****

**《搭建基于KVM的云服务系统》**

**云计算课程报告**

小组成员：

李安南（20211000722）、罗成勇（20211003196）

戚正（20211000729）、肖江伟（20211001866）

班级：111211

指导教师：陈小岛

成 绩：

**中国地质大学计算机学院**

**2023年 12 月**

**目录**

**一、项目背景……………………………………………………………2**

**二、需求分析……………………………………………………………2**

**三、选型设计……………………………………………………………2**

**四、实验步骤……………………………………………………………3**

1. **预期结果……………………………………………………………3**
2. **主要实现……………………………………………………………4**

前端业务实现……………………………………………………………4

后端功能实现……………………………………………………………7

**七、系统演示……………………………………………………………15**

**八、源码附录……………………………………………………………21**

**一、项目背景**

随着云计算和虚拟化技术的迅速发展，企业和个人对虚拟化平台的需求日益增加。KVM（Kernel-based Virtual Machine）作为一种开源的虚拟化解决方案，提供了性能良好、可扩展性强的虚拟化环境，成为企业部署云计算、构建虚拟化基础设施的重要选择。然而，虚拟机的创建、配置和管理对于一般用户来说仍然具有一定的技术门槛，因此需要一个友好、直观的交互界面来简化这些操作。

主要研究内容：

图形用户界面设计与交互体验：利用现代前端技术，如Vue.js，设计直观、美观、用户友好的虚拟机管理界面。

KVM与Libvirt的深度集成：研究如何充分利用KVM和Libvirt提供的接口，实现系统的高效性能和稳定性。

错误处理与报警机制设计：设计灵活的错误处理机制，通过前端报警、日志记录等方式提供详细的错误信息和解决方案。

项目可行性分析：

市场需求：随着云计算和虚拟化技术的发展，企业和个人对于易用的虚拟化管理工具的需求不断增加。

技术可行性：KVM作为一种成熟的虚拟化技术，具备较高的性能和稳定性，且有丰富的社区支持。Vue、Flask等前端开发框架为实现友好用户界面提供了丰富有效的工具。

资源可行性：KVM和相关技术的文档和社区资源丰富，且开源软件的使用可以大大降低开发成本。

**二、需求分析**

1. 用户界面需求：

界面设计应直观、美观，使用户能够轻松理解和使用系统功能，提供良好的用户交互体验，包括简洁的导航、明了的操作流程，以减少用户学习成本。

2. 功能需求：

虚拟机生命周期管理：提供创建、删除、启动、停止、暂停、恢复等基本操作，确保用户能够方便地管理虚拟机。

配置选项：允许用户配置虚拟机的参数，包括虚拟硬件资源分配和网络配置。

统计报告与监控：提供虚拟机资源使用情况的统计报告，实现对虚拟机的实时监控，以便用户了解虚拟机的运行状态。

3. 错误处理与报警机制：

灵活的错误处理：设计系统的错误处理机制，通过前端报警、日志记录等方式提供详细的错误信息和解决方案。

报警机制：实现硬件资源或其他问题发生时的及时报警机制，确保用户能够及时获得系统状态的反馈。

**三、选型设计**

前端框架：Vue.js

Vue.js是一款轻量级、高性能的前端框架，适合快速构建交互式用户界面，提供了响应式数据绑定和组件化开发的特性，可以构建复杂的用户界面。

前端构建工具：Webpack

Webpack用于打包和构建前端代码，可以将Vue.js的单文件组件、样式表等进行打包，提高前端代码的加载性能。

前端路由：Vue Router

Vue Router用于在Vue.js应用中实现前端路由，支持通过URL进行页面之间的切换,提供了SPA（单页应用）的开发支持，提升用户体验。

组件库：Element UI

Element UI是基于Vue.js的一套组件库，具有良好的视觉设计和响应式布局，适合构建美观的用户界面，提供丰富的UI组件，包括表格、表单、对话框等，可以加速前端界面的开发。

后端开发框架：Flask

Flask是一个轻量级的Python后端框架，适合快速搭建RESTful API。

可以用于处理前端与KVM、Libvirt之间的数据交互，提供后端逻辑支持。

虚拟化平台：KVM和Libvirt

KVM作为主要的虚拟化技术，提供了高性能的虚拟化环境。

Libvirt作为管理工具，提供了API和工具，方便对虚拟机进行管理。

**四、实验步骤**

1. 用户界面设计

设计直观、美观的图形用户界面，包括虚拟机列表、创建虚拟机页面、虚拟机配置页面等。实现用户界面的交互和动态效果。

1. KVM和Libvirt集成

通过Libvirt提供的API，实现与KVM的深度集成，包括虚拟机的创建、启动、停止、暂停、恢复等操作。

1. 虚拟机生命周期管理

通过前端界面设计创建虚拟机的页面，可以选择虚拟机的配置参数、启动和停止虚拟机以及暂停和恢复虚拟机。

1. 错误处理与报警机制

设计系统的错误处理机制，通过前端报警、日志记录等方式提供详细的错误信息和解决方案。网络或其他问题发生时报警机制及时启动，确保用户及时获得系统状态的反馈。

1. 测试与优化

进行系统功能测试，确保用户界面和基础功能正常运行；进行性能优化，确保系统在虚拟机管理过程中能够保持高效。

**五、预期结果**

1.实现直观、易用的虚拟机管理界面，拥有布局优美，功能醒目的交互设计，提供友好的交互体验，减少用户操作的技术门槛。

2.支持虚拟机的创建、删除、启动、停止、暂停、恢复等基本操作，使用户能够方便简易地管理虚拟机的生命周期。

3.能提供清晰的错误信息和解决方案的错误处理机制；实现前端报警机制，及时向用户反馈系统状态和可能的问题。

**六、主要实现**

**前端业务实现：**

**1.从后端获取主机信息：**

描述: 此方法向后端的 /api/hostInfo 发送GET请求以获取主机信息。获取成功后，存储响应数据；如果失败，则显示错误信息。

**具体代码：**

fetchHostInfo() {

axios.get('http://localhost:5000/api/hostInfo') // 假设这是获取主机信息的后端URL

.then(response => {

if (response.data) {

this.hostInfo = response.data; // 存储主机信息

} else {

this.showAlert('无法获取主机信息', 'error');

