

Lab Orchestrator

Marco Schlicht

Mohamed El Jemai

July 15, 2021

Abstract

This document is the project documentation and is intended, among other things, to describe and explain all aspects, such as tools and required knowledge, that are necessary to successfully complete the project.

The first chapter explains the motivation of the project and the goals we want to achieve. There is also a division of the project into different project phases. In the second chapter the basics needed to understand this project are explained. There are different tools that are described and the key concepts of Kubernetes are explained. After that there are evaluations of which additional tools are required, for which further explanations are included. This contains a more detailed description of Kubernetes objects and KubeVirt, but also information about noVNC and ttyd, two tools which may be used to connect to the containers and VMs.

The project documentation accompanies the project and is continuously supplemented and expanded and should always reflect the current status of the project.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Description	1
1.3	Target Groups	1
1.4	Project Planning	2
1.4.1	Orchestrator	2
1.4.2	Accessible from the Web Base Images	3
1.4.3	Lab Orchestrator Library	3
1.4.4	REST-API	5
1.5	Milestones	5
1.5.1	Prototype	5
1.5.2	Alpha Phase	6
1.5.3	Beta Phase	6
2	Basics	7
2.1	Generating The Documentation	7
2.2	Terminal Tools	7
2.2.1	Make	7
2.2.2	nohup	7
2.3	Kubernetes	8
2.3.1	Control Plane	8
2.3.2	Custom Resource	8
2.3.3	Kubernetes Objects	8
2.3.4	Pods	8
2.3.5	Deployment	8
2.3.6	Services	8
2.3.7	Ingress	9
2.3.8	Ingress Controllers	11
2.3.9	Namespaces	11
2.3.10	Network Policies	11
2.3.11	Config Maps and Secrets	11
2.4	Kubernetes Tools	11
2.4.1	kubectl	11
2.4.2	kind and minikube	11
2.4.3	Helm, Krew, KubeVirt Virtctl and Rancher	12
2.5	Web-Terminal Tools	12
2.6	Web-VNC Tools	12
3	Installation	13
3.1	Prerequisites	13
3.1.1	Kubernetes Development Installation	13
3.1.2	Kubernetes Productive Installation	14
3.1.3	Helm, Krew, KubeVirt and Virtctl Installation	14
3.2	Lab Orchestrator Installation	14

4	Prototype	15
4.1	KubeVirt and Virtual Machines	15
4.1.1	KubeVirt Basics	15
4.1.2	KubeVirt Run Strategies	17
4.1.3	KubeVirt Presets	18
4.1.4	KubeVirt Disks and Volumes	19
4.1.5	KubeVirt Interfaces and Networks	23
4.1.6	KubeVirt Network Policy	24
4.1.7	KubeVirt ReplicaSets	26
4.1.8	KubeVirt Services	26
4.1.9	KubeVirt Other Features	27
4.1.10	KubeVirt Containerized Data Importer (CDI)	27
4.1.11	KubeVirt Additional Plugins	27
4.1.12	cloud-init	27
4.1.13	KubeVirt Running Windows	28
4.2	Building a custom VM	29
4.2.1	Custom Base Image with Cloud-init Setup	29
4.2.2	Customize Image	33
4.2.3	Web Terminal Access	38
4.2.4	Web VNC Access	40
4.3	Base images	49
4.4	Web access to terminal	49
4.5	Web access to graphical user interface	49
4.6	Integration of terminal and graphical user interface web access to docker base image	49
4.7	Integration of terminal and graphical user interface web access to VM base image	49
4.8	Integration of base images in Kubernetes	49
4.9	Routing of base images in Kubernetes	49
4.10	Multi-user support	49
4.11	Separation of labs	49
4.11.1	Authorization	52
5	Bibliography	55

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 and pandoc-crossref filters. For this, you need to download the latest release from here: github.com/owickstrom/pandoc-include-code/releases¹ and here: github.com/lierdakil/pandoc-crossref/releases/tag/v0.4.0.0-alpha4b². Then extract the tar file and install it with the commands `install pandoc-include-code ~/.local/bin` and `install pandoc-crossref ~/.local/bin`. Also make sure `~/.local/bin` is included in `$PATH`. Optionally if your pandoc version is not the correct one, you need to build these tools by yourself.

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]

¹<https://github.com/owickstrom/pandoc-include-code/releases>

²<https://github.com/lierdakil/pandoc-crossref/releases/tag/v0.4.0.0-alpha4b>

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]

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.

Listing 1 Example of a Service

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: my-service
5 spec:
6   selector:
7     app: MyApp
8   ports:
9     - protocol: TCP
10       port: 80
11       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 endpoint 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]

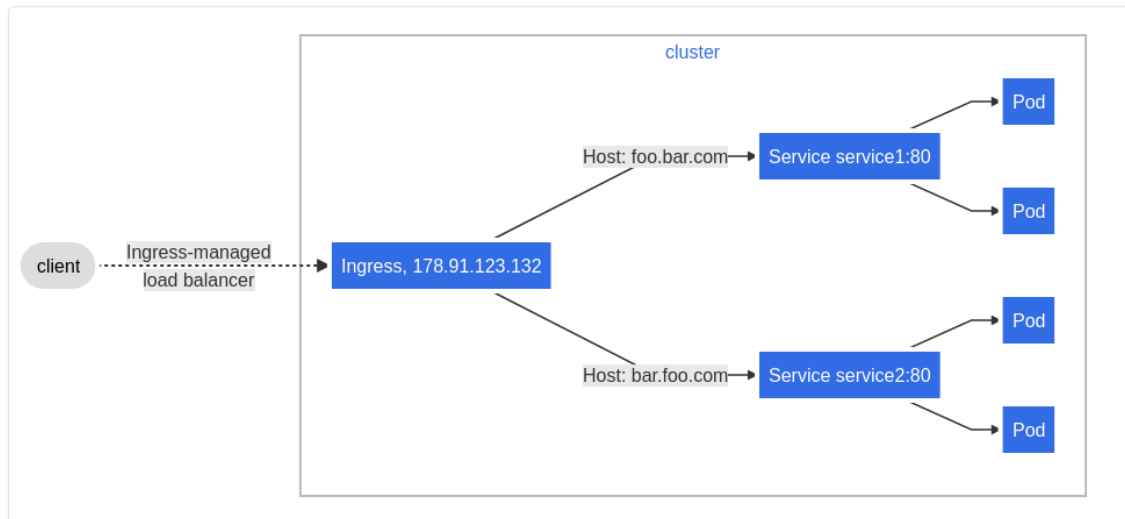


Figure 1: How Ingress interacts with Services and Pods [12]

Listing 2 Example of an Ingress

```

1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: simple-fanout-example
5  spec:
6    rules:
7    - host: foo.bar.com
8      http:
9        paths:
10       - path: /foo
11         pathType: Prefix
12         backend:
13           service:
14             name: service1
15             port:
16               number: 4200
17       - path: /bar
18         pathType: Prefix
19         backend:
20           service:
21             name: service2
22             port:
23               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](https://kubernetes.github.io/ingress-nginx/deploy/)³ and [Traefik Kubernetes Ingress provider](https://doc.traefik.io/traefik/providers/kubernetes-ingress/)⁴.

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 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.

³<https://kubernetes.github.io/ingress-nginx/deploy/>

⁴<https://doc.traefik.io/traefik/providers/kubernetes-ingress/>

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

TLDR:

- [Install Minikube with kvm2](#)⁵
- [Install kubectl](#)⁶
- `minikube start --driver kvm2 --memory=8192 --cpus=4 --disk-size=50g --cni=calico`
- [Install Helm](#)⁷
- [Install Krew](#)⁸
- [Install KubeVirt](#)⁹
- Install Virtctl with Krew
- [Install CDI](#)¹⁰

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](#)¹¹ or some other driver for VMs and follow [this guide](#)¹². Also install kubectl using [this guide](#)¹³.

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]

Minikube creates a VM with 16GB or 20GB disk space. To prevent later errors in the step “Preparing desktop images with cloud-init” which occur due to insufficient space you should create a minikube instance with the parameter `--disk-size=XXGB`, where XX is the amount of space you want to use. You can also increase the memory and CPU amount with `--memory` and `--cpus`. We use `minikube start --vm-driver=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/>

⁹https://kubevirt.io/quickstart_minikube/

¹⁰https://kubevirt.io/user-guide/operations/containerized_data_importer/#install-cdi

¹¹<https://minikube.sigs.k8s.io/docs/drivers/kvm2/>

¹²<https://minikube.sigs.k8s.io/docs/start/>

¹³<https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>

`--memory=8192 --cpus=4 --disk-size=50GB`. In the next steps you should remember to add this parameter by yourself. [19] [20]

Also it's needed to use a network plugin that supports NetworkPolicy. The one we are using is called calico and can be configured during startup with `--cni=calico`. With the parameters from above this results into `minikube start --driver kvm2 --memory=8192 --cpus=4 --disk-size=50g --cni=calico`. [21] [22]

`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: `minikube start --driver kvm2 --memory=8192 --cpus=4 --disk-size=50g --cni=calico`.

Install Helm using [this guide](#)¹⁴.

Install Krew using [this guide](#)¹⁵.

If you are running Minikube, use this installation guide to install KubeVirt and then Virtctl with Krew: [KubeVirt quickstart with Minikube](#)¹⁶. Verify the installation. This adds some commands to `kubectl` for example `kubectl get vms` instead of `kubectl get pods`.

Start `kubevirt` in the minikube cluster: `minikube addons enable kubevirt` or use the in-depth way. After that deploy a test VM using this guide: [Use KubeVirt](#)¹⁷

You also need to [install CDI](#)¹⁸ which is an extension for KubeVirt.

3.2 Lab Orchestrator Installation

¹⁴<https://helm.sh/docs/intro/install/>

¹⁵<https://krew.sigs.k8s.io/docs/user-guide/setup/install/>

¹⁶https://kubevirt.io/quickstart_minikube/

¹⁷<https://kubevirt.io/labs/kubernetes/lab1>

¹⁸https://kubevirt.io/user-guide/operations/containerized_data_importer/#install-cdi

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 if you haven't already done this: [Containerized Data Importer \(CDI\)](#)¹⁹.

The installation of KubeVirt and CDI adds several new CRs, which can be found in the [documentation of kubevirt](#)²⁰.

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. [23] Download the vm config `wget https://raw.githubusercontent.com/kubevirt/kubevirt.github.io/master/labs/manifests/vm.yaml` and apply it: `kubectl apply -f examples/vm.yaml`. Now you should see, that there is a new VM in `kubectl get`

¹⁹<https://kubevirt.io/labs/kubernetes/lab2.html>

²⁰<https://kubevirt.io/user-guide/>

vms called `testvm`. You can start the VM with `kubectrl virt start testvm`. This creates a new VM instance (VMI) that you can see in `kubectrl get vmis`. You can then connect to the serial console using `kubectrl virt console testvm`. Exit the console with `ctrl+]` and stop the VM with `kubectrl virt stop testvm`. Stopping 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` attribute 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. [24]

After starting the VM you can expose its ssh port with this command: `kubectrl virt expose vm testvm --name vmiservice --port 27017 --target-port 22`. Then you can get the cluster-ip from `kubectrl 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`. [24]

VMIs can be paused and unpaused with the commands `kubectrl virt pause vm testvm` or `kubectrl virt pause vmi testvm` and the commands `kubectrl virt unpause vm testvm` or `kubectrl 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. [25]

Listing 3 Example VM (prototype/examples/vm.yaml)

