Distributed Systems Administration

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Deployement of a distributed storage infrastucture over K3s cluster, on a dev environment with Vagrant over Virtualbox

Francisco Javier Pizarro Martínez 821259

29/05/2023

INDEX

Abstract

Now with the core concepts of Vagrant and K8s already learned we are about to deploy a more complex system on this envoiroment, specifically we'll be working with a Ceph storage system in a distributed infrastructure, after sucessfully deploying this in order to propperly test it a Wordpress website with the corresponding mysql database supporting it will be deployed, it also will be tested with the use of a container registry.

In addition we will be looking forward to use a more advanced tool for the provision of our VMs, in this case this tool is going to be Puppet, also we want to play around on how to use multiple cloud providers in a more automatic way to deploy, we will play a little bit with terraform to learn the basics.

This project is splitted into 5º different parts, in the first one the Ceph cluster is set up and tested in order to see if it works properly, the second part consists of setting up a wordpress web and a mysql database running in the k3s infrastructure, also has a more manual deploy that tests the basic storage system, the third one of the parts is the deployment of a more complex app that is supported by a distributed filesystem, this app is a private container repository.

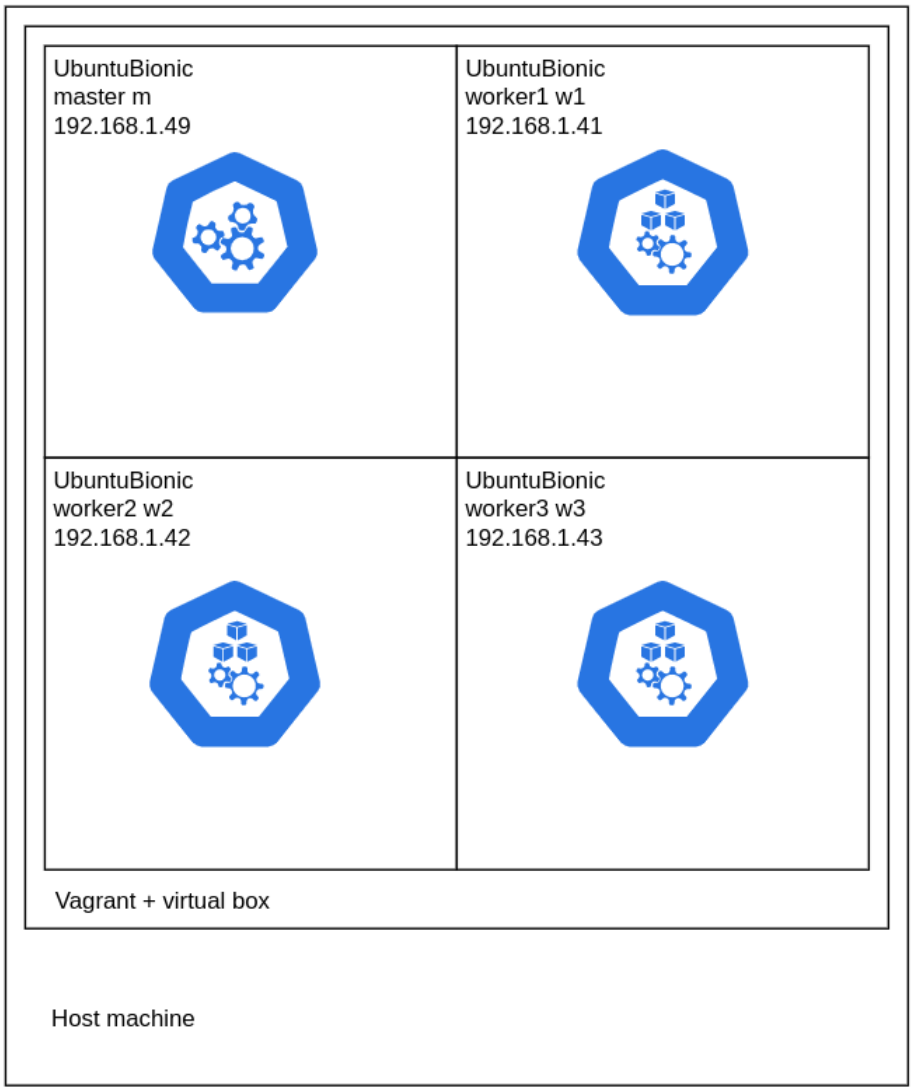
The last 2 parts are about using more high level tools in order to provision the VMs and to generate the VMs by using Puppet and Terraform respectively.

The commands that will be used during this project are this ones:

- kubectl ...
- vagrant ...
- ceph ...
- rados ...
- docker ...
- terraform ...

System architecture

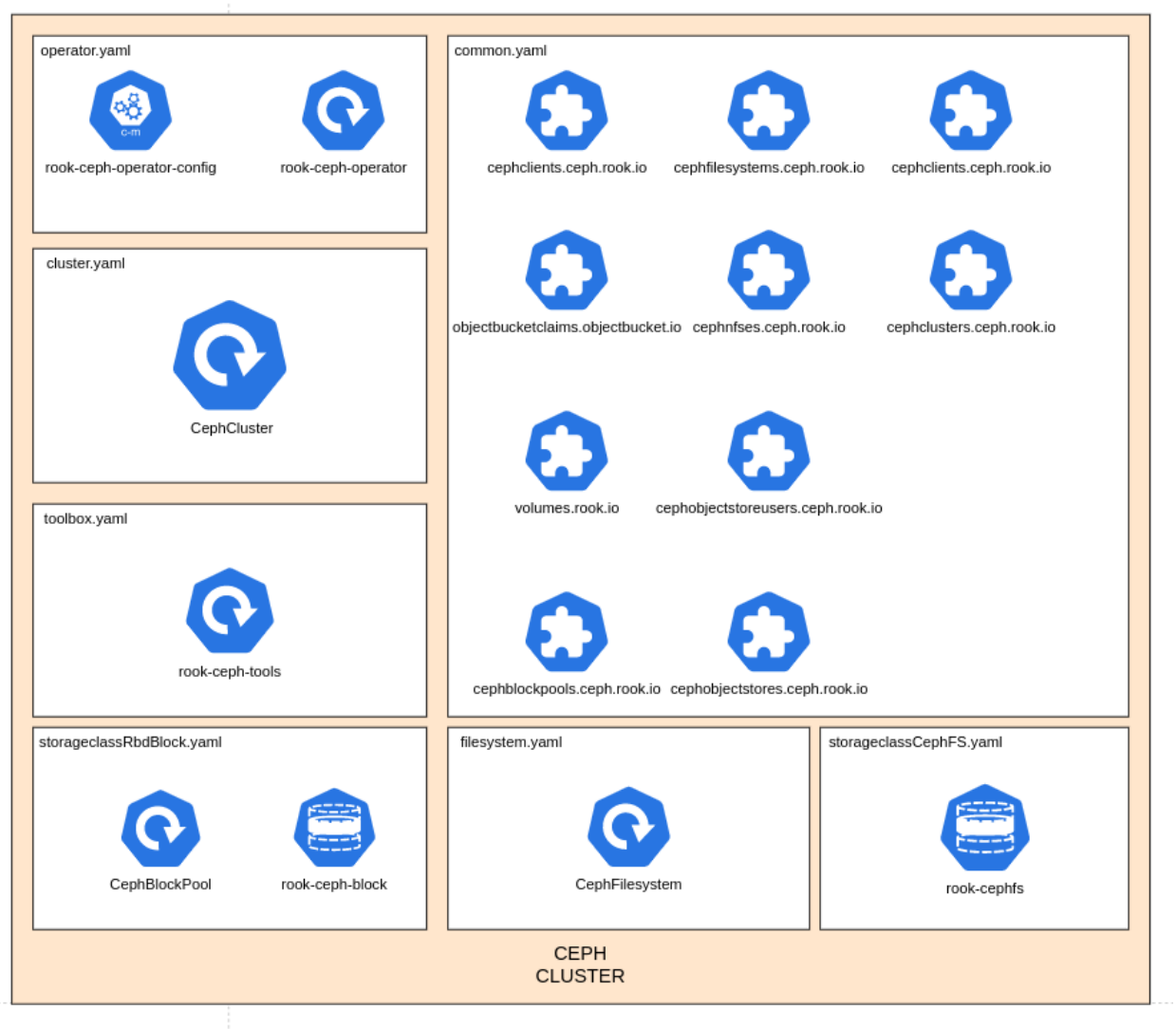
Before going into the architecture of the things that are going to be deployed on our K3s cluster we need to define the architecture of the K3s cluster, in this particular case we are running 4 VMs with Vagrant using Virtual box under the hood, 3 of this VMs are workers, the last one is the master and doesnt execute "common" tasks like Pods.
All this virtual machines are hosted in a single host machine, this machines run ubuntu bionic as OS.