}

})

.catch(error => {

this.showAlert('请求错误: ' + error.message, 'error');

});

},

**2.从后端获取虚拟机信息：**

描述: 此方法向后端的 /api/getInfo 发送GET请求以获取所有虚拟机的信息。成功后，更新虚拟机数据；如果失败，则显示错误信息。

**具体代码：**

get\_reportData(showAlertOnSuccess = false) {

axios.get('http://localhost:5000/api/getInfo')

.then(response => {

console.log(response.data);

if (response.data && response.data.status === 'success') {

// 如果成功，清空原有数据并添加新数据

this.reportData = response.data.VM.map(vm => ({

name: vm.name,

KVM\_status: vm.KVM\_status,

cpu: vm.cpu,

memory: vm.memory,

disk: vm.disk,

CPU\_usage: vm.CPU\_usage

}));

if (showAlertOnSuccess) {

this.showAlert(response.data.message, 'success');

}

} else { // 如果后端返回失败

this.showAlert('虚拟机信息获取失败', 'error');

}

})

.catch(error => { // 网络或服务器错误

this.showAlert('请求错误: ' + error.message, 'error');

});

},

**3.显示用户界面警告：**

描述: 此方法用于在用户界面上显示警告信息。它创建一个新的警告对象，并在一定时间后将其从警告列表中移除。

**具体代码：**

showAlert(msg, type) {

const alert = {

id: Date.now(), // 生成唯一ID

msg: msg,

type: type,

};

this.alerts.push(alert);

setTimeout(() => {

this.alerts = this.alerts.filter(a => a.id !== alert.id);

}, 6000); // 5秒后移除提示

},

**4.创建/删除/启动/关闭/暂停/恢复虚拟机：**

主要功能: 向后端发送创建/删除/启动/关闭/暂停/恢复虚拟机的请求

描述: 在满足限制条件下，向后端的 /api/vm 发送POST请求来创建新的虚拟机，创建成功或失败都会显示相应的信息。在所有选中的虚拟机都已关机的情况下，执行删除操作，如果有虚拟机未关机，则显示错误信息。启动/关闭/暂停/恢复虚拟机使用相同的通用函数逻辑，向后端发送对应请求。

**具体代码：**

createVM() {

// 检查虚拟机名称

if (!this.newVM.name) {

this.showAlert('请输入虚拟机名称', 'error');

return;

}

// 检查虚拟机名称是否包含空格

if (this.newVM.name.includes(' ')) {

this.showAlert('虚拟机名称不能包含空格', 'error');

return;

}

const vmExists = this.reportData.some(vm => vm.name === this.newVM.name);

if (vmExists) {

this.showAlert('虚拟机名称已存在，请选择一个不同的名称', 'error');

return;

}

const vmData = {

operation: "create",

name: this.newVM.name,

cpu: this.newVM.cpu,

memory: this.newVM.memory,

disk: this.newVM.disk

};

axios.post(`http://localhost:5000/api/vm`, vmData)

.then(response => {

if (response.data && response.data.status === 'success') {

this.showAlert(response.data.message, 'success');

this.get\_reportData(); // 调用 get\_reportData 来刷新数据

} else {

this.showAlert(response.data.message, 'error');

}

})

.catch(error => {

this.showAlert('请求错误: ' + error.message, 'error');

})

.finally(() => {

this.newVM.name = '';

this.showCreateVMDialog = false;

});

},

deleteVM() {

// 检查所有选中的虚拟机是否都已关机

const allVmsShutdown = this.selectedVMs.every(vm => vm.KVM\_status === "已关机");

if (!allVmsShutdown) {

// 如果有虚拟机的状态不是已关机，则显示错误消息并返回

this.showAlert('请先关闭选中的所有虚拟机再进行删除操作', 'error');

return;

}

// 所有虚拟机都已关机，可以执行删除操作

this.performVMOperation('delete');

},

startVM() {

this.performVMOperation('start');

},

shutdownVM() {

this.performVMOperation('shutdown');

},

suspendVM() {

this.performVMOperation('suspend');

},

resumeVM() {

this.performVMOperation('resume');

},

performVMOperation(endpoint) {

if (this.selectedVMs.length > 0) {

this.selectedVMs.forEach(vm => {

const vmData = {

operation: endpoint,

name: vm.name,

};

// 执行相应的操作

axios.post(`http://localhost:5000/api/vm`, vmData)

.then(response => {

if (response.data && response.data.status === 'success') {

this.showAlert(response.data.message, 'success');

} else {

this.showAlert(response.data.message, 'error');

}

})

.catch(error => {

this.showAlert('请求错误: ' + error.message, 'error');

})

.finally(() => {

this.get\_reportData(); // 在所有请求完成后刷新数据

})

});

} else {

this.showAlert('没有选中的虚拟机', 'error');

}

},

**后端功能实现：**

**1.后端api接口包装：**

使用flask框架架构起了后端服务器，导入了cors以实现flask后端与vue前端的通讯。

描述：后端向前端开放了三个端口：/api/vm、/api/getInfo和/api/hostInfo。

vm端口接收前端的post信息，预处理前端的信息，执行KVM管理操作并返回操作结果。

getInfo端口接受前端的GET请求，将全部的虚拟机状态信息返回给前端。

hostInfo端口接受前端的GET请求，将主机信息返回给前端。

**具体代码：**

from flask import Flask, request, jsonify

from flask\_cors import CORS # 引入 CORS

import main

app = Flask(\_\_name\_\_)