```
1 apiVersion: kubevirt.io/v1
2 kind: VirtualMachine
3 metadata:
4   name: testvm
5 spec:
6   running: false
7   template:
8     metadata:
9       labels:
10        kubevirt.io/size: small
11        kubevirt.io/domain: testvm
12     spec:
13       domain:
14         devices:
15           disks:
16             - name: containerdisk
17               disk:
18                 bus: virtio
19             - name: cloudinitdisk
20               disk:
21                 bus: virtio
22           interfaces:
23             - name: default
24               bridge: {}
25         resources:
26           requests:
27             memory: 64M
28         networks:
29           - name: default
30             pod: {}
31         volumes:
32           - name: containerdisk
33             containerDisk:
34               image: quay.io/kubevirt/cirros-container-disk-demo
35           - name: cloudinitdisk
36             cloudInitNoCloud:
37               userDataBase64: SGkuXG4=
```

The source of the example can be found in [23].

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: [26]

- 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. [27]

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. [27]

Listing 4 Example `VirtualMachineInstancePreset`

```
1 apiVersion: kubevirt.io/v1alpha3
2 kind: VirtualMachineInstancePreset
3 metadata:
4   name: small-qemu
5 spec:
6   selector:
7     matchLabels:
8       kubevirt.io/size: small
9   domain:
10    resources:
11     requests:
12       memory: 64M
```

Listing 5 Example VMI, that matches the correct labels

```
1 apiVersion: kubevirt.io/v1alpha3
2 kind: VirtualMachineInstance
3 version: v1
4 metadata:
5   name: myvmi
6   labels:
7     kubevirt.io/size: small
```

The source of the examples can be found in [27]. The example shows a preset, which applies 64M of memory to every VMI with the label `kubevirt.io/size: small`. [27]

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. [27]

If you change a preset it is only applied to new created VMIs. Old VMIs doesn't change. [27]

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. [28]

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. [28]

Disks have a bus type. A bus type indicates the type of disk device to emulate. Possible types are: `virtio`, `sata`, `scsi`, `ide`. [29]

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. [30]

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. [28]

Possible disk types are: `cloudInitNoCloud`, `cloudInitConfigDrive`, `persistentVolumeClaim`, `persistentVolumeClaim`, `dataVolume`, `ephemeral`, `containerDisk`, `emptyDisk`, `hostDisk`, `configMap`, `secret`, `serviceAccount`, `downwardMetrics`. [28]

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](http://cloudinit.readthedocs.io/en/latest/topics/datasources/nocloud.html)²¹ or the [KubeVirt cloudInitNoCloud documentation](https://kubevirt.io/user-guide/virtual_machines/disks_and_volumes/#cloudinitnocloud)²². [28]

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 `StorageClasses`. [31] A `StorageClass` is the second resource and it is a way for administrators to customize the types of storage the offer. [32] You can read more about `StorageClass` and `PersistentVolume` in the Kubernetes documentation about [Storage Classes](https://kubernetes.io/docs/concepts/storage/storage-classes/#local)²³ and [Persistent Volumes](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims)²⁴.

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 VMs 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. [28]

²¹<http://cloudinit.readthedocs.io/en/latest/topics/datasources/nocloud.html>

²²https://kubevirt.io/user-guide/virtual_machines/disks_and_volumes/#cloudinitnocloud

²³<https://kubernetes.io/docs/concepts/storage/storage-classes/#local>

²⁴<https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims>

Listing 6 Example of VMI with PVC

```
1 metadata:
2   name: testvmi-pvc
3 apiVersion: kubevirt.io/v1alpha3
4 kind: VirtualMachineInstance
5 spec:
6   domain:
7     resources:
8       requests:
9         memory: 64M
10    devices:
11      disks:
12        - name: mypvcdisk
13          lun: {}
14    volumes:
15      - name: mypvcdisk
16        persistentVolumeClaim:
17          claimName: mypvc
```

The source of the example can be found in [28]. 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](#)²⁵. 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. [28]

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. [28]

²⁵https://kubevirt.io/user-guide/virtual_machines/disks_and_volumes/#datavolume-vm-behavior

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 the docker scratch base image and no other content except the image is required. [28]

Listing 7 Dockerfile example with local qcow2 image

```
1 FROM scratch
2 ADD --chown=107:107 fedora25.qcow2 /disk/
```

Listing 8 Dockerfile example with remote qcow2 image

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

Listing 9 Example VMI with Container Disk

```
1 metadata:
2   name: testvmi-containerdisk
3   apiVersion: kubevirt.io/v1alpha3
4   kind: VirtualMachineInstance
5   spec:
6     domain:
7       resources:
8         requests:
9           memory: 8G
10      devices:
11        disks:
12          - name: containerdisk
13            disk: {}
14      volumes:
15        - name: containerdisk
16          containerDisk:
17            image: vmidisks/fedora25:latest
```

The source of the examples can be found in [28]. 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`. [28]

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. [28]

`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. [28]

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`. [28]

4.1.4.8 Remaining Volumes

`hostDisk`, `configMap`, `secrets` and the other volumes are explained in the [KubeVirt Disks and Volumes Documentation](#)²⁶.

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. [33] 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. [34]

Interfaces describe the properties of a virtual interface and are seen inside the `qemu` instance. They are defined in `spec.domain.devices.interfaces`. You can specify its 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](#)²⁷

²⁶https://kubevirt.io/user-guide/virtual_machines/disks_and_volumes/

²⁷https://kubevirt.io/user-guide/virtual_machines/interfaces_and_networks/

Listing 10 Example Network and Interface

```
1 kind: VM
2 spec:
3   domain:
4     devices:
5       interfaces:
6         - name: default
7           masquerade: {}
8         ports:
9           - name: http
10            port: 80
11   networks:
12   - name: default
13     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.

Listing 11 Example of `autoattachPodInterface`

```
1 kind: VM
2 spec:
3   domain:
4     devices:
5       autoattachPodInterface: false
```

4.1.6 KubeVirt Network Policy

By default, all VMIs in a namespace share a network and are accessible from other VMIs. To isolate them, you can create NetworkPolicy objects. NetworkPolicy objects entirely control the network isolation in a namespace. Examples on how to deny all traffic, only allow traffic in the same namespace or only allow HTTP and HTTPS access can be found [here](#)²⁸. [35]

NetworkPolicy objects are included in Kubernetes and are used to separate networks of pods. But with KubeVirt installed, VMIs and pods are treated equally and NetworkPolicy objects can be used for VMIs too. We need to add NetworkPolicy objects to isolate the VMIs of different users, so that the users can't connect to the VMIs of other users. If we create a new namespace for every user, the default settings are sufficient, but creating NetworkPolicy objects gives us more flexibility, e.g. cross namespace connections or isolation of ports. [36]

To use network policies you need to install a network plugin, that supports network policies. So make sure your cluster fulfills this condition. [36]

²⁸https://kubevirt.io/user-guide/virtual_machines/networkpolicy/

Network policies are additive, so if you add two policies the union of them is chosen. If the egress policy or the ingress policy on a pod denies the traffic, the traffic will not be possible even though the network policy would allow it. [36]

How to create and use a NetworkPolicy object is described in the [kubernetes network policy documentation](#)²⁹. You need to define a name of the network policy in the `metadata.name` field and you can specify the namespace this network policy is running in in the `metadata.namespace` field. After that you can specify the policy in the `spec` field. The `spec` field contains a `podSelector`, `policyTypes`, `ingress` and `egress` fields. The `podSelector` field selects the pods the policy will be applied to by defining labels. If the selector is empty all pods in the namespace are selected. Available `policyTypes` are `Ingress` and `Egress`. They can be added to this field to include them. The `Ingress` type is used for incoming requests and the `Egress` type is used for outgoing requests. If you don't specify this field, `Ingress` is activated by default and `Egress` only if an `Egress` rule is added. To add `Ingress` and `Egress` rules there is also the `ingress` and `egress` field in `spec`. Each `ingress` rule allows traffic which matches both the `from` and `ports` sections. The `egress` rules matches both the `to` and `ports` sections. Inside the `from` or `to` sections you can specify for example a `podSelector`, an `ipBlock` or a `namespaceSelector`. The full list of available options can be found in the [NetworkPolicy reference](#)³⁰. [36]

Listing 12 Example of NetworkPolicy

```
1 apiVersion: networking.k8s.io/v1
2 kind: NetworkPolicy
3 metadata:
4   name: multi-port-egress
5   namespace: default
6 spec:
7   podSelector:
8     matchLabels:
9       role: db
10  policyTypes:
11    - Egress
12  egress:
13    - to:
14      - ipBlock:
15        cidr: 10.0.0.0/24
16    ports:
17      - protocol: TCP
18        port: 32000
19        endPort: 32768
```

The example shows a network policy with an `Egress` rule that allows all pods and VMIs with the label `role: db` to connect to all pods and VMIs within the IP range 10.0.0.0/24

²⁹<https://kubernetes.io/docs/concepts/services-networking/network-policies/#networkpolicy-resource>

³⁰<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.21/#networkpolicy-v1-networking-k8s-io>

over TCP with the ports between 32000 and 32778. The source of the example can be found here: [36].

4.1.7 KubeVirt ReplicaSets

VirtualMachineInstanceReplicaSets are similar like Kubernetes ReplicaSets. They are used to deploy multiple instances of the same VMI to guarantee uptime. There is no state in the instances of a ReplicaSet so you need to use read-only or internal writable tmpfs disks. Since our labs need a state we probably won't need ReplicaSets. [37]

4.1.8 KubeVirt Services

VMIs can be exposed with services. Services were explained earlier. This is needed to connect to a VMI for example over SSH. Services use labels to identify the VMI, so you need to add labels to the VMI you want to connect to. To create a new Service you can either create a File and load it with `kubectl -f file.yaml` or you can use the `virtctl` tool (remember it may also be used with `kubectl virt`): `virtctl expose virtualmachineinstance vmi-ephemeral --name lbsvc --type LoadBalancer --port 27017 --target-port 3389`. This command uses the type `LoadBalancer`, other types are `NodePort` and `ClusterIP`. [38]

Listing 13 Example VMI with Labels

```
1 apiVersion: kubevirt.io/v1alpha3
2 kind: VirtualMachineInstance
3 metadata:
4   name: vmi-ephemeral
5   labels:
6     special: key
7 spec:
8   ...
```

Listing 14 Example VMI exposed as Service

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: vmiservice
5 spec:
6   ports:
7     - port: 27017
8       protocol: TCP
9       targetPort: 22
10  selector:
11    special: key
12  type: ClusterIP
```

The examples are from [KubeVirt's Service Objects Documentation](#)³¹ and they show a VMI with a Label and a Service that exposes SSH on this VMI.

4.1.9 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](#)³², e.g. Resources like CPU, timezone, GPU and memory.
- [Liveness and Readiness Probes](#)³³
- [Startup Scripts](#)³⁴
- [KubeVirt Snapshots](#), may be used to pause VMs.
- [KubeVirt user interface options](#)³⁵, there are different KubeVirt User Interfaces.

4.1.10 KubeVirt Containerized Data Importer (CDI)

The CDI is a separate project that can be added to KubeVirt. To use this you need to [install it](#)³⁶ if you haven't already done this.

TODO

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

4.1.11 KubeVirt Additional Plugins

The [local persistence volume static provisioner](#)³⁷ manages the PersistentVolume lifecycle for preallocated disks.

4.1.12 cloud-init

Cloud-init is a standard for cloud instance initialization. Cloud-init will read any provided metadata and initialize the system accordingly. This includes setting up network and storage devices and configuring SSH. For example it is possible to provide an ssh key as metadata. [39]

Cloud-init supports Windows and all major Linux distributions like: Arch, Alpine, Debian, Fedora, RHEL and SLES. [40]

Cloud-init needs to be integrated into the boot of the VM. For example this can be done with systemd. [41] In addition cloud-init needs a datasource. There are many supported datasources for different cloud providers, but the most important for this project will be the NoCloud datasource, because this can be used in KubeVirt as we have already seen above. [42] NoCloud allows to provide meta-data to the VM via files on a mounted filesystem. [43]

³¹https://kubevirt.io/user-guide/virtual_machines/service_objects/

³²https://kubevirt.io/user-guide/virtual_machines/virtual_hardware/

³³https://kubevirt.io/user-guide/virtual_machines/liveness_and_readiness_probes/

³⁴https://kubevirt.io/user-guide/virtual_machines/startup_scripts/

³⁵https://kubevirt.io/2019/KubeVirt_UI_options.html

³⁶https://kubevirt.io/user-guide/operations/containerized_data_importer/#install-cdi

³⁷<https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner>

It is modularized and there are many modules available to support many different system configurations and different tools. The most important will be the SSH module and maybe Apt Configure, Disk Setup and Mount. All modules can be found in the [cloud-init Modules Documentation](#)³⁸ and examples can be found in the [cloud-init config examples documentaion](#)³⁹.

Listing 15 Example VM with cloud-init NoCloud