We have 3 clearly separated parts which are:The ceph cluster, the website and lastly the container registry. Each one has a different internal architecture.

The ceph cluster is defined in this parts:

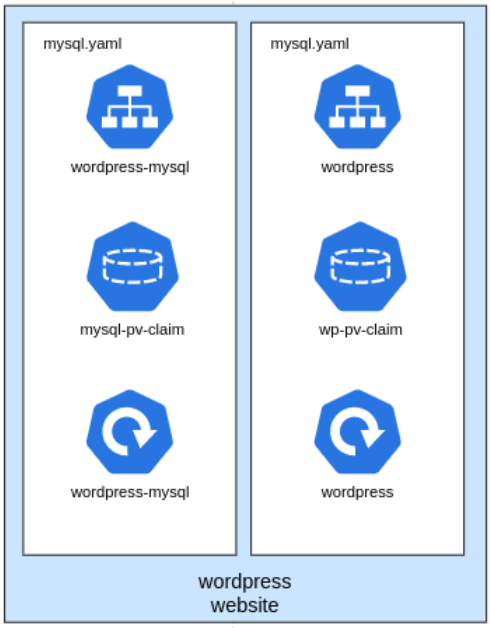
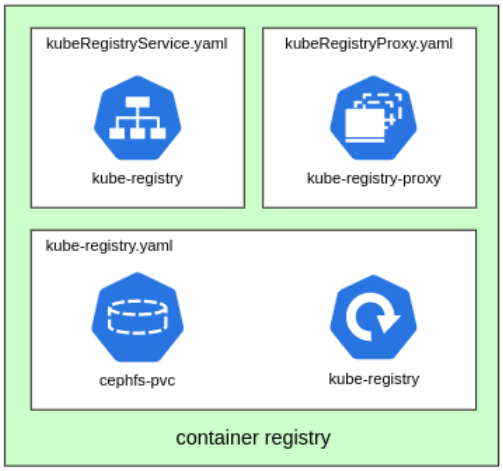
- Custom resources.
- Operators that are needed to launch the whole cluster later.
- The whole cluster deploy.
- Testing Pod.
- PVC and deploy for RBD.
- PVC and deploy for filesystem.



The internal structure of both the registry and the website have a lot of similarities, they both use:

- Service for allowing the traffic to reach the Pods
- PV for using the PVC created and use the ceph storage.
- Deployment to launch all the pods,

The only architectural difference is that additionally the registry has a Proxy.

Wordpress website	**Container registry**
	

Deploy of the basic Ceph cluster

First of all we perform some basic changes on our settings, this are detailed in the annex II, after doing this our starting point is to define our custom resources and to start the whole basic cluster, to achive this we perform the following commands:

```
kubectl create -f common.yaml
kubectl create -f operator.yaml
kubectl -n rook-ceph get pod
kubectl create -f cluster.yaml
kubectl -n rook-ceph get pod
```

The previous step can take a while until it achieves a stable status, after reaching this we can test the storage capabilities of the cluster and view the current status by running this:

```
kubectl create -f toolbox.yaml
kubectl -n rook-ceph get pod
kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-ceph-tools" -o
jsonpath='{.items[0].metadata.name}') bash
  ceph status
  #  cluster:
  #    id:     556d8ecc-a9f5-4e7e-a386-a0d62c77fc5a
  #    health: HEALTH_WARN
  #            clock skew detected on mon.b, mon.c

  # services:
  #    mon: 3 daemons, quorum a,b,c (age 16m)
  #    mgr: a(active, since 11m)
  #    osd: 3 osds: 3 up (since 11m), 3 in (since 11m)

  # data:
  #    pools:   0 pools, 0 pgs
  #    objects: 0 objects, 0 B
  #    usage:   3.0 GiB used, 87 GiB / 90 GiB avail
  #    pgs:

  ceph df
  RAW STORAGE:
  # CLASS     SIZE       AVAIL      USED        RAW USED     %RAW USED
  # hdd       90 GiB     87 GiB     4.7 MiB      3.0 GiB          3.34
  # TOTAL     90 GiB     87 GiB     4.7 MiB      3.0 GiB          3.34

  # POOLS:
  # POOL     ID     STORED     OBJECTS     USED     %USED     MAX AVAIL
  rados df
  # POOL_NAME USED OBJECTS CLONES COPIES MISSING_ON_PRIMARY UNFOUND DEGRADED RD_OPS RD WR_OPS WR USED COMPR UNDER
COMPR

  # total_objects    0
  # total_used       3.0 GiB
  # total_avail      87 GiB
  # total_space      90 GiB
kubectl -n rook-ceph get events
```

Deploy of website and RDB storage

Now with the cluster working we can start to use it, in this case we are going to set up a basic RDB storage system and also a wordpress website with its own mysql database which both of them will be supported by this storage system.
Before doing nothing else we need to modify the image in direct-mount.yaml and use the same as toolbox.yaml.
Now with that fixed we can start creating our storage system and our website by running this commands:

```
kubectl create -f storageclassRbdBlock.yaml
kubectl create -f mysql.yaml;kubectl create -f wordpress.yaml;kubectl get pods
curl 192.168.1.49;wget 192.168.1.49
```