CORS(app) # 应用 CORS 到您的 Flask 应用

@app.route('/api/vm', methods=['POST'])

def vm():

# 获取前端传递的虚拟机信息

data = request.get\_json()

operation = data["operation"]

# 执行KVM管理操作

data = main.operation\_bus(operation, data)

return jsonify(data)

@app.route('/api/getInfo', methods=['GET'])

def get\_info():

# 在这里执行获取KVM信息

data = main.get\_all\_vm\_info()

return jsonify(data)

@app.route('/api/hostInfo', methods=['GET'])

def get\_host\_info():

# 在这里执行获取主机信息

data = main.get\_host\_info()

return jsonify(data)

**2.虚拟机操作总线：**

参数：操作类型，操作参数。

返回值：符合格式的JSON后端响应。

描述：接收来自前端的操作相关参数以执行相应的对虚拟机的操作，并返回符合格式的操作执行结果。该总线的操作为对虚拟机的操作，将所有的对虚拟机的操作（创建，删除，启动，暂停，恢复，关闭）集合到本总线中，简化并统一了后端的端口。

**具体代码：**

def operation\_bus(operation, arg):

"""

接受命令和参数表的命令执行总线

:param operation: 从json解析出的操作 str

:param arg: 从json解析出的含相关参数字典 dict[str:object]

:return: 符合后端样例的操作结果

"""

conn, message = vmop1.get\_conn()

# 后端发回

data = {}

# 执行操作

if conn:

if operation == "create":

message += vmop3.vm\_create\_with\_check(conn, arg["name"], arg["memory"], arg["cpu"], arg["disk"])

data["name"] = arg["name"]

data["cpu"] = arg["cpu"]

data["memory"] = arg["memory"]

data["disk"] = arg["disk"]

elif operation == "delete":

message += vmop3.vm\_delete(conn, arg["name"])

elif operation == "start":

message += vmop2.vm\_start(conn, arg["name"])

elif operation == "suspend":

message += vmop2.vm\_suspend(conn, arg["name"])

elif operation == "resume":

message += vmop2.vm\_resume(conn, arg["name"])

elif operation == "shutdown":

message += vmop2.vm\_destroy(conn, arg["name"])

else:

message += "未知命令 "

message += vmop1.close\_conn(conn)

data["message"] = message

data["status"] = message\_check(message)

return data

**1）启动：**

def vm\_start(conn, name):

try:

domain = conn.lookupByName(name)

domain.create()

message = f"{name} 启动成功 "

except Exception as e:

message = f"{name} 启动失败 "

print(e)

return message

**2）暂停：**

def vm\_suspend(conn, name):

try:

domain = conn.lookupByName(name)

domain.suspend()

message = f"{name} 暂停成功 "

except Exception as e:

message = f"{name} 暂停失败 "

print(e)

return message

**3）恢复：**

def vm\_resume(conn, name):

try:

domain = conn.lookupByName(name)

domain.resume()

message = f"{name} 重启成功 "

except Exception as e:

message = f"{name} 重启失败 "

print(e)

return message

**4）关闭：**

def vm\_destroy(conn, name):

try:

domain = conn.lookupByName(name)

domain.destroy()

message = f"{name} 关闭成功 "

except Exception as e:

message = f"{name} 关闭失败 "

print(e)

return message

**5）创建：**

def vm\_create(conn, name, memory, cpu, disk):

# 创建磁盘

message, fault = create\_img(conn, name, disk)

try:

# 检查磁盘

if fault:

raise ValueError("disk not found")

# 创建虚拟机

xml\_desc = \

f'''

略

'''

conn.defineXML(xml\_desc)

message += "创建虚拟机成功 "

except Exception as e:

message = '创建虚拟机失败 '

print(e)

return message

def create\_img(conn, name, size):

try:

pool = conn.storagePoolLookupByName('default')

vol\_xml = f"""

<volume type='file'>

<name>{name}.qcow2</name>

<capacity unit='G'>{size}</capacity>

<allocation unit='G'>{size}</allocation>

<target>

<format type='qcow2'/>

</target>

</volume>

"""

vol = pool.createXML(vol\_xml, 0)

message = '磁盘创建成功 '

fault = False

except Exception as e:

message = '磁盘创建失败 '

fault = True

print(e)

return message, fault

**6）删除：**

def vm\_delete(conn, name):

dom = conn.lookupByName(name)

message, fault = delete\_img(conn, name)

try:

if fault:

raise ValueError("disk not found")

dom.undefine()

message += f"虚拟机{name} 删除成功 "

except Exception as e:

message = f"虚拟机{name} 删除失败 "

print(e)

return message

def delete\_img(conn, name):

try:

pool = conn.storagePoolLookupByName("default")

vol = pool.storageVolLookupByName(f"{name}.qcow2")

vol.delete(0)

message = "磁盘删除成功 "

fault = False

except Exception as e:

message = '磁盘删除失败 '

fault = True

print(e)

return message, fault

**3.获取所有虚拟机信息函数：**

参数：无

返回值：符合格式的含全部虚拟机相关状态的数据

描述：本函数将调用相关函数，该函数会遍历所有虚拟机并获取相关状态且封装为指定格式。该函数也做出了报错处理，在发生错误时，向前端发送空的虚拟机状态。

**具体代码：**

def get\_all\_vm\_info():

conn, message = vmop1.get\_conn()

# 后端发回

data = {}

# 执行操作

try:

VM = vmop1.show\_vm(conn)

message += "成功获取全部虚拟机 "

except Exception as e:

VM = []

message += "获取全部虚拟机失败 "

print(e)

message += vmop1.close\_conn(conn)

data["message"] = message

data["status"] = message\_check(message)

data["VM"] = VM

return data

**1）获取所有虚拟机的详细信息**

描述: 用于获取和组装所有虚拟机的详细信息，包括名称、状态、CPU使用率、CPU核数、内存和磁盘大小。

**具体代码**

def show\_vm(conn):

vm = []

domains = conn.listAllDomains() # 获取所有虚拟机

for domain in domains:

detail = {}

name = domain.name()

detail["name"] = name

state, \_ = domain.state()

KVM\_status = "未知"

if state == libvirt.VIR\_DOMAIN\_RUNNING:

KVM\_status = "运行中"

CPU\_usage = caclulate\_CPU\_usage(domain)

elif state == libvirt.VIR\_DOMAIN\_PAUSED:

KVM\_status = "已暂停"

CPU\_usage = "0%"

elif state == libvirt.VIR\_DOMAIN\_SHUTOFF:

KVM\_status = "已关机"

CPU\_usage = "0%"

detail["KVM\_status"] = KVM\_status

detail["CPU\_usage"] = CPU\_usage

cpu = domain.info()[3]

detail["cpu"] = cpu

memory = domain.info()[2] // 1024 \*\* 2

detail["memory"] = memory

pool = conn.storagePoolLookupByName("default")

vol = pool.storageVolLookupByName(f"{name}.qcow2")

vol\_info = vol.info()

disk = vol\_info[1] // 1024 \*\* 3

detail["disk"] = disk

vm.append(detail)

return vm

**2）获取虚拟机CPU使用率**

描述: 该函数用于计算虚拟机的CPU使用率。它通过两次获取CPU时间状态，并计算这两个状态之间的差异来估算CPU使用率。

**具体代码**

def caclulate\_CPU\_usage(domain):

# 获取虚拟机CPU使用率（需要两次获取状态间的时间差）

cpu\_time1 = domain.getCPUStats(True)[0]['cpu\_time']

time.sleep(1) # 等待一秒

cpu\_time2 = domain.getCPUStats(True)[0]['cpu\_time']

cpu\_usage = ((cpu\_time2 - cpu\_time1) / 1e9) \* 100 # 将ns转换为秒并计算CPU使用率

return f"{int(cpu\_usage)}%"

**4.获取主机信息函数：**

参数：无

返回值：符合格式的主机信息

描述：函数调用了libvirt库，从连接中获取主机的CPU类型、CPU个数、内存总大小；调用了psutil库，获取主机的空闲内存；调用了os库，获取主机的磁盘总大小。调用check\_remain\_size(conn)函数，获取主机的理论磁盘空间。

**具体代码：**

def get\_host\_info(): # 主机信息

conn, message = vmop1.get\_conn()

# 后端发回

data = {}

# 执行操作

nodeInfo = conn.getInfo() # 获取虚拟化主机信息

data["model"] = str(nodeInfo[0])

data["cpu"] = str(nodeInfo[2])

data["memory"] = f"{nodeInfo[1]//1024} GB"

# 获取内存信息

memory\_info = psutil.virtual\_memory()

# 获取空闲内存大小

mem = memory\_info.available

data["free\_memory"] = f"{mem/1024\*\*3:.1f} GB" # 主机空闲内存

partitions = os.statvfs("/")

total\_size = partitions.f\_frsize \* partitions.f\_blocks

data["disk"] = f"{total\_size / (1024 \*\* 3):.2f} GB"

data["free\_disk"] = f"{vmop1.check\_remain\_size(conn)//1024\*\*3} GB" # 主机理论磁盘空闲

message += vmop1.close\_conn(conn)

return data

**1）检查存储空间**

描述：此函数用于判断请求的磁盘大小是否超过了存储池的剩余空间。如果请求的大小超过剩余空间，则返回True，表示不允许进行此操作。

**具体代码**

def check\_allow\_size(conn, disk):

pool\_total\_remain = check\_remain\_size(conn)

disk\_size = disk \* 1024 \* 1024 \* 1024

if disk\_size > pool\_total\_remain:

return True

else:

return False

**2）计算存储池剩余空间**

描述：该函数计算存储池的剩余空间。它获取存储池的总可用空间，并从中减去所有卷占用的空间。

**具体代码**

def check\_remain\_size(conn):

pool = conn.storagePoolLookupByName("default")

pool\_info = pool.info()

pool\_available = pool\_info[3]

volumes = pool.listVolumes()

# 计算实际占用和总占用大小

volume\_real\_size = 0

volume\_total\_size = 0

for volume in volumes:

vol = pool.storageVolLookupByName(volume)

vol\_info = vol.info()

volume\_real\_size += vol\_info[2]

volume\_total\_size += vol\_info[1]