```
1 apiVersion: kubevirt.io/v1alpha1
2 kind: VirtualMachine
3 metadata:
4   name: myvm
5 spec:
6   terminationGracePeriodSeconds: 5
7   domain:
8     resources:
9       requests:
10        memory: 64M
11   devices:
12     disks:
13     - name: registrydisk
14       volumeName: registryvolume
15       disk:
16         bus: virtio
17     - name: cloudinitdisk
18       volumeName: cloudinitvolume
19       disk:
20         bus: virtio
21   volumes:
22     - name: registryvolume
23       registryDisk:
24         image: kubevirt/cirros-registry-disk-demo:devel
25     - name: cloudinitvolume
26       cloudInitNoCloud:
27         userData: |
28           ssh-authorized-keys:
29             - ssh-rsa AAAAB3NzaK8L93bWxnyp test@test.com
```

This is an example that shows how cloud-init NoCloud could be used in KubeVirt to add an ssh key. The created VM contains two disks, one for the image that should be used and another disk that is used by cloud-init. The source can be found here: [44].

4.1.13 KubeVirt Running Windows

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

³⁸<https://cloudinit.readthedocs.io/en/latest/topics/modules.html>

³⁹<https://cloudinit.readthedocs.io/en/latest/topics/examples.html>

4.2 Building a custom VM

In the first step we try to get a cloud image from a Linux distribution running inside of KubeVirt and then excess it. The second step tries to build a custom image on top of the cloud image. This is needed to install software for labs. In the third and fourth step we will install ttyd and noVNC or alternatives of that and access them with a web browser.

4.2.1 Custom Base Image with Cloud-init Setup

In KubeVirt you need cloud-images in the format of qcow2 or raw. You can obtain your preferred distro from [the openstack image guide](#)⁴⁰. The list in the openstack image guide contains images that comes with cloud-init preinstalled. This is useful, because most of them doesn't have a default login and we need to add the login data with cloud-init. In this example we have used the [Ubuntu Hirsute cloud-image](#)⁴¹, saved it in the folder images and we have a docker hub account.

Listing 16 Example Dockerfile for Custom Image

```
1 FROM scratch
2 ADD --chown=107:107 images/ubuntu-21.04-server-cloudimg-amd64.img /disk/
```

After downloading the image create a dockerfile that adds the image into /disk/. The [listing 16](#) shows how to do this with the ubuntu image. Save this file in a file called dockerfile.

After that, build the dockerfile with `docker build -t dockerhubusername/reponame:ubuntu2104 -f dockerfile ..`. Then login the docker client to docker hub with `docker login` and providing your login credentials. Then upload the build image to docker hub with `docker push dockerhubusername/reponame:ubuntu2104`. Now you have a docker image in docker hub that contains the ubuntu cloud-image in /disk/.

In the next step we will use this image with a container disk to run the ubuntu cloud-image. First create a file called `ubuntu_container_disk.yaml` and add a container disk setup.

⁴⁰<https://docs.openstack.org/image-guide/obtain-images.html>

⁴¹<https://cloud-images.ubuntu.com/releases/hirsute/release/>

Listing 17 Example Container Disk for Custom Image

```
1 metadata:
2   name: testvmi-containerdisk
3   labels:
4     special: key
5 apiVersion: kubevirt.io/v1alpha3
6 kind: VirtualMachineInstance
7 spec:
8   domain:
9     resources:
10      requests:
11        memory: 500M
12   devices:
13     disks:
14       - name: containerdisk
15         disk: {}
16       - name: cloudinitdisk
17         disk:
18           bus: virtio
19   volumes:
20     - name: containerdisk
21       containerDisk:
22         image: dockerhubusername/reponame:ubuntu2104
23     - name: cloudinitdisk
24       cloudInitNoCloud:
25         userData: |-
26           #cloud-config
27           users:
28             - name: root
29               ssh-authorized-keys:
30                 - ssh-rsa AAAABSSHKEY
31           ssh_pwauth: True
32           password: toor
33           chpasswd:
34             expire: False
35             list: |-
36               root:toor
```

The **listing 17** is an example of a container disk that uses the docker image we have created previously. Also there is a cloud-init NoCloud disk attached that adds login credentials that can be used to login to the VM via console and via ssh. In this example the username is root and the password toor. If you want to use ssh replace ssh-rsa AAAABSSHKEY with your ssh-key, else remove this part of the configuration. To disable password login set ssh_pwauth: False.

Then run this VMI with the command `kubectl apply -f ubuntu_container_-`

disk.yaml and wait until the VMI is started with `kubectl wait --for=condition=Ready vmis/testvmi-containerdisk` or `kubectl wait --for=condition=Ready -f ubuntu_container_disk.yaml`.

Now the VMI is running and you can access it over console: `kubectl virt console testvmi-containerdisk`. To access the ssh, you need to create a service and connect over minikube ssh. Create the service with `kubectl virt expose vmi testvmi-containerdisk --name vmiservice --port 27017 --target-port 22`. You can get the ip with `kubectl get svc`. To connect to ssh, you need to execute `minikube ssh`, then insert your ssh private key with:

Listing 18 Insert SSH Key in Minikube

```
1 cat <<<EOF > ~/.ssh/id_rsa
2 YOURSSHKEY
3 EOF
```

After that change the permissions of the file to 600 with `chmod 600 ~/.ssh/id_rsa` and connect to the ip from the service on the given port with `ssh -p PORT root@IP` and you should be connected to the VMI.

```
→ prototype git:(master) X kubectl virt console testvmi-containerdisk
Successfully connected to testvmi-containerdisk console. The escape sequence is ^]

testvmi-containerdisk login: root
Password:
Welcome to Ubuntu 21.04 (GNU/Linux 5.11.0-22-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Jul  3 20:28:23 UTC 2021

System load:  0.44           Processes:           113
Usage of /:   65.1% of 1.96GB Users logged in:          0
Memory usage: 41%           IPv4 address for enp1s0: 172.17.0.17
Swap usage:   0%

0 updates can be applied immediately.

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@testvmi-containerdisk:~# whoami
root
root@testvmi-containerdisk:~# □
```

Figure 2: Example of Console Login


```

$ ssh -p 27017 root@10.98.58.45
Welcome to Ubuntu 21.04 (GNU/Linux 5.11.0-22-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Jul  3 20:34:33 UTC 2021

System load:  0.0               Processes:            111
Usage of /:   65.7% of 1.96GB   Users logged in:     1
Memory usage: 39%              IPv4 address for enp1s0: 172.17.0.17
Swap usage:   0%

0 updates can be applied immediately.

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Sat Jul  3 20:32:27 2021 from 172.17.0.1
root@testvmi-containerdisk:~# whoami
root
root@testvmi-containerdisk:~# █

```

Figure 4: Example of SSH Setup Login

4.2.2 Customize Image

4.2.2.1 Docker Notice

Because we use container disks in this part to run our VMs in the Kubernetes cluster and container disks are using docker you need to know about the build context of docker to prevent errors. The build context of docker is a folder from where all recursive contents of files and directories will be sent to the docker daemon. This is used by docker to isolate the context where the image is build and to prevent errors in later use of the image. If one of your subdirectories contains all your VM images, all of them will be sent to the docker daemon and this may slow down the build process. Also if your build context is too big docker hub will not accept your images even if they are much smaller (e.g. your docker image contains one VM and is 5GB big, but the context contains 6 VMs and is 30GB big). To speedup your builds and to prevent docker hub errors you should always separate your dockerfiles and VM images into different folders or exclude all VM images that you are not adding to the docker image in the `.dockerignore`. [45] [46]

4.2.2.2 Non Cloud Images

If you use other images than cloud images, you need to install cloud-init manually. To use such an image, open it with gnome boxes, then install your software and shutdown. After that your qcow2 image is saved in `~/.var/app/org.gnome.Boxes/data/gnome-boxes/images/` if you installed gnome boxes with snap or in `~/.local/share/gnome-boxes` if you installed it with apt. [47]

4.2.2.3 Startup, password and internet setup

First install virt-customize, for example by following this guide: [Customize qcow2-raw-image templates with virt-customize](https://computingforgeeks.com/customize-qcow2-raw-image-templates-with-virt-customize/)⁴². [48]

virt-customize allows you to customize your cloud images. The command `sudo virt-customize -a your_image.img --root-password password:StrongRootPassword` for example changes the password of the root user. This is needed, because most of the cloud images doesn't have a default root password. [48]

Now you can start the image with gnome boxes by adding a new VM with this as image. This creates a new qcow2 image in `~/.var/app/org.gnome.Boxes/data/gnome-boxes/images/` if you installed gnome boxes with snap or in `~/.local/share/gnome-boxes` if you installed it with apt. [47] You can check the filetype with `file filename`. After every change on the original image you need to delete the VM and create a new or you directly change the new image. After starting the image login with the previously set root password. You need to connect to the internet with the command `dhclient`. This gets an ip-address.

4.2.2.4 Resize

To resize the image first stop it if you are running it. Check the image size with the command `qemu-img info your_image.img`. Then resize the image size with `sudo qemu-img resize your_image.img +4G` and check the new size again with `qemu-img info your_image.img`. [49] [50]

Now start it and check how big the partitions are with `df -h`. Then execute `fdisk /dev/sda`. Then use `p` command to print all partitions and search for Linux filesystem. In the Ubuntu cloud image this is `/dev/sda1`. Next you need to delete this partition with the `d` command and then input the partition number (1). Now add a new partition with `n` and the same number (1) and then take the default values of the next two questions. After that don't remove the signature (N). Last execute `w` to write the changes. Now the partition is resized and you need to resize the filesystem. This is done with `resize2fs /dev/sda1`. [49] [51] [52]

Now you have a resized image of your cloud image in the gnome boxes folder. Make a backup of it by copying the file.

In the following images, the host terminal has white background and the VM terminal has a black background.

⁴²<https://computingforgeeks.com/customize-qcow2-raw-image-templates-with-virt-customize/>

```

→ images git:(master) X qemu-img info custom-ubuntu-21.04-server-cloudimg-amd64.img
image: custom-ubuntu-21.04-server-cloudimg-amd64.img
file format: qcow2
virtual size: 2.2 GiB (2361393152 bytes)
disk size: 551 MiB
cluster_size: 65536
Format specific information:
    compat: 0.10
    refcount bits: 16

```

Figure 5: Image size before change

```

→ images git:(master) X sudo qemu-img resize custom-ubuntu-21.04-server-cloudimg-amd64.img +
4G
Image resized.
→ images git:(master) X qemu-img info custom-ubuntu-21.04-server-cloudimg-amd64.img
image: custom-ubuntu-21.04-server-cloudimg-amd64.img
file format: qcow2
virtual size: 6.2 GiB (6656360448 bytes)
disk size: 551 MiB
cluster_size: 65536
Format specific information:
    compat: 0.10
    refcount bits: 16

```

Figure 6: Image size after change

```

root@ubuntu:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           198M  964K  197M   1% /run
/dev/sda1       2.0G  1.3G  677M  67% /
tmpfs           989M    0  989M   0% /dev/shm
tmpfs           5.0M    0   5.0M   0% /run/lock
tmpfs           4.0M    0   4.0M   0% /sys/fs/cgroup
/dev/sda15      105M  5.2M  100M   5% /boot/efi
tmpfs           198M  4.0K  198M   1% /run/user/0
root@ubuntu:~# fdisk /dev/sda

Welcome to fdisk (util-linux 2.36.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

GPT PMBR size mismatch (4612095 != 13000703) will be corrected by write.
The backup GPT table is not on the end of the device. This problem will be corrected by write.

