image

4.6 Integration of base images in Kubernetes

4.7 Routing of base images in Kubernetes

4.8 Multi-user support

4.8.1 Authorization

KubeVirt Authorization[25]

[25] https://kubevirt.io/user-guide/operations/authorization/

List of Figures

5 Bibliography

1. Wissenschaftliche texte schreiben mit markdown und pandoc. Retrieved June 1, 2021 from https://vijual.de/2019/03/11/artikel-mit-markdown-und-pandoc-schreiben/

2. Replacing placeholders with their metadata value. Retrieved June 1, 2021 from https://pandoc.org/lua-filters.html#replacing-placeholders-with-their-metadata-value

3. Ubuntuusers signale. Retrieved June 1, 2021 from https://wiki.ubuntuusers.de/Signale/

4. Ubuntuusers nohup. Retrieved June 1, 2021 from https://wiki.ubuntuusers.de/nohup/

5. What is kubernetes? Retrieved June 2, 2021 from https://www.redhat.com/en/topics/containers/what-is-kubernetes

6. Introduction to kubernetes architecture. Retrieved June 2, 2021 from https://www.redhat.com/en/topics/containers/kubernetes-architecture

7. What is a kubernetes operator? Retrieved June 2, 2021 from https://www.redhat.com/en/topics/containers/what-is-a-kubernetes-operator

8. Understanding kubernetes objects | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

9. What is a kubernetes deployment? Retrieved June 2, 2021 from https://www.redhat.com/en/topics/containers/what-is-kubernetes-deployment

10. Service | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/services-networking/service/

11. Configuration best practices | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/configuration/overview/

12. Ingress | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/services-networking/ingress/

13. Network policies | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/services-networking/network-policies/

14. ConfigMaps | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/configuration/configmap/

15. Secrets | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/configuration/secret/

16. Install tools | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/tasks/tools/

17. Experiment with cdi. Retrieved June 4, 2021 from https://kubevirt.io/labs/kubernetes/lab2.html

18. How to use kubevirt with minikube. Retrieved June 4, 2021 from https://minikube.sigs.k8s.io/docs/tutorials/kubevirt/

19. Use kubevirt. Retrieved June 5, 2021 from https://kubevirt.io/labs/kubernetes/lab1.html

20. Architecture. Retrieved June 5, 2021 from https://kubevirt.io/user-guide/architecture/

21. Lifecycle. Retrieved June 5, 2021 from https://kubevirt.io/user-guide/virtual_machines/lifecycle/

22. Run strategies. Retrieved June 5, 2021 from https://kubevirt.io/user-guide/virtual_machines/run_strategies/

23. Presets. Retrieved June 6, 2021 from https://kubevirt.io/user-guide/virtual_machines/presets/

24. Disks and volumes. Retrieved June 7, 2021 from https://kubevirt.io/user-guide/virtual_machines/disks_and_volumes/

25. Top level api objects. Retrieved June 7, 2021 from https://kubevirt.io/api-reference/v0.6.4/definitions.html

26. Volumes. Retrieved June 10, 2021 from https://kubernetes.io/docs/concepts/storage/volumes/

27. Storage classes. Retrieved June 10, 2021 from https://kubernetes.io/docs/concepts/storage/storage-classes/#local

28. Persistent volumes. Retrieved June 10, 2021 from https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims

29. Interfaces and networks. Retrieved June 10, 2021 from https://kubevirt.io/user-guide/virtual_machines/interfaces_and_networks/

30. Multus. Retrieved June 10, 2021 from https://github.com/k8snetworkplumbingwg/multus-cni