pool\_total\_remain = pool\_available + volume\_real\_size - volume\_total\_size

return pool\_total\_remain

**七、系统演示**

进入主页面，mounted预先抓取后端信息，目前虚拟机信息为空：



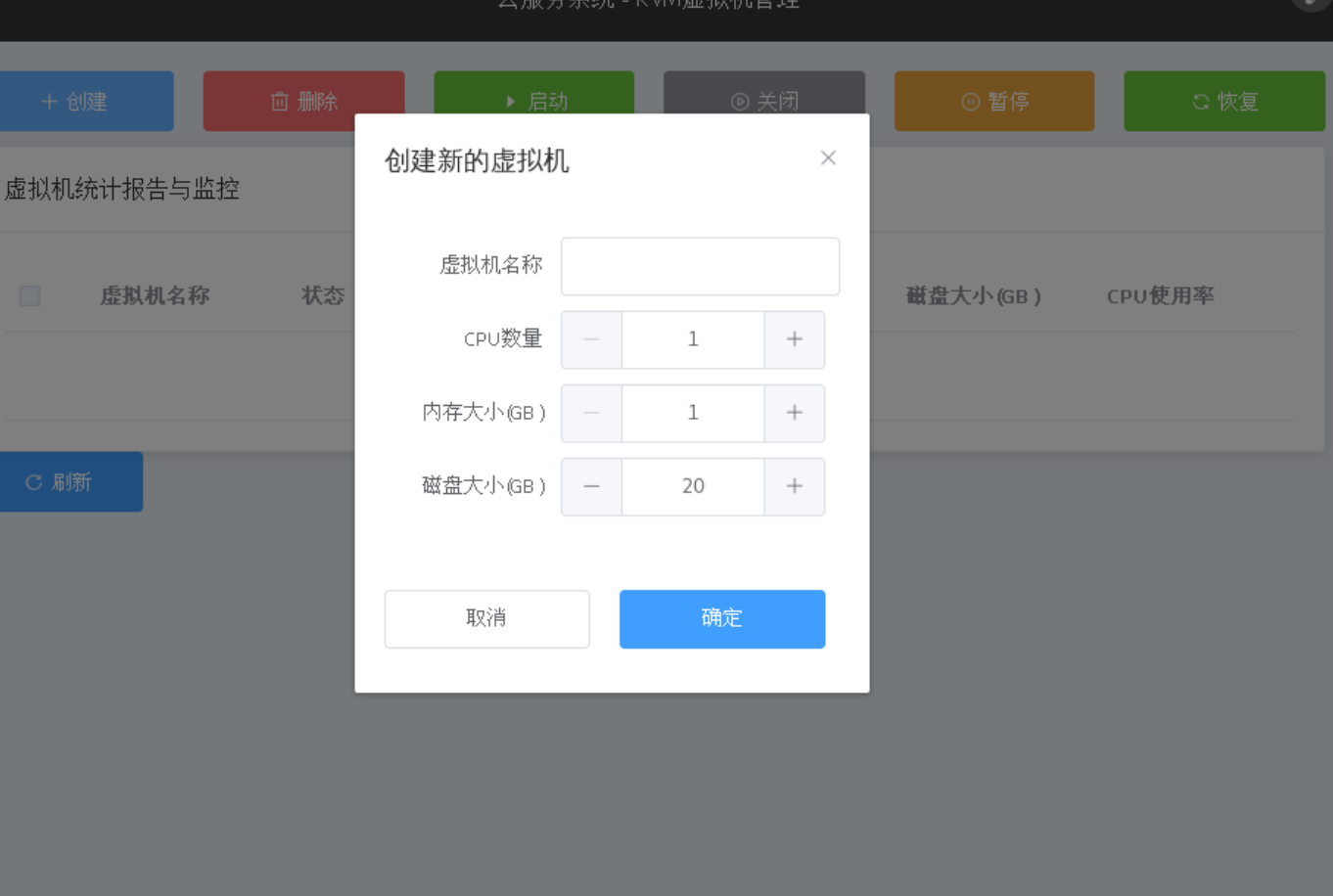
查看主机信息：



刷新虚拟机信息：



创建虚拟机：



创建虚拟机成功：



删除虚拟机失败：



删除虚拟机成功：



关闭虚拟机失败：



关闭虚拟机成功：



启动虚拟机失败：



启动虚拟机成功：



暂停虚拟机失败：



暂停虚拟机成功：



恢复虚拟机失败：



恢复虚拟机成功：



八、源码附录

**前端代码：**

**Homeview.vue:**

<template>

<div class="home">

<el-menu

default-active="1-4-1"

class="el-menu-vertical-demo"

background-color="#333"

>

<router-link :to="{ name: 'home' }">

<el-menu-item index="1" class="menu-item-center">

<i class="el-icon-s-home"></i>

</el-menu-item>

</router-link>

</el-menu>

<div class="center">

<router-view></router-view>

</div>

</div>

</template>

<style>

.home {

width: 100vw;

height: 100vh;

position: relative;

display: flex;

overflow: hidden;

}

.center {

position: absolute;

right: 0;

top: 0;

width: 95vw;

height: 100%;

}

.el-menu-vertical-demo {

box-sizing: border-box;

width: 5vw;

height: 100vh;

z-index: 9;

display: flex;

flex-direction: column;

justify-content: center;

}

.menu-item-center {

display: flex;

justify-content: center;

align-items: center;

}

.el-menu-vertical-demo .el-menu-item.is-active {

background-color: #000000;

}

.el-menu-item i {

color: #FFF;

font-size: 56px;

font-weight: bold;

}

</style>

**home.vue:**

<template>

<el-container style="height: 100vh; border: 1px solid #eee;">

<el-header>云服务系统 - KVM虚拟机管理</el-header>

<el-button class="host-info-btn" icon="el-icon-info"

@click="showHostInfoDialog = true; fetchHostInfo();"></el-button>

<el-dialog title="主机信息" :visible.sync="showHostInfoDialog" width="500px">

<div v-if="hostInfo" class="host-info-content">

<p>模型: {{ hostInfo.model }}</p>

<p>CPU数量: {{ hostInfo.cpu }}</p>

<p>内存大小: {{ hostInfo.memory }}</p>

<p>空闲内存: {{ hostInfo.free\_memory }}</p>

<p>磁盘大小: {{ hostInfo.disk }}</p>

<p>空闲磁盘: {{ hostInfo.free\_disk }}</p>

</div>

<div v-else>

暂无主机信息

</div>

</el-dialog>

<el-main>

<el-row :gutter="20">

<!-- VM生命周期管理按钮 -->

<el-col :span="4">

<el-button type="primary" icon="el-icon-plus" @click="openCreateVMDialog">创建</el-button>