Command (m for help):

```

Figure 7: Disk size before change

```

Command (m for help): p

Disk /dev/sda: 6.2 GiB, 6656360448 bytes, 13000704 sectors
Disk model: QEMU HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 65EEDECF-3EFC-47F0-A985-253F82CD250D

Device      Start      End Sectors  Size Type
/dev/sda1   227328   4612062 4384735   2.1G Linux filesystem
/dev/sda14    2048    10239    8192     4M BIOS boot
/dev/sda15   10240   227327  217088   106M EFI System

Partition table entries are not in disk order.

Command (m for help): _

```

Figure 8: fdisk partition size before change

```

Command (m for help): d
Partition number (1,14,15, default 15): 1

Partition 1 has been deleted.

Command (m for help): n
Partition number (1-13,16-128, default 1):
First sector (34-13000670, default 227328):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (227328-13000670, default 13000670):

Created a new partition 1 of type 'Linux filesystem' and of size 6.1 GiB.
Partition #1 contains a ext4 signature.

Do you want to remove the signature? [Y]es/[N]o: N

Command (m for help):

```

Figure 9: fdisk delete partition and create new

```

Command (m for help): p

Disk /dev/sda: 6.2 GiB, 6656360448 bytes, 13000704 sectors
Disk model: QEMU HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 65EEDECF-3EFC-47F0-A985-253F82CD250D

Device        Start      End   Sectors   Size Type
/dev/sda1    227328 13000670 12773343   6.1G Linux filesystem
/dev/sda14     2048    10239    8192     4M BIOS boot
/dev/sda15    10240   227327   217088   106M EFI System

Partition table entries are not in disk order.

Command (m for help): w
The partition table has been altered.
Syncing disks.

root@ubuntu:~#

```

Figure 10: fdisk partition size after change and write changes

```

root@ubuntu:~# resize2fs /dev/sda1
resize2fs 1.45.7 (28-Jan-2021)
Filesystem at /dev/sda1 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/sda1 is now 1596667 (4k) blocks long.

root@ubuntu:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           198M  964K  197M   1% /run
/dev/sda1       5.9G  1.3G  4.6G  23% /
tmpfs           989M    0  989M   0% /dev/shm
tmpfs           5.0M    0   5.0M   0% /run/lock
tmpfs           4.0M    0   4.0M   0% /sys/fs/cgroup
/dev/sda15      105M  5.2M  100M   5% /boot/efi
tmpfs           198M  4.0K  198M   1% /run/user/0
root@ubuntu:~# _

```

Figure 11: Resize filesystem and disk size after change

4.2.2.5 Installing software

There are two ways of installing software. The first uses `virt-customize` and the second uses `gnome boxes`.

To install software with `virt-customize` you can append the command `--install PackageName`, e.g. `virt-customize -a your_image.img --install firefox`. This uses the default package manager. [48]

If it's not possible to install software with the package manager, you can use the second way, `gnome boxes`. Start the resized image and connect to internet (`dhclient`). Then update the software with `apt update && apt upgrade` and install your software and shutdown. Remember that you are editing the image in the `gnome boxes` folder and not the original image.

4.2.3 Web Terminal Access

In this step we will try to get access to the terminal over a website. There are two ways of archiving this goal, the first is to install `ttyd` or a similar software inside the VM and the second one is to run `ttyd` outside of the VM and share `kubevirt virt console`.

4.2.3.1 ttyd inside VM

To archive this, we need one of the VMs from before with enough space to install software.

Start the VM in `boxes` and install `ttyd` with [this guide](#)⁴³. Then after the installation `ttyd` needs to be started automatically within `systemstart`. This can be done for example by adding a cronjob with `@reboot`. So execute `crontab -e` and add `@reboot ttyd bash` there. This will start `ttyd bash` when the system starts. It will automatically log in the user from which you run the cronjob. If you run `crontab -e` with the root user, the `ttyd` shell will have root permissions in the VM. If you run `crontab -e` with a custom user, the `ttyd` shell will have the permissions of this user. You can change `bash` with all other commands you want to be executed inside the webshell. `ttyd command` will execute the command and share it over http. In this scenario we like to have access to a bash console in the web browser, but you can also start a `zsh` or other shell or even `nodejs`, `python` interpreter or other software. [53] [54]

After installing `ttyd` and starting it automatically on system start with `cron`, we need to run this image in `Kubernetes` and expose the `ttyd` service. For this first stop the VM and copy the generated `qcow2` image from `~/.var/app/org.gnome.Boxes/data/gnome-boxes/images/` or `~/.local/share/gnome-boxes` to your folder. Then build a new docker image that adds this `qcow2` file, an example is given in [listing 16](#). Push the image to your docker registry. Then start a new `Kubernetes` VM with an container disk attached that references this docker image like shown in [listing 17](#). If you are using [listing 17](#), the root password that you may have set before will be overwritten.

Now there should be our custom VM running in `Kubernetes`. To access the `ttyd` service we need to expose the port. The default `ttyd` port is 7681, so execute the command

⁴³<https://github.com/tsl0922/ttyd#installation>

`kubectl virt expose vmi your_vmi_name --type=NodePort --name ttydservice --port 27017 --target-port 7681`. This creates a Kubernetes NodePort service, that makes it possible for us to access the port 7681 of the VM over the ip of our node with the port 27017. `minikube service ttydservice` will let us connect to the service and open it in our default browser. [55]

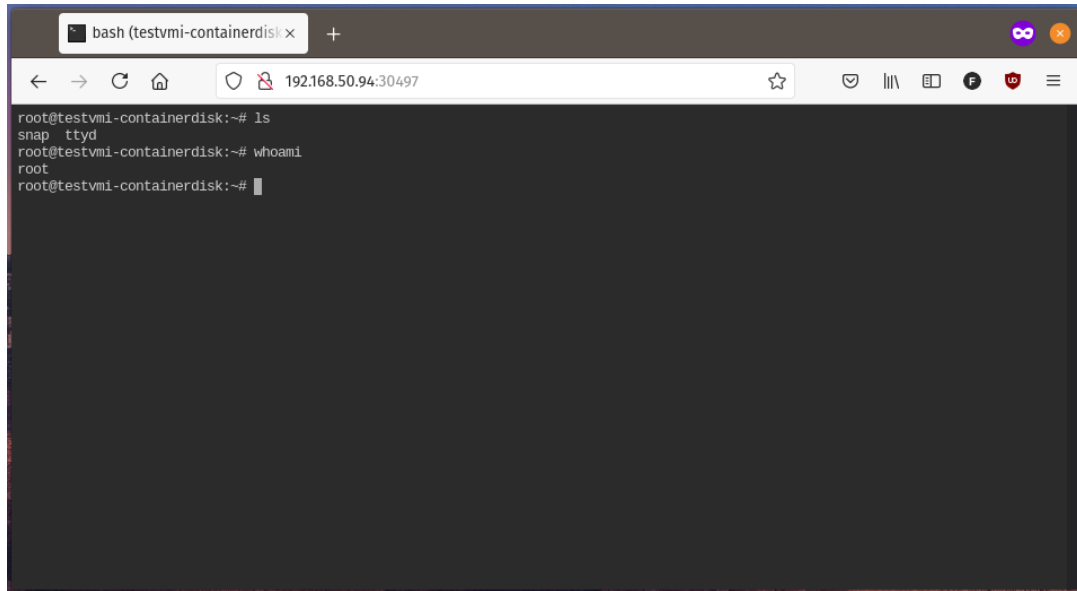
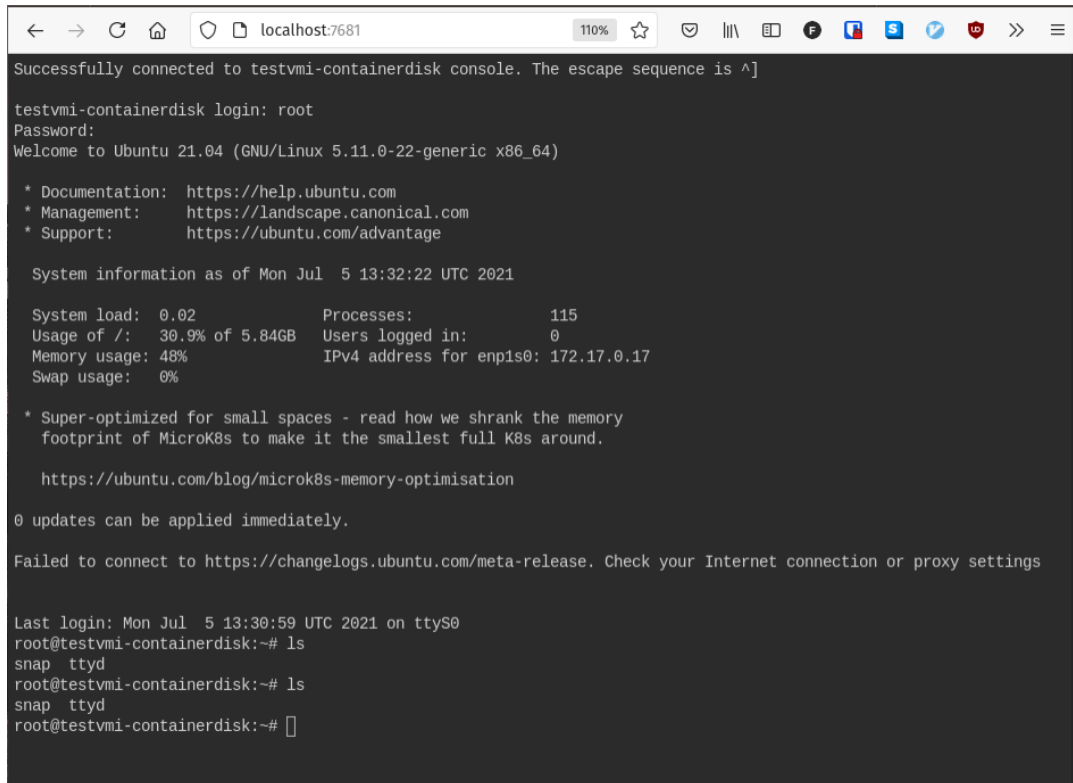


Figure 12: ttyd running inside the container

This allows us to create custom images and access them with any software, for example bash, zsh, python, nodejs. This solution is very customizable, but it's not possible to share the system console which shows e.g. the boot process.

4.2.3.2 ttyd outside VM

The second way is to run ttyd outside of the container and run `ttyd kubectl virt console your_vmi_name`. This allows to share the console of the VM with ttyd and this includes the boot process of the VM. VM developers can't customize which command is executed here, because this is run on the host machine. Also you aren't logged in automatically. It may be possible to run this in a second container that maintains the VMs but that for another time.



```
Successfully connected to testvmi-containerdisk console. The escape sequence is ^]
testvmi-containerdisk login: root
Password:
Welcome to Ubuntu 21.04 (GNU/Linux 5.11.0-22-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

System information as of Mon Jul  5 13:32:22 UTC 2021

System load:  0.02               Processes:            115
Usage of /:   30.9% of 5.84GB    Users logged in:     0
Memory usage: 48%               IPv4 address for enp1s0: 172.17.0.17
Swap usage:   0%

* Super-optimized for small spaces - read how we shrank the memory
  footprint of MicroK8s to make it the smallest full K8s around.

https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be applied immediately.