After testing this we can also do a basic Pod which has access to our storage system to do more specific tests, to do it we just need to execute the following commands:

```
kubectl create -f direct-mount.yaml
kubectl -n rook-ceph get pod -l app=rook-direct-mount; kubectl -n rook-ceph exec -it rook-direct-mount-95b6b4d88-
zr75q bash
  # rbd create replicapool/test --size 10
  # rbd info replicapool/test
    # rbd image 'test':
    #         size 10 MiB in 3 objects
    #         order 22 (4 MiB objects)
    #         snapshot_count: 0
    #         id: 38e6702fc31f
    #         block_name_prefix: rbd_data.38e6702fc31f
    #         format: 2
    #         features: layering
    #         op_features:
    #         flags:
    #         create_timestamp: Tue May 23 13:24:44 2023
    #         access_timestamp: Tue May 23 13:24:44 2023
    #         modify_timestamp: Tue May 23 13:24:44 2023
  # lsblk
  #   NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
  #   sdb 8:16   0  10M  0 disk
  #   sdc 0  30G  0 disk
  #   `-ceph--400b4de8--1511--4ebf--b173--f1c8b15faf13-osd--data--b472534f--ae14--4a1a--aa8a--0493d4092e58 253:0
0  30G  0 lvm
  #   sda 8:0    0  40G  0 disk
  #   `-sda1 8:1    0  40G  0 part /etc/resolv.conf
  # rbd map replicapool/test
    # lsblk | grep rbd
    # rbd0 252:0    0  10M  0 disk
  # mkfs.ext4 /dev/rbd1;mkdir /tmp/testing
  # mount /dev/rbd1 /tmp/testing
  # touch /tmp/testing/a.txt
  # echo "zzz" > /tmp/testing/sleep.txt
kubectl get pods --all-namespaces -o wide
vagrant ssh w2
  # sudo shutdown now
kubectl -n rook-ceph get pod -l app=rook-direct-mount; kubectl -n rook-ceph exec -it rook-direct-mount-68ccfbd4f5-
2qwdl bash
  # cat /tmp/testing/sleep.txt
    # zzz
  # rbd unmap /dev/rbd0
```

Deploy of containers registry and filesystem over ceph

Before starting this part we need to create a basic container that later on will be pushed to and pulled of the registry. For testing purposes we are going to use a basic container running an Alpine Linux with a shell inside it, here are the contents of the Dockerfile:

```
FROM alpine:latest

# Update package repositories and install bash
RUN apt-get update && apt-get install -y bash

# Set the default command to run bash
CMD [ "shell" ]
RUN sleep 360000
```

Now we need to build the container and tag it:

```
docker build -t testingpodman .
docker tag testingpodman localhost:5000/testingpodman
```

In order to have a normal filesystem behaviour in our ceph cluster we need to add some special nodes of type MDS, this store the metadata needed to run the use the filesystem, after upgrading up our cluster and do the appropriate checks, we can create the containers registry.

This is how to upgrade and check the cluster:

```
kubectl create -f filesystem.yaml
kubectl -n rook-ceph get pod -l app=rook-ceph-mds
kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-ceph-tools" -o
jsonpath='{.items[0].metadata.name}') bash
  # ceph status
  #   cluster:
  #     id:     a652c4d9-e312-4a7c-b371-fa10a0823bb9
  #     health: HEALTH_WARN
  #             clock skew detected on mon.b, mon.d

  #   services:
  #     mon: 3 daemons, quorum a,b,d (age 6m)
  #     mgr: a(active, since 6m)
  #     mds: myfs:1 {0=myfs-b=up:active} 1 up:standby-replay
  #     osd: 3 osds: 3 up (since 3m), 3 in (since 3m)

  #   data:
  #     pools:   3 pools, 96 pgs
  #     objects: 22 objects, 2.2 KiB
  #     usage:   3.0 GiB used, 87 GiB / 90 GiB avail
  #     pgs:     96 active+clean

  #   io:
  #     client:   853 B/s rd, 1 op/s rd, 0 op/s wr
```

And this is how to create the containers registry and prepare a portforward to test it.

```
kubectl create -f storageclassCephFS.yaml
kubectl create -f kubeRegistryService.yaml
kubectl create -f kube-registry.yaml
kubectl create -f kubeRegistryProxy.yaml
POD=$(kubectl get pods --namespace kube-system -l k8s-app=kube-registry -o template --template '{{range .items}}
{{.metadata.name}} {{.status.phase}}{{"\n"}}{{end}}'| grep Running | head -1 | cut -f1 -d' ')
kubectl port-forward --namespace kube-system $POD 5000:5000 &
```

Now to test that it runs as it should, we execute:

```
docker push localhost:5000/testingpodman
  # The push refers to repository [localhost:5000/testingpodman]
  # Handling connection for 5000
  # Handling connection for 5000
  # ded7a220bb05: Preparing
  # Handling connection for 5000
  # Handling connection for 5000
  # ded7a220bb05: Pushing  7.338MB
  # Handling connection for 5000
  # ded7a220bb05: Pushed
  # Handling connection for 5000
  # Handling connection for 5000
  # Handling connection for 5000
  # Handling connection for 5000
  # Handling connection for 5000
  # Handling connection for 5000
  # 1.0: digest: sha256:3481788b85037db3d9c0fb758181f965e8d75015ff6d3f9c765656e51e2978ab size: 528
docker pull localhost:5000/testingpodman
kubectl run -i --tty p --image=localhost:5000/testingpodman -- sh
```

Provissioning cluster nodes with Puppet

To level up our provissioner system we are going to use a most sofisticated tool that works also in distributed envoiroments, this tool is Puppet in order to use it we must adapt a few things in our Vagrantfile, this moddified version can be found on the annex IV.

The changes on our new Vagrantfile are the following, first we start by doing a basic shell provission which installs puppet agents on the VMs, after installing puppet we can do the jump to this new tool, in order to do this we must create a folder called `manifest` inside it we create the manifest vagrant_vm.pp, this one contains this:

```
node default {
  class { 'vagrant_vm':
    hostname  => $::hostname,
    nodeip    => $::nodeip,
    masterip  => $::masterip,
    nodetype  => $::nodetype,
  }
}

class vagrant_vm (
  String $hostname,
  String $nodeip,
  String $masterip,
  String $nodetype,
) {
  exec { 'set_timezone':
    command => 'timedatectl set-timezone Europe/Madrid',
    path    => '/bin:/usr/bin',
  }
  file { '/vagrant':
    ensure => 'directory',
    path   => '/vagrant',
  }
  host { 'hostname_entry':
    ensure  => present,
    name    => $nodeip,
    ip    => $nodeip,
    host_aliases => $hostname,
  }
  file { '/etc/hosts':
    ensure  => present,
    path    => '/etc/hosts',
    content => "192.168.0.49 m\n192.168.0.41 w1\n192.168.0.42 w2\n192.168.0.43 w3\n",
  }
  file { '/usr/local/bin/k3s':
  ensure  => present,
  source  => '/vagrant/k3s',
  mode    => '0755',
  require => File['/vagrant'],
  }
  if $nodetype == 'master' {
    exec { 'install_k3s_master':
      command => "env INSTALL_K3S_SKIP_DOWNLOAD=true /vagrant/install.sh server --token
'wCdC16AlP8qpqqI53DM6ujtrfZ7qsEM7PHLxD+Sw+RNK2d1oDJQQOsBkIwy5OZ/5' --flannel-iface enp0s8 --bind-address $nodeip --
-node-ip $nodeip --node-name $hostname --disable traefik --node-taint k3s-controlplane=true:NoExecute",
      path    => '/bin:/usr/bin',
      require => File['/usr/local/bin/k3s'],
    }
    exec { 'copy_k3s_yaml':
      command     => 'cp /etc/rancher/k3s/k3s.yaml /vagrant',
      path        => '/bin:/usr/bin',
      refreshonly => true,
      subscribe   => Exec['install_k3s_master'],
    }
  } else {
    exec { 'install_k3s_agent':
```

```
      command => "env INSTALL_K3S_SKIP_DOWNLOAD=true /vagrant/install.sh agent --server https://$masterip:6443 --
token 'wCdC16AlP8qpqqI53DM6ujtrfZ7qsEM7PHLxD+Sw+RNK2d1oDJQQOsBkIwy5OZ/5' --node-ip $nodeip --node-name $hostname -
-flannel-iface enp0s8",
      path    => '/bin:/usr/bin',
      require => File['/usr/local/bin/k3s'],
    }
  }
}
```