</el-col>

<el-col :span="4">

<el-button type="danger" icon="el-icon-delete" @click="deleteVM">删除</el-button>

</el-col>

<el-col :span="4">

<el-button type="success" icon="el-icon-caret-right" @click="startVM">启动</el-button>

</el-col>

<el-col :span="4">

<el-button type="info" icon="el-icon-video-play" @click="shutdownVM">关闭</el-button>

</el-col>

<el-col :span="4">

<el-button type="warning" icon="el-icon-video-pause" @click="suspendVM">暂停</el-button>

</el-col>

<el-col :span="4">

<el-button type="success" icon="el-icon-refresh" @click="resumeVM">恢复</el-button>

</el-col>

</el-row>

<!-- 配置选项表单 -->

<el-dialog title="创建新的虚拟机" :visible.sync="showCreateVMDialog" width="350px">

<el-form :model="newVM" label-width="120px">

<el-form-item label="虚拟机名称">

<el-input v-model="newVM.name"></el-input>

</el-form-item>

<el-form-item label="CPU数量">

<el-input-number v-model="newVM.cpu" :min="1" :max="4"></el-input-number>

</el-form-item>

<el-form-item label="内存大小(GB)">

<el-input-number v-model="newVM.memory" :min="1" :max="10"></el-input-number>

</el-form-item>

<el-form-item label="磁盘大小(GB)">

<el-input-number v-model="newVM.disk" :min="1" :max="99"></el-input-number>

</el-form-item>

</el-form>

<span slot="footer" class="dialog-footer" style="display: flex; justify-content: space-between;">

<el-button @click="showCreateVMDialog = false">取消</el-button>

<el-button type="primary" @click="createVM">确定</el-button>

</span>

</el-dialog>

<!--统计报告与监控-->

<el-row>

<el-col :span="24">

<el-card>

<div slot="header" class="clearfix">

<span>虚拟机统计报告与监控</span>

</div>

<el-table :data="reportData" style="width: 100%" ref="multipleTable"

@selection-change="handleSelectionChange">

<el-table-column type="selection" width="55"></el-table-column>

<el-table-column prop="name" label="虚拟机名称"></el-table-column>

<el-table-column prop="KVM\_status" label="状态"></el-table-column>

<el-table-column prop="cpu" label="CPU数量"></el-table-column>

<el-table-column prop="memory" label="内存大小(GB)"></el-table-column>

<el-table-column prop="disk" label="磁盘大小(GB)"></el-table-column>

<el-table-column prop="CPU\_usage" label="CPU使用率"></el-table-column>

</el-table>

</el-card>

</el-col>

</el-row>

<el-col :span="3">

<el-button type="primary" icon="el-icon-refresh-right" @click="get\_reportData(true)">刷新</el-button>

</el-col>

<!-- 错误处理与报警信息显示 -->

<div v-for="alert in alerts" :key="alert.id">

<el-alert

:title="alert.msg"

:type="alert.type === 'success' ? 'success' : 'error'"

:icon="alert.type === 'success' ? 'el-icon-success' : 'el-icon-error'"

show-icon>

</el-alert>

</div>

</el-main>

<el-footer>小组成员：李安南、罗成勇、戚正、肖江伟</el-footer>

</el-container>

</template>

<script>

import axios from 'axios';

export default {

name: 'VmManagement',

data() {

return {

showCreateVMDialog: false,

newVM: {

name: '',

cpu: 1,

memory: 1,

disk: 20

},

reportData: [],

showHostInfoDialog: false,

hostInfo: null,

alertVisible: false,

alerts: [], // 存储多个提示信息

selectedVMs: [], // 存储被选中的虚拟机

};

},

mounted() {

this.get\_reportData(true);

},

methods: {

openCreateVMDialog() {

this.showCreateVMDialog = true;

},

handleSelectionChange(val) {

this.selectedVMs = val;

},

fetchHostInfo() {

axios.get('http://localhost:5000/api/hostInfo') // 假设这是获取主机信息的后端URL

.then(response => {

if (response.data) {

this.hostInfo = response.data; // 存储主机信息

} else {

this.showAlert('无法获取主机信息', 'error');

}

})

.catch(error => {

this.showAlert('请求错误: ' + error.message, 'error');

});

},

get\_reportData(showAlertOnSuccess = false) {

axios.get('http://localhost:5000/api/getInfo')

.then(response => {

console.log(response.data);

if (response.data && response.data.status === 'success') {

// 如果成功，清空原有数据并添加新数据

this.reportData = response.data.VM.map(vm => ({

name: vm.name,

KVM\_status: vm.KVM\_status,

cpu: vm.cpu,

memory: vm.memory,

disk: vm.disk,

CPU\_usage: vm.CPU\_usage

}));

if (showAlertOnSuccess) {

this.showAlert(response.data.message, 'success');

}

} else {

// 如果后端返回失败

this.showAlert('虚拟机信息获取失败', 'error');

}

})

.catch(error => {

// 网络或服务器错误

this.showAlert('请求错误: ' + error.message, 'error');

});

},

createVM() {

// 检查虚拟机名称

if (!this.newVM.name) {

this.showAlert('请输入虚拟机名称', 'error');

return;

}

// 检查虚拟机名称是否包含空格

if (this.newVM.name.includes(' ')) {

this.showAlert('虚拟机名称不能包含空格', 'error');

return;

}

const vmExists = this.reportData.some(vm => vm.name === this.newVM.name);

if (vmExists) {

this.showAlert('虚拟机名称已存在，请选择一个不同的名称', 'error');

return;