Failed to connect to https://changelogs.ubuntu.com/meta-release. Check your Internet connection or proxy settings

Last login: Mon Jul  5 13:30:59 UTC 2021 on ttyS0
root@testvmi-containerdisk:~# ls
snap  ttyd
root@testvmi-containerdisk:~# ls
snap  ttyd
root@testvmi-containerdisk:~#
```

Figure 13: ttyd running outside the container

4.2.4 Web VNC Access

4.2.4.1 Tools

noVNC is a VNC viewer that runs in the browser. Usually VNC uses TCP sockets, but noVNC needs websockets. There is a tool called websockify which converts TCP sockets to websockets. This can be used to connect noVNC to any VNC server. [56]

KubeVirt has a command that creates a VNC server and opens a VNC client for any VMI: `kubectl virt vnc your_vmi_name`. If you want to connect your own VNC client there is the command `kubectl virt vnc your_vmi_name --proxy-only`. This creates a VNC server for the VMI with an TCP proxy. [57]

There is also a tool called [virtVNC⁴⁴](https://github.com/wavezhang/virtVNC). This tool can be used to access the VMIs graphical console using noVNC and combines the above two tools.

4.2.4.2 Preparing images

VNC enables us to use the desktop of the system. So we need to have images with a desktop environment installed. There are two ways of archiving this: 1. install desktop environment in cloud-images or 2. use images that already have a desktop environment installed.

4.2.4.3 Preparing desktop images with cloud-init

⁴⁴<https://github.com/wavezhang/virtVNC>

We will use an ubuntu 20.04 desktop image in this step with gnome installed. You can download the image from [here](#)⁴⁵ or use other images. Open the image in gnome boxes and install it. After the installation and configuration of the system start it and install cloud-init. An example installation tutorial for ubuntu can be found [here](#)⁴⁶. After the installation of ubuntu and cloud-init stop the VM and copy the qcow2 image to your working folder, add it to a docker image and push it to docker hub like described earlier.

While starting the image in minikube an error occurred in my system: “no space left on device”. This is because minikube runs in a VM that has a fixed size of 16GB by default, only 2GB left free and the image has 10GB. You can check how much space is available inside minikube by executing `minikube ssh` and then `df -h`. To fix this issue, you need to stop and delete your minikube instance and then create a new with the parameter `--disk-size=XXGB` where XX is the size you want. [19] [58] Commands:

1. `minikube stop`
2. `minikube delete`
3. `minikube start --vm-driver=kvm2 --disk-size=50GB`
4. Then reinstall everything: KubeVirt, virtVNC, etc.

Now start this image in Kubernetes. This may take a while because you need to download the image which might be very big. You can check if VNC is working by executing `kubectl virt vnc your_vmi_name`. If you can see the desktop it's working. [59]

After connecting to the VNC of the Ubuntu machine you are able to login. The login and loading of Gnome takes some time. At the moment we don't know why, but maybe it would be faster if we attach a GPU to the cluster. Fasten up the VMs will be done in a different step. TODO

4.2.4.4 Preparing cloud images with desktop environment

In this step we use the previously build ubuntu cloud image, that has be resized. To install gnome-desktop you need at least 2.2GB free space on the VM. To install gnome-desktop in ubuntu cloud image start it in boxes and then execute `apt install ubuntu-gnome-desktop`. After that shutdown the VM, copy the qcow2 image to your working folder, add it to a docker image and push it to docker hub like described earlier. Now start this image in Kubernetes with enough memory (e.g. 2GB). You can check if VNC is working by executing `kubectl virt vnc your_vmi_name`. If you can see the desktop and login it's working. [60]

To install other desktop environments you can follow the steps in this guide: [Ubuntu cloud desktop adding gui to your cloud server instance](#)⁴⁷.

4.2.4.5 Connect noVNC to kubectl vnc

Start the VNC proxy with `kubectl virt vnc your_vmi_name --proxy-only`. In the output of the command the port where the VNC server is reachable is shown. Now you

⁴⁵<https://ubuntu.com/download/desktop/thank-you?version=20.04.2.0&architecture=amd64>

⁴⁶<https://zoomadmin.com/HowToInstall/UbuntuPackage/cloud-init>

⁴⁷<https://www.suhendro.com/2019/04/ubuntu-cloud-desktop-adding-gui-to-your-cloud-server-instance/>

can access the VNC with your VNC viewer. If you connect with an RDP program, or if you disconnect the VNC viewer, the kubectl proxy will break. This may be a problem, because it seems to be easy to break this solution. Also if you restart, a new random port will be used if you don't specify a fixed port with `--port`.

Install noVNC from github.com/novnc/novnc. You can run noVNC with `sudo novnc --listen 6081 --vnc localhost:40753`, where 6081 is the port where noVNC will be reachable and 40753 is the port where the VNC server is reachable. When starting this and the VNC server doesn't support websockets, websockify will automatically be started by noVNC. Now you can open this in your browser and connect to your VM.

```
→ Documentation git:(master) X kubectl virt vnc ubuntu-desktop --proxy-only --port 40753
{"port":40753}
{"component":"","level":"info","msg":"connection timeout: 1m0s","pos":"vnc.go:144","timestamp":"2021-07-09T09:57:31.963647Z"}

→ Documentation git:(master) X sudo novnc --listen 6081 --vnc localhost:40753
[sudo] password for marco:
Warning: could not find self.pem
Using installed websockify at /snap/novnc/6/bin/websockify
Starting webserver and WebSockets proxy on port 6081
WebSocket server settings:
- Listen on :6081
- Web server. Web root: /snap/novnc/6
- No SSL/TLS support (no cert file)
- proxying from :6081 to localhost:40753

Navigate to this URL:

http://pop-os:6081/vnc.html?host=pop-os&port=6081

Press Ctrl-C to exit
```

Figure 14: noVNC and vnc proxy

kubectl is only a tool that wraps around the Kubernetes API. kubevirt vnc should execute some API commands to get a VNC connection. Maybe we can use this to directly get the VNC connection without kubevirt. This can then be used in our library.

noVNC can also be run as a service and listen on multiple ports and connecting to multiple VNC servers. It's also possible to [embed noVNC into our own application](#)⁴⁸.

4.2.4.6 virtVNC

virtVNC can be installed by just applying their yaml file: `kubectl apply -f https://github.com/wavezhang/virtVNC/raw/master/k8s/virtvnc.yaml`. After this is deployed to your minikube cluster, you can see there is a new service installed with `kubectl get svc -n kubevirt virtvnc`. Execute `minikube service virtvnc -n kubevirt` to open virtVNC in your browser. [57]

virtVNC enables you to have a minimalistic dashboard, where you can see all running VMs and access their desktop over noVNC. You can filter the namespace by appending `?namespace=your_namespace` to the url. If VMs are currently not running but starting, for example if the image is downloading, they are shown with message Scheduling. An example of this is shown in Figure 16. [57]

⁴⁸<https://github.com/novnc/noVNC/blob/master/docs/EMBEDDING.md>

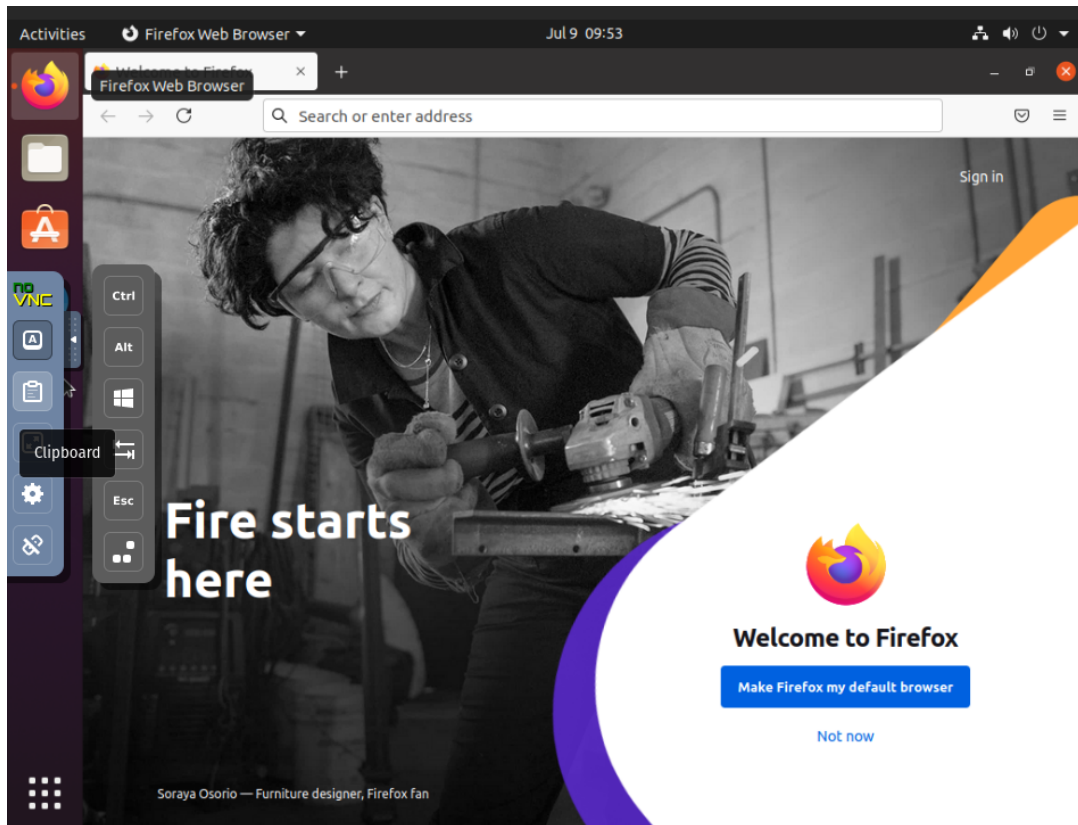


Figure 15: noVNC in browser

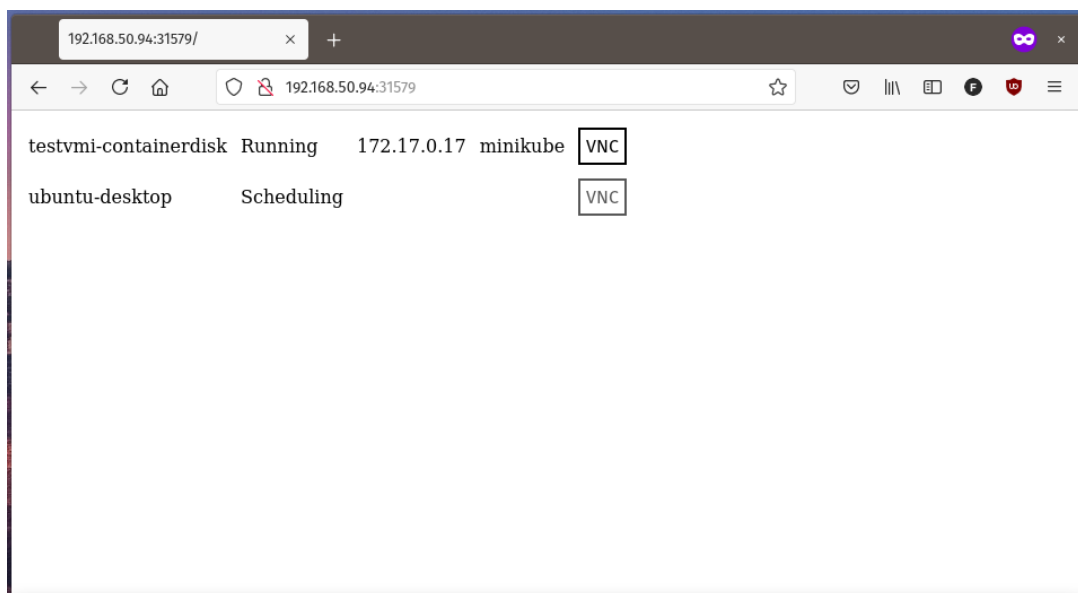


Figure 16: virtVNC list of VMs

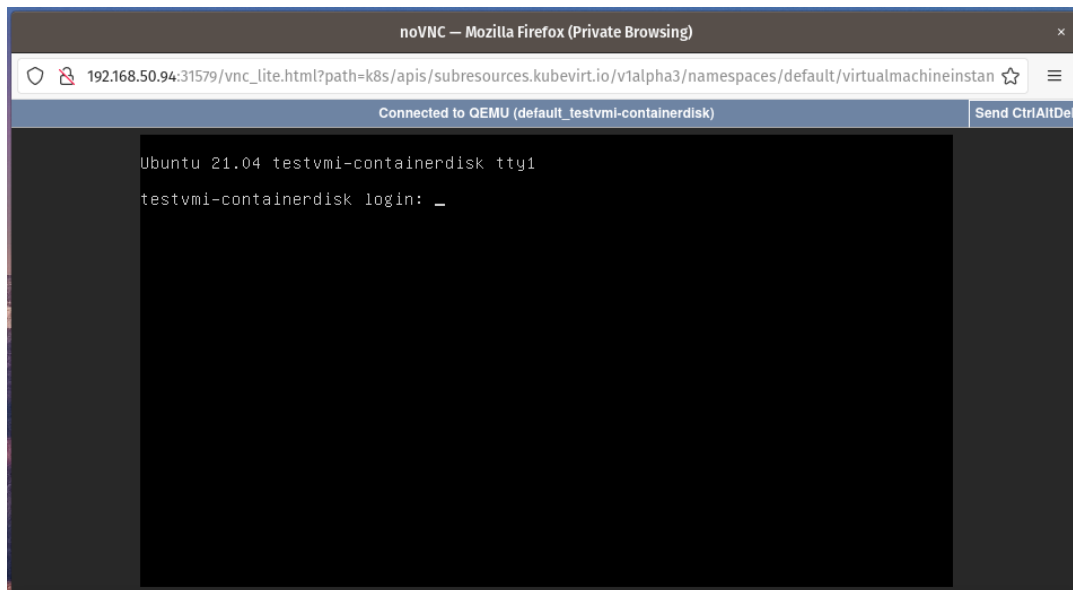


Figure 17: virtVNC showing a console

If you don't have a desktop environment installed, virtVNC gives you access to the console. This is seen in Figure 17.

In the Figure 18 you can see the noVNC connection to the Ubuntu VM.

virtVNC doesn't have a permission system out of the box and it's possible to access any VNC console from any VM. In the lab orchestrator we need to be able to restrict users from accessing VMs that aren't theirs. Maybe we can extend virtVNC with a permission system or user authentication to restrict accessing every VM or build our own solution on top of the same principles like virtVNC. Maybe it will be enough to change the RBAC rules. Nevertheless, it is worth taking a look at how virtVNC works.

4.2.4.7 Directly accessing the API

To understand what's happening in the virtVNC pod we will take a look at the script that is running there. You can see it in the [kubernetes docs about noVNC](https://kubernetes.io/docs/concepts/containers/vnc/)⁴⁹.

First of all there is a hint that KubeVirt provides WebSocket api access for VNC under: `APISERVER:/apis/subresources.kubevirt.io/v1alpha3/namespaces/NAMESPACE/virtualmachineinstances/VM/vnc`. [61]

Kubernetes has an API and if you run Kubernetes with Minikube you can access the API over the ip of minikube: `minikube ip`. This API needs some authentication. To bypass the authentication you can run `kubectl proxy`. This gives you an ip and port where you can access the api without authentication.

What's also needed is a python program called SimpleHTTPServer. You can start this server with `python -m SimpleHTTPServer`. Now the server will run on `localhost:8000` and serve all files in the current directory. [62]

⁴⁹<https://kubernetes.io/2019/Access-Virtual-Machines-graphic-console-using-noVNC.html>

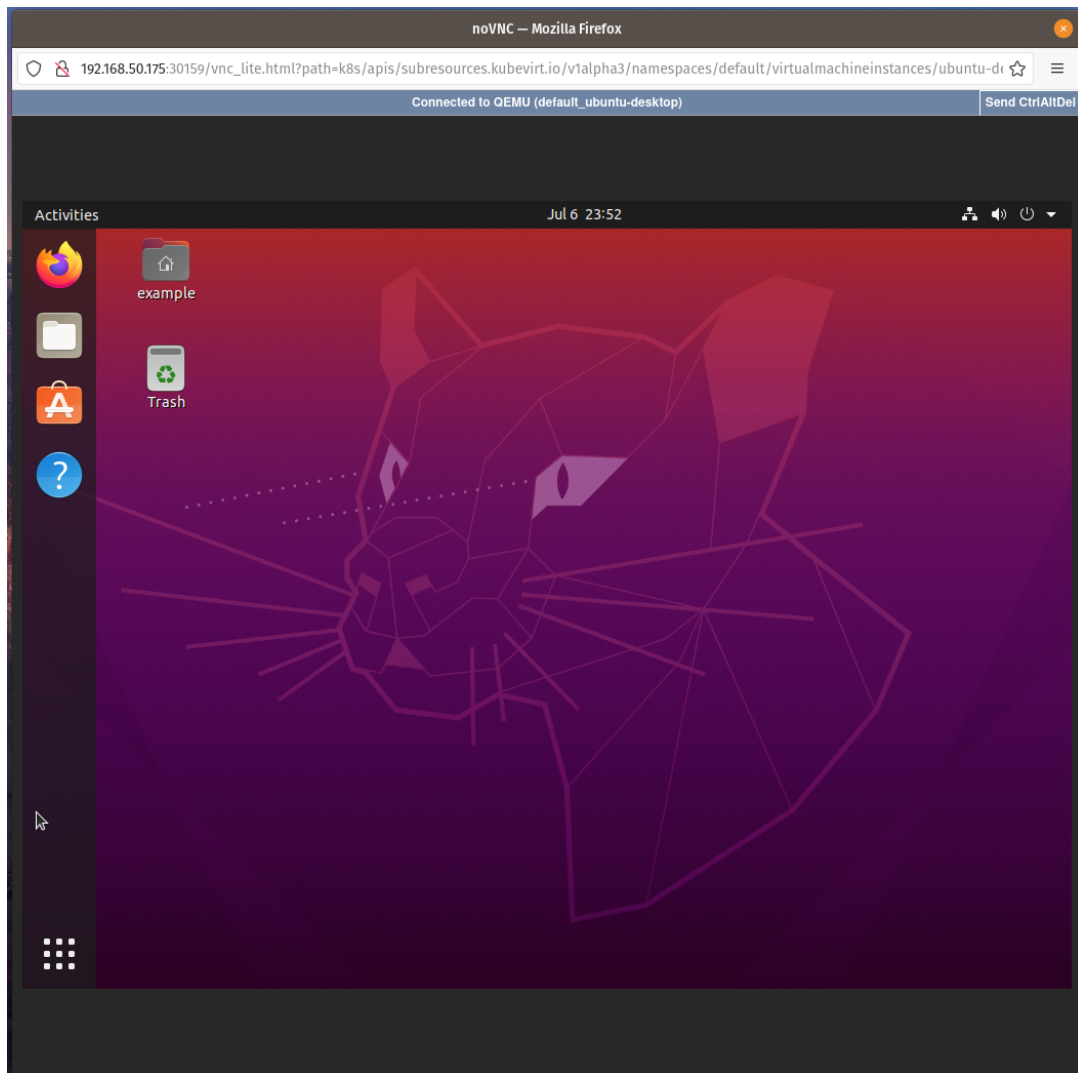


Figure 18: virtVNC showing Gnome

Download noVNC with `git clone https://github.com/novnc/noVNC`. Then `cd` into the folder and run `python -m SimpleHTTPServer`. Now you are serving the noVNC files over your localhost. Check it by opening `localhost:8000/vnc.html` in your browser. If you see the noVNC client, it's working. [61]

`vnc.html` has three parameters that are needed by us: `host`, `port` and `path`. `host` and `port` refers to the host and port the websocket has to connect to. `path` is the path that should be used. noVNC will build the websocket url as `host:port/path`. Other parameters can be found here: [Embedding and Deploying noVNC Application](#)⁵⁰. [63]

Now you have a running noVNC server and can use the Kubernetes API unauthenticated. In my case `kubectl proxy` runs on port 8001 and noVNC (with SimpleHTTPServer) on 8000.

The Kubernetes API server lets you query and manipulate the state of API objects in Kubernetes. Every resource we have used is available over the Kubernetes API and can be created and viewed over the API. For example pods, namespaces, VMs and volumes are all API objects. [64]

The Kubernetes API is a self describing API. And there are some URLs that explain the sub-URLs. For example some of the API methods/resources that we can use are listed in the following two URLs:

- `http://localhost:8001/apis/kubevirt.io/v1alpha3/`
- `http://localhost:8001/apis/subresources.kubevirt.io/v1alpha3`

The first contains KubeVirt resources like VMs and VMIs and the second contains KubeVirt subresources like the VNC and console. Notice: The second URL does not work if you append a trailing `/` to the URL.

As we already know some resources are available within namespaces, e.g. pods, VMs and VMIs. The API is structured like `<host>:<port>/apis/<resource_api>/<api_version>`. This will list all methods and resources available in this resource api. Then, when you want to open a resource and the resource is namespaced, it's structured like `<host>:<port>/apis/<resource_api>/<api_version>/namespaces/<namespace>/<resource>` and if you call a subresource `<host>:<port>/apis/<resource_api>/<api_version>/namespaces/<namespace>/<resource>/<subresource>`.

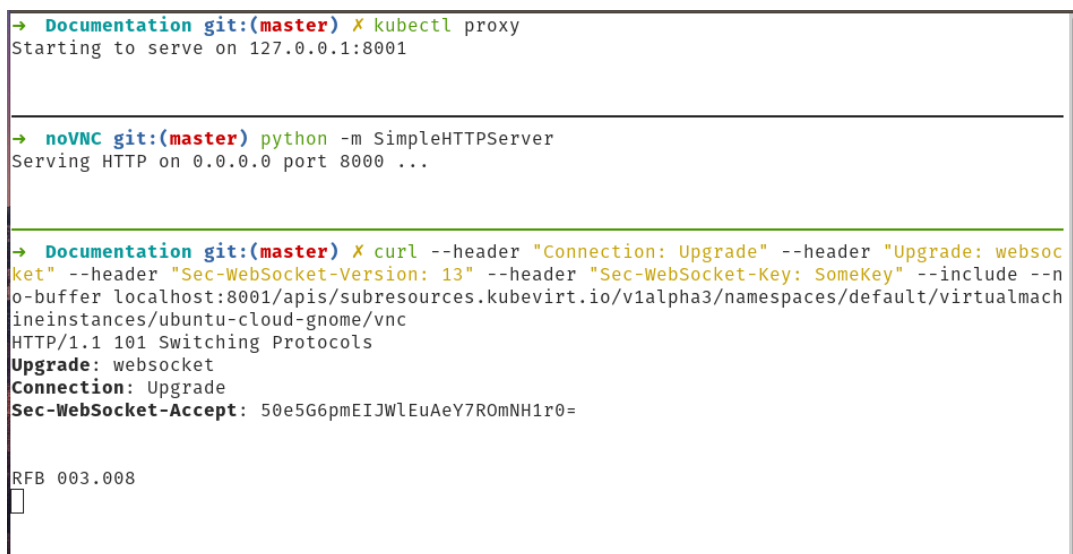
To get all VMIs from namespace default you can make a GET request to the URL: `http://localhost:8001/apis/kubevirt.io/v1alpha3/namespaces/default/virtualmachineinstances/`. VirtVNC uses this in the script to list all VMIs with status and name of the VMI. The result of the request contains information about the VMI. This contains the name (`metadata.name`) of the VMI, the phase (`status.phase`; e.g. "Running", "Failed"), the nodeName where the VMI is running (`status.nodeName`), the internal ip of the VMI (`status.interfaces[0].ipAddress`) and many other information for example the cloud-init configuration that contains our root password. This information maybe needs to be hidden from the user. The VMI name is used to identify the VMI in other calls. For example you can make a GET request to

⁵⁰<https://github.com/novnc/noVNC/blob/master/docs/EMBEDDING.md>

http://localhost:8001/apis/kubevirt.io/v1alpha3/namespaces/default/virtualmachineinstances/<your_vmi_name>. This gives you only the details to the specific VMI and not a list of all VMIs.

VirtVNC uses the above information to display all VMIs in the select page. Now when you click on one of the VMIs in VirtVNC it opens a new window with a link to the `vnc_lite.html` and the parameter path set to `apis/subresources.kubevirt.io/v1alpha3/namespaces/default/virtualmachineinstances/<your_vmi_name>/vnc`. This API URL is the one pointed out at the beginning that contains the Websocket api access for VNC.

To test if your WebSocket works, you first need to assure that the VMI is running. Then you can execute `curl --header "Connection: Upgrade" --header "Upgrade: websocket" --header "Sec-WebSocket-Version: 13" --header "Sec-WebSocket-Key: SomeKey" --include --no-buffer localhost:8001/apis/subresources.kubevirt.io/v1alpha3/namespaces/default/virtualmachineinstances/<your_vmi_name>/vnc`. If curl doesn't throw an exception and remains in a running connection, the WebSocket works. [65]



```
→ Documentation git:(master) ✕ kubect1 proxy
Starting to serve on 127.0.0.1:8001

→ noVNC git:(master) python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...