The manifest has 2 different parts, one which defines the class vagrant_vm and the other one which instanciates it(this is really important). The content of the vagrant_vm class is almost a complete traduction from our previous bash providing script.

This part of the project was executed in a Home Lab that is worth mentioning, the college lab has some really special situations like no root privileges and a max disk quota, so for avoiding more innecessary problems i set up the following Home Lab:

A raspberry server which is 24/7 running because it is hosting other projects.

A PC that can be remotly turned on via WOL, the raspberry is configured so it can wake up this PC.

In order to be able to access the PC with just 1 ssh jump and without using the raspberry as an intermediate point, it isnt listening in the default ssh port so you can connect to it and do some special things like the flag -X or ssh tunneling.

Deploying with Terraform

Terraform is a tool that give us a higher level of abstraction, by using this tool we can define in a declarative way the components that need to be up and running and the tool manages their deployment even across different cloud providers.

The objective here is to be able to install the tool and do a few simple test using a cloud provider, after understanding the basics of the tool we are going to deploy a K8s cluster with 3 nodes. This cluster for testing purposes will have just a simple deployment and a service.

First things first, we need to setup the tool and to config the cloud provider properly so we can use it, to install the tool we run: `snap install terraform`. After this we create a API token in the web UI of the cloud in this case Linode.

Basic webserver deploy

For the first example we'll be using a basic website with a database, in order to start we must set properly the terraform file, inside it first of all we define the provider and the plugin that it offers to interact with its own API, in this particular case linode, apart from setting up the provider we need to define each resource will be using in this case one machine will be hosting the webserver while the other one is running the database, each cloud provider has its own parameters to define the resources but usually they are similar.

fichero tf.main:

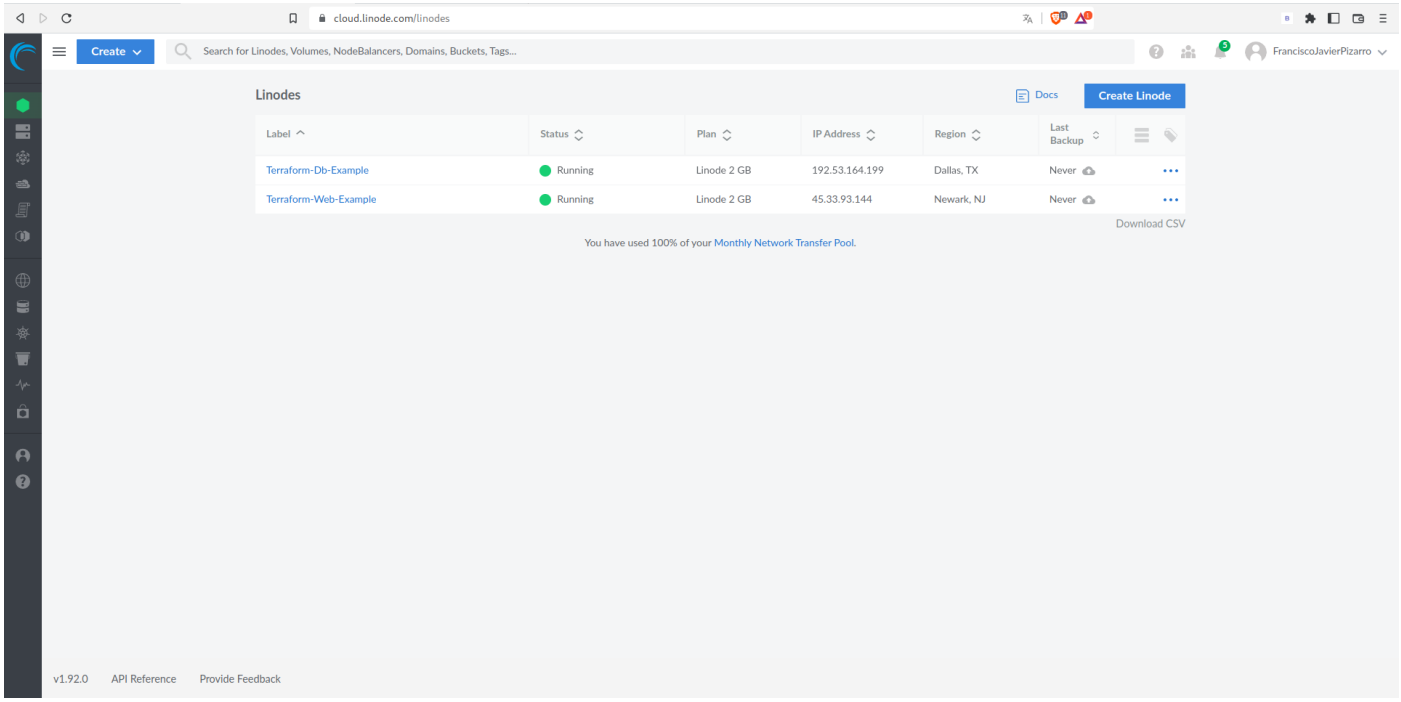
```
terraform {
  required_providers {
    linode = {
      source  = "linode/linode"
      version = "2.0.0"
    }
  }
}

provider "linode" {
  token = "HERE GOES YOUR PRIVATE TOKEN"
}

resource "linode_instance" "terraform-web" {
  image           = "linode/ubuntu18.04"
  label           = "Terraform-Web-Example"
  group           = "Terraform"
  region          = "us-east"
  type            = "g6-standard-1"
  authorized_keys = ["HERE GOES YOUR SSH PUBLIC KEY"]
  root_pass       = "relativamentejusto"
}
```

```
resource "linode_instance" "terraform-db" {
  image = "linode/centos7"
  label = "Terraform-Db-Example"
  group = "Terraform"
  region = "us-south"
  type = "g6-standard-1"
  swap_size = 1024
  authorized_keys = ["HERE GOES YOUR SSH PUBLIC KEY"]
  root_pass       = "relativamentejusto"
}
```