}

const vmData = {

operation: "create",

name: this.newVM.name,

cpu: this.newVM.cpu,

memory: this.newVM.memory,

disk: this.newVM.disk

};

axios.post(`http://localhost:5000/api/vm`, vmData)

.then(response => {

if (response.data && response.data.status === 'success') {

this.showAlert(response.data.message, 'success');

this.get\_reportData(); // 调用 get\_reportData 来刷新数据

} else {

this.showAlert(response.data.message, 'error');

}

})

.catch(error => {

this.showAlert('请求错误: ' + error.message, 'error');

})

.finally(() => {

this.newVM.name = '';

this.showCreateVMDialog = false;

});

},

deleteVM() {

// 检查所有选中的虚拟机是否都已关机

const allVmsShutdown = this.selectedVMs.every(vm => vm.KVM\_status === "已关机");

if (!allVmsShutdown) {

// 如果有虚拟机的状态不是已关机，则显示错误消息并返回

this.showAlert('请先关闭选中的所有虚拟机再进行删除操作', 'error');

return;

}

// 所有虚拟机都已关机，可以执行删除操作

this.performVMOperation('delete');

},

startVM() {

this.performVMOperation('start');

},

shutdownVM() {

this.performVMOperation('shutdown');

},

suspendVM() {

this.performVMOperation('suspend');

},

resumeVM() {

this.performVMOperation('resume');

},

performVMOperation(endpoint) {

if (this.selectedVMs.length > 0) {

this.selectedVMs.forEach(vm => {

const vmData = {

operation: endpoint,

name: vm.name,

};

// 执行相应的操作

axios.post(`http://localhost:5000/api/vm`, vmData)

.then(response => {

if (response.data && response.data.status === 'success') {

this.showAlert(response.data.message, 'success');

} else {

this.showAlert(response.data.message, 'error');

}

})

.catch(error => {

this.showAlert('请求错误: ' + error.message, 'error');

})

.finally(() => {

this.get\_reportData(); // 在所有请求完成后刷新数据

})

});

} else {

this.showAlert('没有选中的虚拟机', 'error');

}

},

showAlert(msg, type) {

const alert = {

id: Date.now(), // 生成唯一ID

msg: msg,

type: type,

};

this.alerts.push(alert);

setTimeout(() => {

this.alerts = this.alerts.filter(a => a.id !== alert.id);

}, 6000); // 5秒后移除提示

},

},

}

</script>

<style>

.el-header {

background-color: #333;

color: white;

text-align: center;

padding: 0 20px;

line-height: 60px;

}

.el-footer {

background-color: #333;

color: white;

text-align: center;

line-height: 60px;

}

.el-main {

background-color: #e9eef3;

padding: 20px;

overflow-y: auto; /\* 添加滚动条 \*/

}

/\* 扩展样式，确保内容区域占满剩余空间 \*/

.el-container {

display: flex;

flex-direction: column;

}

.el-main {

flex: 1;

}

/\* 表单和表格样式 \*/

.el-form-item {

margin-bottom: 10px;

}

/\* 报警信息样式 \*/

.el-alert {

margin-bottom: 20px;

}

/\* 使表单和按钮宽度一致 \*/

.el-form-item, .el-button {

width: 100%;

}

/\* 表格样式调整 \*/

.el-table {

background-color: white;

}

/\* 按钮间距 \*/

.el-button {

margin: 0 10px 10px 0;

}

/\* 适应所有屏幕尺寸 \*/

@media (max-width: 768px) {

.el-form-item, .el-button {

width: 100%;

}

}

.host-info-content {

font-size: 20px;

}

.host-info-btn {

position: fixed;

top: 10px;

right: 5px;

z-index: 100;

font-size: 36px;

border-radius: 50%;

width: 30px;

height: 30px;

display: flex;

align-items: center;

justify-content: center;

padding: 0;

}

</style>

**后端代码：**

**main.py:**

import libvirt

import os

import psutil

import vmop1

import vmop2

import vmop3

def operation\_bus(operation, arg):

"""

接受命令和参数表的命令执行总线

:param operation: 从json解析出的操作 str

:param arg: 从json解析出的含相关参数字典 dict[str:object]

:return: 符合后端样例的操作结果

"""

conn, message = vmop1.get\_conn()

# 后端发回

data = {}

# 执行操作

if conn:

if operation == "create":

message += vmop3.vm\_create\_with\_check(conn, arg["name"], arg["memory"], arg["cpu"], arg["disk"])

data["name"] = arg["name"]

data["cpu"] = arg["cpu"]

data["memory"] = arg["memory"]

data["disk"] = arg["disk"]

elif operation == "delete":

message += vmop3.vm\_delete(conn, arg["name"])

elif operation == "start":

message += vmop2.vm\_start(conn, arg["name"])

elif operation == "suspend":

message += vmop2.vm\_suspend(conn, arg["name"])

elif operation == "resume":

message += vmop2.vm\_resume(conn, arg["name"])

elif operation == "shutdown":

message += vmop2.vm\_destroy(conn, arg["name"])

else:

message += "未知命令 "

message += vmop1.close\_conn(conn)

data["message"] = message

data["status"] = message\_check(message)

return data

def get\_all\_vm\_info():

conn, message = vmop1.get\_conn()

# 后端发回

data = {}

# 执行操作

try:

VM = vmop1.show\_vm(conn)

message += "成功获取全部虚拟机 "

except Exception as e:

VM = []

message += "获取全部虚拟机失败 "

print(e)

message += vmop1.close\_conn(conn)

data["message"] = message

data["status"] = message\_check(message)

data["VM"] = VM

return data

def message\_check(message):

if "失败" in message:

return "error"

else:

return "success"

def get\_host\_info(): # 主机信息

conn, message = vmop1.get\_conn()