→ Documentation git:(master) ✕ curl --header "Connection: Upgrade" --header "Upgrade: websocket" --header "Sec-WebSocket-Version: 13" --header "Sec-WebSocket-Key: SomeKey" --include --no-buffer localhost:8001/apis/subresources.kubevirt.io/v1alpha3/namespaces/default/virtualmachineinstances/ubuntu-cloud-gnome/vnc
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: 50e5G6pmEIJWLEuAeY7R0mNH1r0=

RFB 003.008
□
```

Figure 19: Testing of Websockets with curl

Figure 19 shows the `kubevirt proxy` in the first window, the `SimpleHTTPServer` serving the `noVNC` client in the second window and a working `curl-WebSocket` test in the third window.

Now you can open you own `noVNC` Url in the browser (<http://localhost:8000/vnc.html>) with the parameters `host=localhost`, `port=8001` and `path=apis/subresources.kubevirt.io/v1alpha3/namespaces/default/virtualmachineinstances/<your_vmi_name>/vnc`. In my case this results in the URL: <http://localhost:8000/vnc.html?host=localhost&port=8001&path=apis/subresources.kubevirt.io/v1alpha3/namespaces/default/virtualmachineinstances/ubuntu-cloud-gnome/vnc>.

The figures 20-22 shows our own custom `noVNC` instance connected to the VNC of the KubeVirt VMIs over the Kubernetes API.



Figure 20: Selfhostet noVNC interface

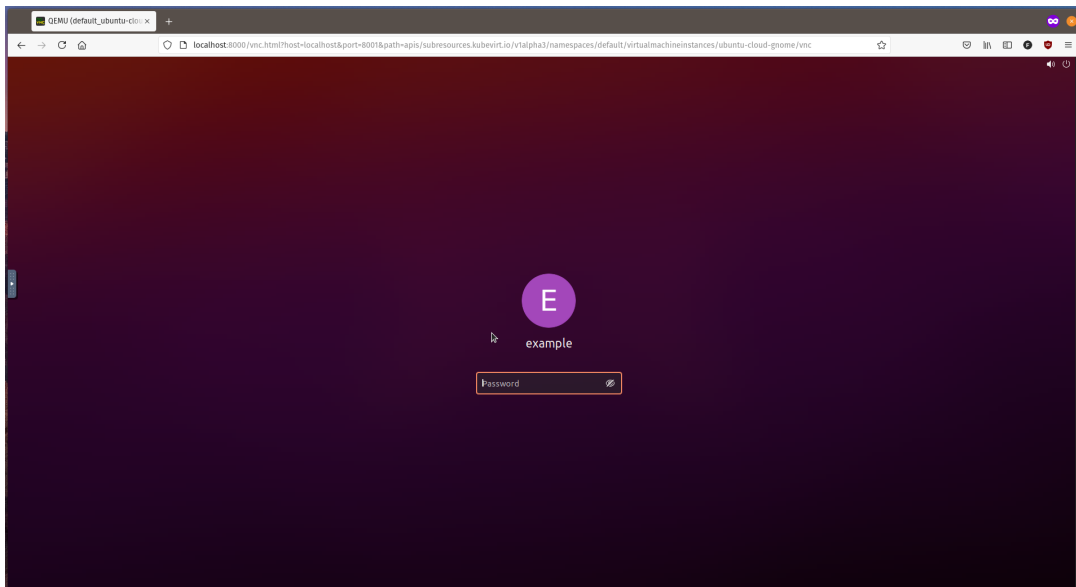


Figure 21: Selfhostet noVNC connected to VMI with login screen

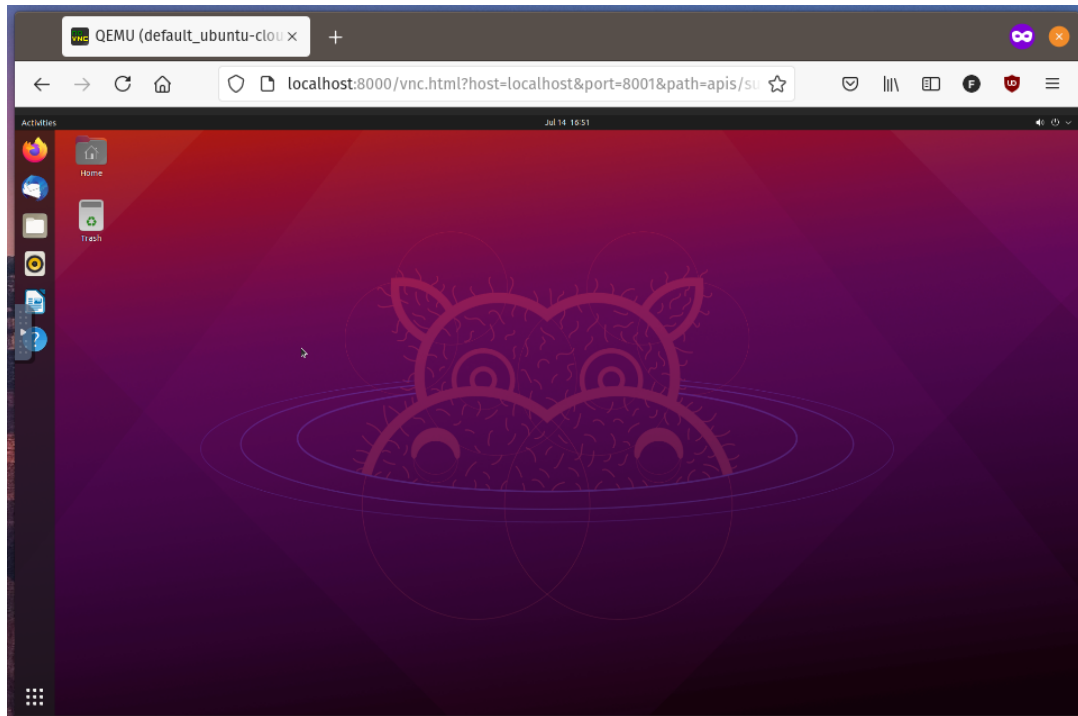


Figure 22: Selfhostet noVNC connected to VMI logged in

4.3 Base images

4.4 Web access to terminal

4.5 Web access to graphical user interface

<https://kubevirt.io/2019/Access-Virtual-Machines-graphic-console-using-noVNC.html>

4.6 Integration of terminal and graphical user interface web access to docker base image

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

4.8 Integration of base images in Kubernetes

4.9 Routing of base images in Kubernetes

4.10 Multi-user support

4.11 Separation of labs

In this step we want to archive a separation of labs. That means that if you are connected to one lab, you can't access another lab from there. For example you are connected to a VMI in the lab 1 and the VMI in lab 2 has the ip 10.244.120.80 and you try to ssh into the machine it should not work.

To archive this NetworkPolicy is needed. NetworkPolicy only work if you are running a network plugin that supports network policies for example calico. If you are using Minikube you can start the cluster with the network plugin calico with the following parameter: `--cni=calico`. If you haven't done this during the first start of Minikube, you need to delete your cluster and create a new one. If you don't do this, Kubernetes won't throw any errors in the next steps, but in the last step it will not fail when you ping the VMI in namespace lab1. [22] [21] [66]

To check if we can reach other namespaces create a new namespace using `kubectl create namespace testing`. Then create a second VMI yaml file and add namespace: testing to metadata. Then start the second VMI. Now you should have two VMIs: You can see the first one when you execute `kubectl get vmi` and the second when you execute `kubectl get vmi -n testing`. The command also shows you the ips of the VMIs. For reference my VMIs are called `ubuntu-cloud-gnome` and `ubuntu-cloud-gnome2`. Also the images have a default password set for the user `example` with the `gnome-boxes` way which was explained earlier.

Now connect to the console of one VMI: `kubectl virt console ubuntu-cloud-gnome2 -n testing` or `kubectl virt console ubuntu-cloud-gnome`. Then connect to the ip of the other machine over ssh: `ssh example@172.17.0.18`. If that works, the connection between namespaces is allowed. With this method we can check if the network policies work. Now delete the namespace testing and the two VMIs.

In the Lab Orchestrator every lab gets its own namespace and in this namespace there can be multiple VMs that you can connect to. To depict this create two namespaces called lab1 and lab2.

Network policies are bound to a namespace and only apply to this namespace. So if you create a network policy in the namespace lab1 that denies all traffic to other namespaces, this only affects the namespace lab1. Because of this behavior we need to create one of these network policies in every namespace that we use as a lab. The network policy configuration that denies traffic to other namespaces can be found in the examples of [KubeVirt's NetworkPolicy Documentation](#)⁵¹. Create two of them, one for namespace lab1 and one for lab2. You can display them with the command `kubectl get networkpolicy --all-namespaces` like in Figure 23. [66]

```
→ prototype git:(master) ✗ kubectl get networkpolicy --all-namespaces
NAMESPACE   NAME                               POD-SELECTOR  AGE
lab1         allow-same-namespace              <none>        13m
lab2         allow-same-namespace              <none>        13m
→ prototype git:(master) ✗
```

Figure 23: NetworkPolicies in Namespaces lab1 and lab2

⁵¹https://kubevirt.io/user-guide/virtual_machines/networkpolicy/

Listing 19 Create two namespaces (prototype/examples/namespaces.yaml)

```
1 kind: Namespace
2 apiVersion: v1
3 metadata:
4   name: lab1
5 ---
6 kind: Namespace
7 apiVersion: v1
8 metadata:
9   name: lab2
```

Listing 20 NetworkPolicy allow same namespace (prototype/examples/network_policy_allow_same_namespace.yaml)

```
1 kind: NetworkPolicy
2 apiVersion: networking.k8s.io/v1
3 metadata:
4   namespace: lab1
5   name: allow-same-namespace
6 spec:
7   podSelector:
8     matchLabels:
9   ingress:
10    - from:
11      - podSelector: {}
12 ---
13 kind: NetworkPolicy
14 apiVersion: networking.k8s.io/v1
15 metadata:
16   namespace: lab2
17   name: allow-same-namespace
18 spec:
19   podSelector:
20     matchLabels:
21   ingress:
22    - from:
23      - podSelector: {}
```

Listing 19 creates two namespaces: lab1 and lab2. **Listing 20** creates two network policies that denies all traffic to other namespaces for the namespaces lab1 and lab2. [67]

After that you can start some VMIs in these namespaces. For example we run one VMI in lab1 and two in lab2. Now get the ip addresses of them with the command `kubectl get vmi --all-namespaces` like in Figure 24.