After writing the terraform file we need to execute the following commands to initialize terraform in the current folder and install the necessary plugins, format and validate our terraform file, then terraform plans the actions that need to be performed in order to achieve the desired configuration state, after that we apply that actions. The last two commands plan the destruction of all the defined resources and execute that actions.

```
terraform init
terraform format
terraform validate
```

```
terraform plan
terraform apply
terraform plan -destroy
terraform destroy
```



More sofisticated deploy with variables and a database

First we define the cariables in a file like this: variables.tf

```
variable "token" {}
variable "authorized_keys" {}
variable "root_pass" {}
variable "region" {
  default = "us-southeast"
}
```

After declaring our variables we need to give them values in a special file called terraform.tfvars, the content should look like this:
terraform.tfvars:

```
authorized_keys = "HERE GOES YOUR SSH PUBLIC KEY"
root_pass       = "relativamentejusto"
token           = "HERE GOES YOUR PRIVATE TOKEN"
```

As done before now we define the resources and the provider:
linode-template.tf:

```
terraform {
  required_providers {
    linode = {
      source  = "linode/linode"
      version = "2.0.0"
    }
  }
}
provider "linode" {
  token = var.token
}
resource "linode_instance" "terraform-web" {
  image           = "linode/centos7"
  label           = "Terraform-Web-Example"
  group           = "Terraform"
  region          = var.region
  type            = "g6-standard-1"
  swap_size       = 1024
  authorized_keys = [var.authorized_keys]
  root_pass       = var.root_pass
}
resource "linode_instance" "terraform-db" {
  image           = "linode/ubuntu18.04"
  label           = "Terraform-Db-Example"
  group           = "Terraform"
  region          = var.region
  type            = "g6-standard-1"
  swap_size       = 1024
  authorized_keys = [var.authorized_keys]
  root_pass       = var.root_pass
}
```

To run this new configuration we should execute all the commands as shown before.

Deploying of a K8s cluster

Now with the knowledge of the basic terraform concepts, we can start playing with deploying a K8s cluster.

To speed up a little bit the configuration this time we'll use a straightforward tool that is designed to deploy in a fast way a K8s cluster to linode. This is the tool https://github.com/codingforentrepreneurs/terraforming-kubernetes-rapid.

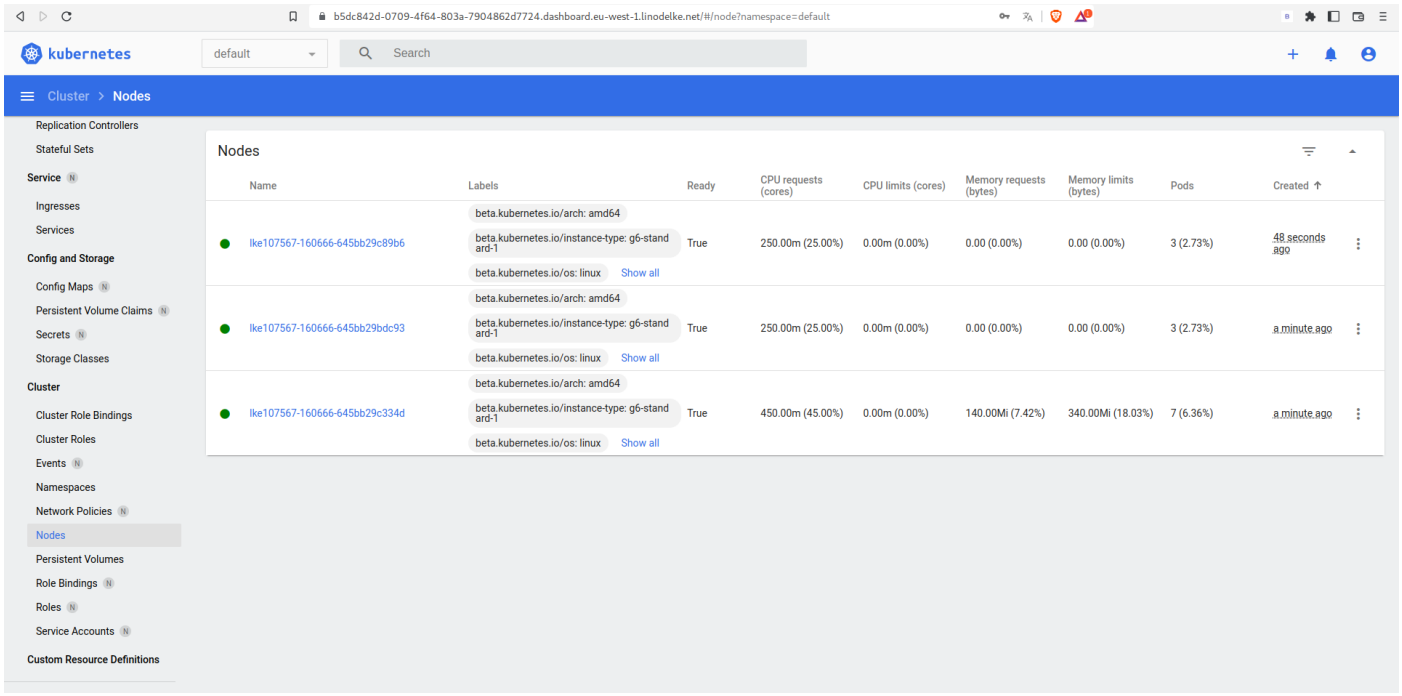
The terraform file looks like this:

```
terraform {
    required_version = ">= 0.15"
    required_providers {
        linode = {
            source = "linode/linode"
        }
    }
}

provider "linode" {
    token = var.linode_api_token
}

locals {
    root_dir = "${abspath(path.root)}"
    k8s_config_dir = "${local.root_dir}/.kube/"
    k8s_config_file = "${local.root_dir}/.kube/kubeconfig.yaml"
}

variable "linode_api_token" {
    description = "Your Linode API Personal Access Token. (required)"
    sensitive   = true
}

resource "linode_lke_cluster" "terraform_k8s" {
    k8s_version="1.26"
    label="terraform-k8s"
    region="eu-west"
    tags=["terraform-k8s"]
    pool {
        type  = "g6-standard-1"
        count = 3

    }
}

resource "local_file" "k8s_config" {
    content = "${nonsensitive(base64decode(linode_lke_cluster.terraform_k8s.kubeconfig))}"
    filename = "${local.k8s_config_file}"
    file_permission = "0600"
}
```