```
Every 2,0s: kubectl get vmi --all-namespaces      pop-os: Thu Jul 15 17:07:36 2021
```

NAMESPACE	NAME	AGE	PHASE	IP	NODENAME
lab1	ubuntu-cloud-gnome	12m	Running	10.244.120.77	minikube
lab2	ubuntu-cloud-gnome	2m27s	Running	10.244.120.79	minikube
lab2	ubuntu-cloud-gnome2	2m27s	Running	10.244.120.80	minikube

Figure 24: Multiple VMIs in different namespaces

To check if the network policies work connect to one VMI in lab2 with the command `kubectl virt console ubuntu-cloud-gnome -n lab2`. Then ping the ip of the VMI in lab1. If that doesn't work, the policy is working. After that ping the ip of the second VMI in lab2. That should work. Figure 25 shows an example of this step.

```
root@ubuntu-cloud-gnome:~# ip a | grep "inet 10"
    inet 10.244.120.79/32 scope global dynamic enp1s0
root@ubuntu-cloud-gnome:~# ping 10.244.120.77
PING 10.244.120.77 (10.244.120.77) 56(84) bytes of data.
^C
--- 10.244.120.77 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4095ms

root@ubuntu-cloud-gnome:~# ping 10.244.120.80
PING 10.244.120.80 (10.244.120.80) 56(84) bytes of data.
64 bytes from 10.244.120.80: icmp_seq=1 ttl=63 time=1.05 ms
64 bytes from 10.244.120.80: icmp_seq=2 ttl=63 time=1.83 ms
64 bytes from 10.244.120.80: icmp_seq=3 ttl=63 time=0.366 ms
64 bytes from 10.244.120.80: icmp_seq=4 ttl=63 time=0.767 ms
64 bytes from 10.244.120.80: icmp_seq=5 ttl=63 time=0.758 ms
64 bytes from 10.244.120.80: icmp_seq=6 ttl=63 time=0.345 ms
^C
--- 10.244.120.80 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5010ms
rtt min/avg/max/mdev = 0.345/0.852/1.830/0.500 ms
root@ubuntu-cloud-gnome:~#
```

Figure 25: Check if network policies are working

Now we are able to deploy multiple VMIs in different namespaces and separate these namespaces from each other, so that if you are connected to one namespace you can only navigate inside this namespace. For every user we need to create one namespace, one network policy and any amount of VMIs.

4.11.1 Authorization

KubeVirt Authorization⁵²

⁵²<https://kubevirt.io/user-guide/operations/authorization/>

List of Figures

1	How Ingress interacts with Services and Pods [12]	10
2	Example of Console Login	31
3	Example of SSH	32
4	Example of SSH Setup Login	33
5	Image size before change	35
6	Image size after change	35
7	Disk size before change	35
8	fdisk partition size before change	36
9	fdisk delete partition and create new	36
10	fdisk partition size after change and write changes	37
11	Resize filesystem and disk size after change	37
12	ttyd running inside the container	39
13	ttyd running outside the container	40
14	noVNC and vnc proxy	42
15	noVNC in browser	43
16	virtVNC list of VMs	43
17	virtVNC showing a console	44
18	virtVNC showing Gnome	45
19	Testing of Websockets with curl	47
20	Selfhostet noVNC interface	48
21	Selfhostet noVNC connected to VMI with login screen	48
22	Selfhostet noVNC connected to VMI logged in	49
23	NetworkPolicies in Namespaces lab1 and lab2	50
24	Multiple VMIs in different namespaces	52
25	Check if network policies are working	52

List of Listings

1	Example of a Service	9
2	Example of an Ingress	10
3	Example VM (prototype/examples/vm.yaml)	17
4	Example VirtualMachineInstancePreset	18
5	Example VMI, that matches the correct labels	19
6	Example of VMI with PVC	21
7	Dockerfile example with local qcow2 image	22
8	Dockerfile example with remote qcow2 image	22
9	Example VMI with Container Disk	22
10	Example Network and Interface	24
11	Example of autoattachPodInterface	24
12	Example of NetworkPolicy	25
13	Example VMI with Labels	26
14	Example VMI exposed as Service	26
15	Example VM with cloud-init NoCloud	28
16	Example Dockerfile for Custom Image	29
17	Example Container Disk for Custom Image	30
18	Insert SSH Key in Minikube	31
19	Create two namespaces (prototype/examples/namespaces.yaml)	51
20	NetworkPolicy allow same namespace (prototype/examples/network_policy_allow_same_namespace.yaml)	51

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. Signale | ubuntuusers. Retrieved June 1, 2021 from <https://wiki.ubuntuusers.de/Signale/>
4. Nohup | ubuntuusers. 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 | kubevirt. 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. Minikube increase disk size. Retrieved July 6, 2021 from <https://www.maxoberberger.net/blog/2018/09/minikube-increase-disk-size.html>
20. Build your kubernetes armory with minikube, kail, and kubens. Retrieved July 6, 2021 from https://developers.redhat.com/blog/2019/04/16/build-your-kubernetes-armory-with-minikube-kail-and-kubens#__minikube
21. Alternative containerlaufzeitumgebungen | kubernetes. Retrieved July 15, 2021 from <https://kubernetes.io/de/docs/setup/minikube/#alternative-containerlaufzeitumgebungen>
22. Networking und network policy | kubernetes. Retrieved July 15, 2021 from <https://kubernetes.io/de/docs/concepts/cluster-administration/addons/>

23. Use kubevirt | kubevirt. Retrieved June 5, 2021 from <https://kubevirt.io/labs/kubernetes/lab1.html>
24. Architecture | kubevirt. Retrieved June 5, 2021 from <https://kubevirt.io/user-guide/architecture/>
25. Lifecycle | kubevirt. Retrieved June 5, 2021 from https://kubevirt.io/user-guide/virtual_machines/lifecycle/
26. Run strategies | kubevirt. Retrieved June 5, 2021 from https://kubevirt.io/user-guide/virtual_machines/run_strategies/
27. Presets | kubevirt. Retrieved June 6, 2021 from https://kubevirt.io/user-guide/virtual_machines/presets/
28. Disks and volumes | kubevirt. Retrieved June 7, 2021 from https://kubevirt.io/user-guide/virtual_machines/disks_and_volumes/
29. Top level api objects | kubevirt. Retrieved June 7, 2021 from <https://kubevirt.io/api-reference/v0.6.4/definitions.html>
30. Volumes | kubernetes. Retrieved June 10, 2021 from <https://kubernetes.io/docs/concepts/storage/volumes/>
31. Storage classes | kubernetes. Retrieved June 10, 2021 from <https://kubernetes.io/docs/concepts/storage/storage-classes/#local>
32. Persistent volumes | kubernetes. Retrieved June 10, 2021 from <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims>
33. Interfaces and networks | kubevirt. Retrieved June 10, 2021 from https://kubevirt.io/user-guide/virtual_machines/interfaces_and_networks/
34. Multus. Retrieved June 10, 2021 from <https://github.com/k8snetworkplumbingwg/multus-cni>
35. NetworkPolicy | kubevirt. Retrieved June 14, 2021 from https://kubevirt.io/user-guide/virtual_machines/networkpolicy/
36. Network policies | kubernetes. Retrieved June 14, 2021 from <https://kubernetes.io/docs/concepts/services-networking/network-policies/>
37. VirtualMachineInstanceReplicaSet | kubevirt. Retrieved July 3, 2021 from https://kubevirt.io/user-guide/virtual_machines/replicaset/
38. Service objects | kubevirt. Retrieved July 3, 2021 from https://kubevirt.io/user-guide/virtual_machines/service_objects/
39. Cloud-init documentation | cloud-init. Retrieved June 14, 2021 from <https://cloudinit.readthedocs.io/en/latest/index.html>
40. Availability | cloud-init. Retrieved June 14, 2021 from <https://cloudinit.readthedocs.io/en/latest/topics/availability.html>
41. Boot stages | cloud-init. Retrieved June 14, 2021 from <https://cloudinit.readthedocs.io/en/latest/topics/boot.html>
42. Datasources | cloud-init. Retrieved June 14, 2021 from <https://cloudinit.readthedocs.io/en/latest/topics/datasources.html>
43. NoCloud | cloud-init. Retrieved June 14, 2021 from <https://cloudinit.readthedocs.io/en/latest/topics/datasources/nocloud.html>
44. Getting to know kubevirt | kubernetes. Retrieved June 14, 2021 from <https://kubernetes.io/blog/2018/05/22/getting-to-know-kubevirt/>
45. Build context for docker image very large | stackoverflow. Retrieved July 12, 2021 from <https://stackoverflow.com/a/26604307>

46. Best practices for writing dockerfiles | docker docs. Retrieved July 12, 2021 from https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#understand-build-context
47. Where does boxes store disk images? | gnome help. Retrieved July 4, 2021 from <https://help.gnome.org/users/gnome-boxes/stable/disk-images.html.en>
48. How to customize qcow2/raw linux os disk image with virt-customize | computing for geeks. Retrieved July 4, 2021 from <https://computingforgeeks.com/customize-qcow2-raw-image-templates-with-virt-customize/>
49. How to extend/increase kvm virtual machine (vm) disk size | computing for geeks. Retrieved July 4, 2021 from <https://computingforgeeks.com/how-to-extend-increase-kvm-virtual-machine-disk-size/>
50. Use ubuntu cloud image with kvm | medium. Retrieved July 4, 2021 from <https://medium.com/@art.vasilyev/use-ubuntu-cloud-image-with-kvm-1f28c19f82f8>
51. How do you increase a kvm guest's disk space? | server fault. Retrieved July 4, 2021 from <https://serverfault.com/questions/324281/how-do-you-increase-a-kvm-guests-disk-space>
52. Resizing a filesystem using qemu-img and fdisk | github gist. Retrieved July 4, 2021 from <https://gist.github.com/larsks/3933980>
53. Ttyd - share your terminal over the web | kubernetes. Retrieved July 5, 2021 from <https://github.com/tsl0922/ttyd>
54. Ttyd - share your terminal over the web | kubernetes. Retrieved July 5, 2021 from <https://github.com/tsl0922/ttyd>
55. Use a service to access an application in a cluster | kubernetes. Retrieved July 5, 2021 from <https://kubernetes.io/docs/tasks/access-application-cluster/service-access-application-cluster/>
56. Connect to qemu/kvm via noVNC and websockify | programmer sought. Retrieved July 6, 2021 from <https://www.programmersought.com/article/19755825712/>
57. Accessing virtual machines. Retrieved July 6, 2021 from https://kubevirt.io/user-guide/virtual_machines/accessing_virtual_machines/
58. I getting always“ no space left on device” issue in minikube even system have free space , how to resolve? | stackoverflow. Retrieved July 6, 2021 from <https://stackoverflow.com/questions/62998012/i-getting-always-no-space-left-on-device-issue-in-minikube-even-system-have-fr>
59. How to install cloud-init package on ubuntu. Retrieved July 6, 2021 from <https://zoomadmin.com/HowToInstall/UbuntuPackage/cloud-init>
60. How to create desktop cloud image of ubuntu on openstack environment? | github gist. Retrieved July 6, 2021 from <https://gist.github.com/sasukeh/0a50ca570722df3a4a65>
61. Access virtual machines' graphic console using noVNC. Retrieved July 14, 2021 from <https://kubevirt.io/2019/Access-Virtual-Machines-graphic-console-using-noVNC.html>
62. How to quickly serve files and folders over http in linux. Retrieved July 14, 2021 from <https://ostechnix.com/how-to-quickly-serve-files-and-folders-over-http-in-linux/>
63. Embedding and deploying noVNC application | github. Retrieved July 14, 2021 from <https://github.com/novnc/novnc/blob/master/docs/EMBEDDING.md>
64. The kubernetes api | kubernetes. Retrieved July 14, 2021 from <https://kubernetes.io/docs/concepts/overview/kubernetes-api/>
65. Test a websocket using curl. | github gist. Retrieved July 14, 2021 from <https://gist.github.com/httpfbce19069187ec1cc486b594104f01d0>
66. Network plugins | kubernetes. Retrieved July 15, 2021 from <https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/>

67. Namespaces walkthrough | kubernetes. Retrieved July 15, 2021 from <https://kubernetes.io/docs/tasks/administer-cluster/namespaces-walkthrough/>