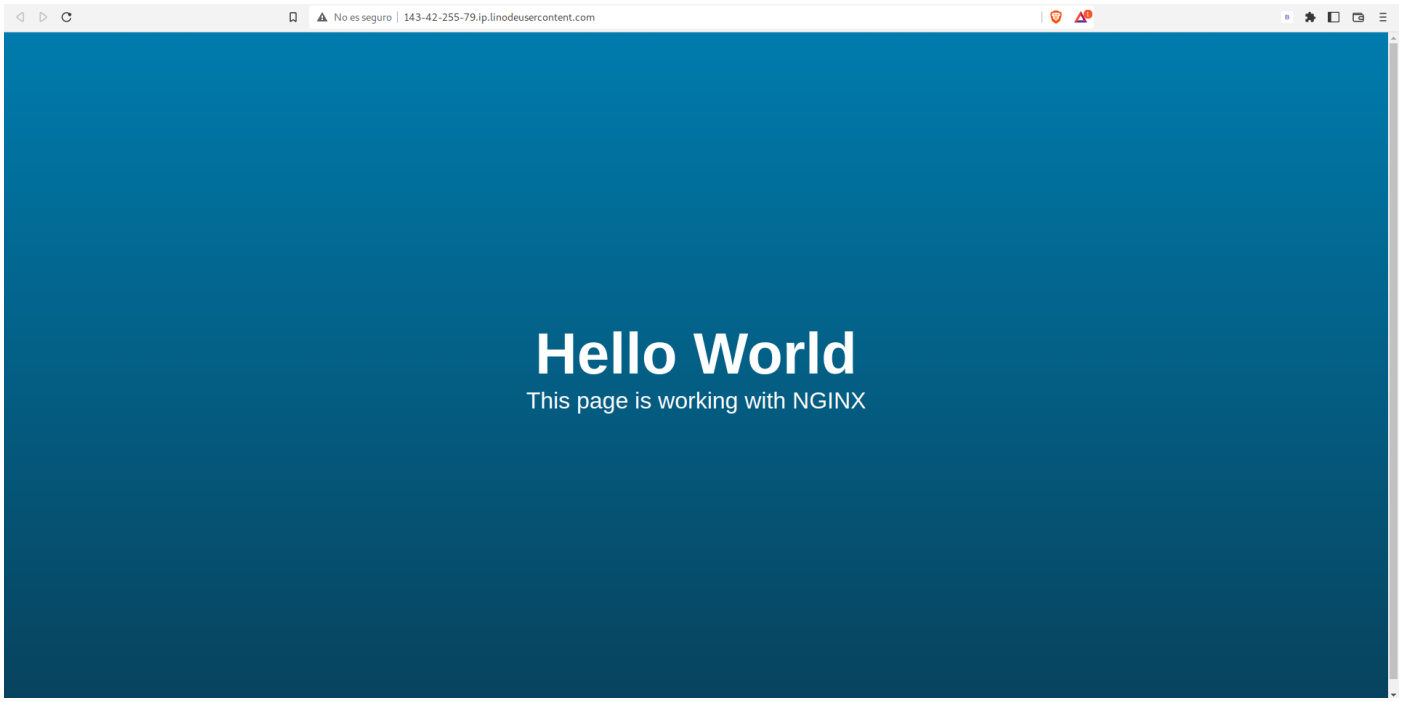
The k8s.yaml contents are this ones:

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cfe-nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: cfe-nginx-deployment
  template:
    metadata:
      labels:
        app: cfe-nginx-deployment
    spec:
      containers:
        - name: cfe-nginx-container
          image: codingforentrepreneurs/cfe-nginx:latest
          imagePullPolicy: Always
          ports:
            - name: cfe-nginx-port
              containerPort: 80

---
apiVersion: v1
kind: Service
metadata:
  name: cfe-nginx-service
spec:
  selector:
    app: cfe-nginx-deployment
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
      targetPort: cfe-nginx-port
```

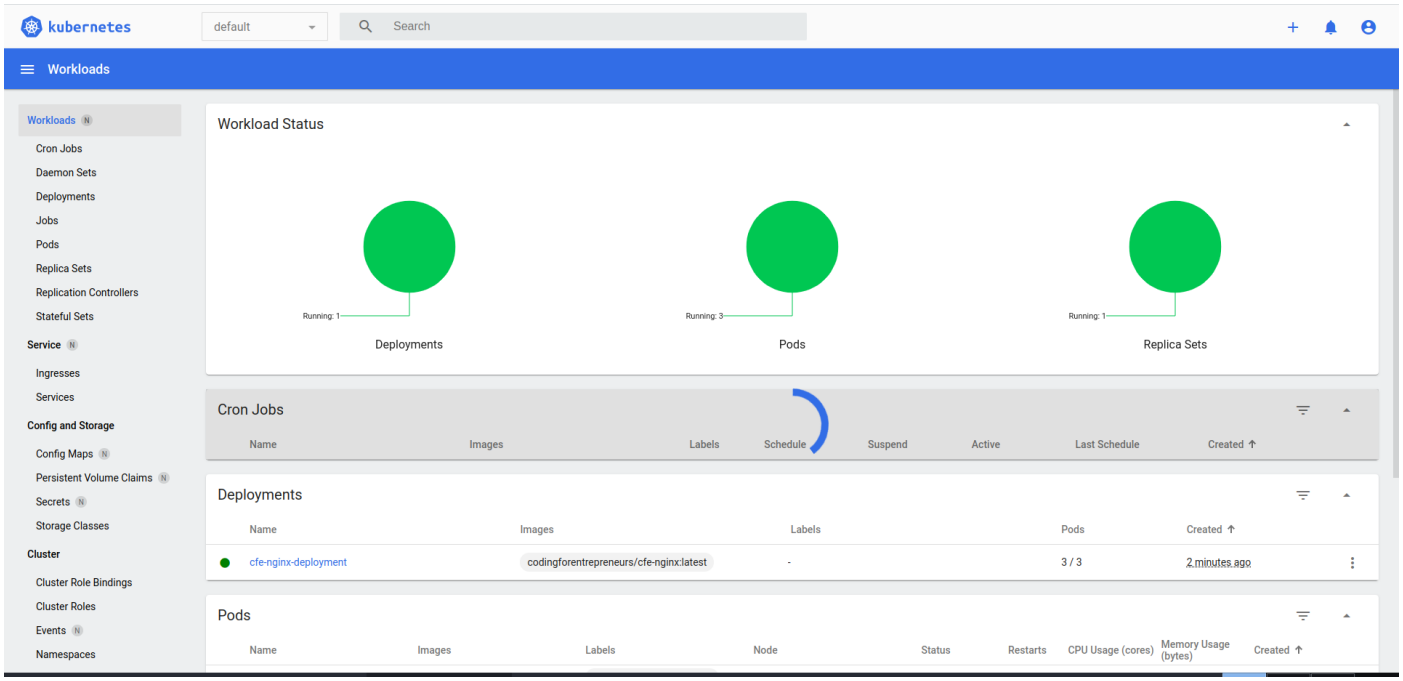
The expected output is something like this, first we have the machines running in linode:



We also can access the website that is running inside the cluster:



The cluster have an already configured dashboard to monitor all the things inside it:



Annex I Upgrading Ruby && Installing podman

To be able to install the libvirt plugin of Vagrant we had a problem with the ruby version in the server, to fix it we need to execute the following commands:

```
# Download and upload ruby zip to the server special folder

unzip ruby-2.7.3.zip
cd ruby-2.7.3/
./configure --prefix=/misc/alumnos/as2/as22022/a821259/bin/ruby
make
make install
ruby -v
```

After running all the commands we add the new PATH into our .bashrc

All the commands are important because we have 2 main problems in the installation process the first and obvious one we dont have root priviledges, the second one is a disk quota limit in all of our directorys except a special one, so we need to install ruby in this special directory.

To setup Podman we download the binaries into the server and after that we execute the following command:

```
tar -xvzf podman-remote-static-linux_amd64.tar.gz
# move it to a folder and add it to PATH
```

Annex II Modifications on the previous setup

In the provision.sh script we enable the default load balancer(servicelb).
We also change the RAM of all the VMs to 1800M, the distributed storage system that we are deploying needs to have support under it so we also need to add a new disk to the workers VMs that will do this task.

Annex III Modified Vagrantfile to use puppet

```
U = 'ubuntu/bionic64'
MASTER = '192.168.0.49'
NODES = [
  { hostname: 'm', type: "master", ip: MASTER, mem: 1800, m: MASTER },
  { hostname: 'w1', type: "worker", ip: '192.168.0.41', mem: 1800, m: MASTER },
  { hostname: 'w2', type: "worker", ip: '192.168.0.42', mem: 1800, m: MASTER },
  { hostname: 'w3', type: "worker", ip: '192.168.0.43', mem: 1800, m: MASTER },
]

Vagrant.configure("2") do |config|
    NODES.each do |node|
        config.vm.define node[:hostname] do |nodeconfig|
            nodeconfig.vm.box = U
            nodeconfig.vm.hostname = node[:hostname]
            # Additional Network
            nodeconfig.vm.network   :public_network,
                                    bridge: "eth0",
                                    ip: node[:ip],
                                    nic_type: "virtio"
            # Virtual hardware configuration
            nodeconfig.vm.provider :virtualbox do |v|
                v.memory = node[:mem]
                v.cpus = 1
                v.default_nic_type = "virtio"
                v.customize ["modifyvm", :id, "--name", node[:hostname]]
                if node[:type] == "worker"
                  v.customize [ "createmedium",
                  "--filename", "disk-#{node[:hostname]}.vdi",
                  "--size", 30*1024 ] # Que tamaÃ±o es este ??
                  v.customize [ "storageattach", :id,
                  "--storagectl", "SCSI",
                  "--port", 2, "--device", 0,"--type", "hdd",
                  "--medium", "disk-#{node[:hostname]}.vdi" ]
                end
            end
            nodeconfig.vm.boot_timeout = 600
            nodeconfig.vm.provision "shell", path: "./puppet.sh"
            nodeconfig.vm.provision "puppet" do |puppet|
              puppet.options = "--verbose --logdest /var/log/puppet.log"
              puppet.facter         = {
                "hostname"  => node[:hostname],
                "nodeip"    => node[:ip],
                "masterip"  => node[:m],
                "nodetype"  => node[:type]
              }
              puppet.manifests_path = "./manifests/"
              puppet.manifest_file = "vagrant_vm.pp"
            end

            if node[:type] == "master"
              nodeconfig.trigger.after :up do |trigger|
                trigger.run = \
                        {inline: "sh -c 'cp k3s.yaml /home/fjpizarro/.kube/config'"}
              end
            end
        end
    end
end
```

The script that installs puppet is this:

```
#!/bin/sh
command -v puppet > /dev/null && { echo "Puppet is installed! skipping" ; exit 0; }

ID=$(cat /etc/os-release | awk -F= '/^ID=/{print $2}' | tr -d '"')
VERS=$(cat /etc/os-release | awk -F= '/^VERSION_ID=/{print $2}' | tr -d '"')

case "${ID}" in
  centos|rhel)
    wget https://yum.puppet.com/puppet5/puppet5-release-el-${VERS}.noarch.rpm
    rpm -Uvh puppet5-release-el-${VERS}.noarch.rpm
    yum install -y puppet-agent
    ;;
  fedora)
    rpm -Uvh https://yum.puppet.com/puppet5/puppet5-release-fedora-${VERS}.noarch.rpm
    yum install -y puppet-agent
    ;;
  debian|ubuntu)
    wget https://apt.puppetlabs.com/puppet5-release-$(lsb_release -cs).deb
    dpkg -i puppet5-release-$(lsb_release -cs).deb
    apt-get -qq update
    apt-get install -y puppet-agent
    ;;
  *)
    echo "Distro '${ID}' not supported" 2>&1
    exit 1
    ;;
esac
```

Annex IV Setup libvirt

We start by doing a security backup of the previous vagrant configuration and envoiroment, after doing this we can start by installing the libvirt plugin that Vagrant needs in order to interact with libvirt, this can be done by running:
In the lab envoiroment the ruby version must be upgraded without root privileges this process can be found in annex 1
`vagrant plugin install vagrant-libvirt`

```
virsh pool-define-as a821259remote dir - - - - $(pwd)
virsh pool-list --all
virsh pool-build a821259remote
virsh pool-start a821259remote
virsh pool-autostart a821259remote
virsh pool-info a821259remote
```

The modifications on the Vagrantfile were the following, we just need to substitute the part of the provider by this:

```
          config.vm.provider :libvirt do |lib|
            lib.uri = "qemu+ssh://a821259@155.210.154.206/system"
            lib.username = "a821259"
            lib.memory = node[:mem]
            lib.nic_model_type = "virtio"
            lib.driver = "kvm"
            lib.cpus = 1
            lib.keymap = 'es'
            #lib.storage_pool :default do |pool|
            #  pool.path = "/misc/alumnos/as2/as22022/a821259/remote/"
            #end
            lib.storage_pool_name = "a821259remote"
          end

          nodeconfig.vm.synced_folder ".", "/vagrant", type: "rsync"
```

And the network part by this:

```
                              nodeconfig.vm.network   :public_network,
                                # substituir por vuestra interfaz de red
                                #bridge: "eth0",
                                ip: node[:ip],
                                nic_type: "virtio"
                                :dev => "br1",
                                :mode => "bridge",
                                :type => "bridge"
```

Annex V Problems found along the project

Throughout the project, many different types of problems have been encountered here are the most relevant ones.

This first problem was a really big headache, because at the start we didnt have any clue of what could be the possible cause. After a lot of time checking Vagrant and Virtual box versions, difference between manifests and other kind of things, we finally found the reason. It was a difference in the K3s version and also in the kubectl client.

```
kubectl create -f common.yaml
namespace/rook-ceph created
serviceaccount/rook-ceph-system created
serviceaccount/rook-ceph-osd created
serviceaccount/rook-ceph-mgr created
serviceaccount/rook-ceph-cmd-reporter created
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
podsecuritypolicy.policy/rook-privileged created
clusterrole.rbac.authorization.k8s.io/psp:rook created
clusterrolebinding.rbac.authorization.k8s.io/rook-ceph-system-psp created
rolebinding.rbac.authorization.k8s.io/rook-ceph-default-psp created
rolebinding.rbac.authorization.k8s.io/rook-ceph-osd-psp created
rolebinding.rbac.authorization.k8s.io/rook-ceph-mgr-psp created
rolebinding.rbac.authorization.k8s.io/rook-ceph-cmd-reporter-psp created
serviceaccount/rook-csi-cephfs-plugin-sa created
serviceaccount/rook-csi-cephfs-provisioner-sa created
role.rbac.authorization.k8s.io/cephfs-external-provisioner-cfg created
rolebinding.rbac.authorization.k8s.io/cephfs-csi-provisioner-role-cfg created
clusterrole.rbac.authorization.k8s.io/cephfs-csi-nodeplugin created
clusterrole.rbac.authorization.k8s.io/cephfs-csi-nodeplugin-rules created
clusterrole.rbac.authorization.k8s.io/cephfs-external-provisioner-runner created
clusterrole.rbac.authorization.k8s.io/cephfs-external-provisioner-runner-rules created
clusterrolebinding.rbac.authorization.k8s.io/rook-csi-cephfs-plugin-sa-psp created
clusterrolebinding.rbac.authorization.k8s.io/rook-csi-cephfs-provisioner-sa-psp created
clusterrolebinding.rbac.authorization.k8s.io/cephfs-csi-nodeplugin created
clusterrolebinding.rbac.authorization.k8s.io/cephfs-csi-provisioner-role created
serviceaccount/rook-csi-rbd-plugin-sa created
serviceaccount/rook-csi-rbd-provisioner-sa created
role.rbac.authorization.k8s.io/rbd-external-provisioner-cfg created
rolebinding.rbac.authorization.k8s.io/rbd-csi-provisioner-role-cfg created
clusterrole.rbac.authorization.k8s.io/rbd-csi-nodeplugin created
clusterrole.rbac.authorization.k8s.io/rbd-csi-nodeplugin-rules created
clusterrole.rbac.authorization.k8s.io/rbd-external-provisioner-runner created
clusterrole.rbac.authorization.k8s.io/rbd-external-provisioner-runner-rules created
clusterrolebinding.rbac.authorization.k8s.io/rook-csi-rbd-plugin-sa-psp created
clusterrolebinding.rbac.authorization.k8s.io/rook-csi-rbd-provisioner-sa-psp created
clusterrolebinding.rbac.authorization.k8s.io/rbd-csi-nodeplugin created
clusterrolebinding.rbac.authorization.k8s.io/rbd-csi-provisioner-role created
resource mapping not found for name: "cephclusters.ceph.rook.io" namespace: "" from "common.yaml": no matches for
kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "cephclients.ceph.rook.io" namespace: "" from "common.yaml": no matches for
kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "cephfilesystems.ceph.rook.io" namespace: "" from "common.yaml": no matches
for kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "cephnfses.ceph.rook.io" namespace: "" from "common.yaml": no matches for
kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "cephobjectstores.ceph.rook.io" namespace: "" from "common.yaml": no matches
for kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "cephobjectstoreusers.ceph.rook.io" namespace: "" from "common.yaml": no
matches for kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "cephblockpools.ceph.rook.io" namespace: "" from "common.yaml": no matches
```

```
for kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "volumes.rook.io" namespace: "" from "common.yaml": no matches for kind
"CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "objectbuckets.objectbucket.io" namespace: "" from "common.yaml": no matches
for kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "objectbucketclaims.objectbucket.io" namespace: "" from "common.yaml": no
matches for kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-object-bucket" namespace: "" from "common.yaml": no matches for
kind "ClusterRoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-cluster-mgmt" namespace: "" from "common.yaml": no matches for
kind "ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-cluster-mgmt-rules" namespace: "" from "common.yaml": no matches
for kind "ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-system" namespace: "rook-ceph" from "common.yaml": no matches for
kind "Role" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-global" namespace: "" from "common.yaml": no matches for kind
"ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-global-rules" namespace: "" from "common.yaml": no matches for
kind "ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-mgr-cluster" namespace: "" from "common.yaml": no matches for kind
"ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-mgr-cluster-rules" namespace: "" from "common.yaml": no matches
for kind "ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-object-bucket" namespace: "" from "common.yaml": no matches for
kind "ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-system" namespace: "rook-ceph" from "common.yaml": no matches for
kind "RoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-global" namespace: "" from "common.yaml": no matches for kind
"ClusterRoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-osd" namespace: "rook-ceph" from "common.yaml": no matches for
kind "Role" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-osd" namespace: "" from "common.yaml": no matches for kind
"ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-mgr-system" namespace: "" from "common.yaml": no matches for kind
"ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-mgr-system-rules" namespace: "" from "common.yaml": no matches for
kind "ClusterRole" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-mgr" namespace: "rook-ceph" from "common.yaml": no matches for
kind "Role" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-cmd-reporter" namespace: "rook-ceph" from "common.yaml": no
matches for kind "Role" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-cluster-mgmt" namespace: "rook-ceph" from "common.yaml": no
matches for kind "RoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-osd" namespace: "rook-ceph" from "common.yaml": no matches for
```

```
kind "RoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-mgr" namespace: "rook-ceph" from "common.yaml": no matches for
kind "RoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-mgr-system" namespace: "rook-ceph" from "common.yaml": no matches
for kind "RoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-mgr-cluster" namespace: "" from "common.yaml": no matches for kind
"ClusterRoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-osd" namespace: "" from "common.yaml": no matches for kind
"ClusterRoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
resource mapping not found for name: "rook-ceph-cmd-reporter" namespace: "rook-ceph" from "common.yaml": no
matches for kind "RoleBinding" in version "rbac.authorization.k8s.io/v1beta1"
ensure CRDs are installed first
```

While creating the direct mount container in the 2º part of the project a small error appeared, this was just a error inside the yaml manifest that we were using and the fix was just changing one line of the manifest.

```
kubectl -n rook-ceph get events
  # 53s         Normal    Pulled              pod/rook-direct-mount-95b6b4d88-zr75q
Container image "rook/ceph:master" already present on machine
  # 53s         Normal    Created             pod/rook-direct-mount-95b6b4d88-zr75q
Created container rook-direct-mount
  # 53s         Warning   Failed              pod/rook-direct-mount-95b6b4d88-zr75q
Error: failed to create containerd task: OCI runtime create failed: container_linux.go:370: starting container
process caused: exec: "/tini": stat /tini: no such file or directory: unknown
```

In the laboratory which is a pretty special envoiroments because of some limitations like not having sudo privileges or a really small max quota disk, an error ocurred while trying to install the necessary plugin to setup Vagrant with libvirt, this was because the version of ruby. The fix to the problem is described in the Annex I.

```
Vagrant failed to properly resolve required dependencies. These
errors can commonly be caused by misconfigured plugin installations
or transient network issues. The reported error is:

nokogiri requires Ruby version >= 2.7, < 3.3.dev. The current ruby version is 2.6.6.146.
```

They were also a few problem while trying to use Libvirt on remote, specificly relative to the folders that were being used to store the data.

The last and most ridiculous problem was while setting up puppet as a provider, the problem was the following, all the Vagrantfile was OK, the VMs already have puppet agent installed on them, but for no apparent reason the puppet provider didnt activate even if the puppet configuration was OK. The solution was as simple as the problem hard to see, in the puppet manifest a class was defined vm_vagrant, but it was never called or instantiated by any node.