# 后端发回

data = {}

# 执行操作

nodeInfo = conn.getInfo() # 获取虚拟化主机信息

data["model"] = str(nodeInfo[0])

data["cpu"] = str(nodeInfo[2])

data["memory"] = f"{nodeInfo[1]//1024} GB"

# 获取内存信息

memory\_info = psutil.virtual\_memory()

# 获取空闲内存大小

mem = memory\_info.available

data["free\_memory"] = f"{mem/1024\*\*3:.1f} GB" # 主机空闲内存

partitions = os.statvfs("/")

total\_size = partitions.f\_frsize \* partitions.f\_blocks

data["disk"] = f"{total\_size / (1024 \*\* 3):.2f} GB"

data["free\_disk"] = f"{vmop1.check\_remain\_size(conn)//1024\*\*3} GB" # 主机理论磁盘空闲

message += vmop1.close\_conn(conn)

return data

if \_\_name\_\_ == '\_\_main\_\_':

# 打开qemu

conn, message = vmop1.get\_conn()

print(get\_host\_info())

vmop1.close\_conn(conn)

**client.py:**

from flask import Flask, request, jsonify

from flask\_cors import CORS # 引入 CORS

import main

app = Flask(\_\_name\_\_)

CORS(app) # 应用 CORS 到您的 Flask 应用

@app.route('/api/vm', methods=['POST'])

def vm():

# 获取前端传递的虚拟机信息

data = request.get\_json()

operation = data["operation"]

# 在这里执行KVM管理操作

data = main.operation\_bus(operation, data)

return jsonify(data)

@app.route('/api/getInfo', methods=['GET'])

def get\_info():

# 在这里执行获取KVM信息

data = main.get\_all\_vm\_info()

return jsonify(data)

@app.route('/api/hostInfo', methods=['GET'])

def get\_host\_info():

# 在这里执行获取主机信息

data = main.get\_host\_info()

return jsonify(data)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**vmop1.py:**

import libvirt

import time

def check\_allow\_size(conn, disk):

pool\_total\_remain = check\_remain\_size(conn)

disk\_size = disk \* 1024 \* 1024 \* 1024

if disk\_size > pool\_total\_remain:

return True

else:

return False

def check\_remain\_size(conn):

pool = conn.storagePoolLookupByName("default")

pool\_info = pool.info()

pool\_available = pool\_info[3]

volumes = pool.listVolumes()

volume\_real\_size = 0

for volume in volumes:

vol = pool.storageVolLookupByName(volume)

vol\_info = vol.info()

vol\_size = vol\_info[2]

volume\_real\_size += vol\_size

volume\_total\_size = 0

for volume in volumes:

vol = pool.storageVolLookupByName(volume)

vol\_info = vol.info()

vol\_size = vol\_info[1]

volume\_total\_size += vol\_size

pool\_total\_remain = pool\_available + volume\_real\_size - volume\_total\_size

return pool\_total\_remain

def get\_conn():

# 打开qemu

conn = libvirt.open("qemu:///system")

if conn is None:

message = "连接失败 "

else:

message = "连接成功 "

return conn, message

def close\_conn(conn): # 关闭连接

try:

conn.close()

message = "成功关闭连接 "

except:

message = "关闭连接失败 "

return message

def show\_vm(conn):

vm = []

domains = conn.listAllDomains() # 获取所有虚拟机

for domain in domains:

detail = {}

name = domain.name()

detail["name"] = name

state, \_ = domain.state()

KVM\_status = "未知"

if state == libvirt.VIR\_DOMAIN\_RUNNING:

KVM\_status = "运行中"

CPU\_usage = caclulate\_CPU\_usage(domain)

elif state == libvirt.VIR\_DOMAIN\_PAUSED:

KVM\_status = "已暂停"

CPU\_usage = "0%"

elif state == libvirt.VIR\_DOMAIN\_SHUTOFF:

KVM\_status = "已关机"

CPU\_usage = "0%"

detail["KVM\_status"] = KVM\_status

detail["CPU\_usage"] = CPU\_usage

cpu = domain.info()[3]

detail["cpu"] = cpu

memory = domain.info()[2] // 1024 \*\* 2

detail["memory"] = memory

pool = conn.storagePoolLookupByName("default")

vol = pool.storageVolLookupByName(f"{name}.qcow2")

vol\_info = vol.info()

disk = vol\_info[1] // 1024 \*\* 3

detail["disk"] = disk

vm.append(detail)

return vm

def caclulate\_CPU\_usage(domain):

# 获取虚拟机CPU使用率（需要两次获取状态间的时间差）

cpu\_time1 = domain.getCPUStats(True)[0]['cpu\_time']

time.sleep(1) # 等待一秒

cpu\_time2 = domain.getCPUStats(True)[0]['cpu\_time']

cpu\_usage = ((cpu\_time2 - cpu\_time1) / 1e9) \* 100 # 将ns转换为秒并计算CPU使用率

return f"{int(cpu\_usage)}%"

**vmop2.py:**

import libvirt

def vm\_start(conn, name):

try:

domain = conn.lookupByName(name)

domain.create()

message = f"{name} 启动成功 "

except Exception as e:

message = f"{name} 启动失败 "

print(e)

return message

def vm\_suspend(conn, name):

try:

domain = conn.lookupByName(name)

domain.suspend()

message = f"{name} 暂停成功 "

except Exception as e:

message = f"{name} 暂停失败 "

print(e)

return message

def vm\_resume(conn, name):

try:

domain = conn.lookupByName(name)

domain.resume()

message = f"{name} 重启成功 "

except Exception as e:

message = f"{name} 重启失败 "

print(e)

return message

def vm\_destroy(conn, name):

try:

domain = conn.lookupByName(name)

domain.destroy()

message = f"{name} 关闭成功 "

except Exception as e:

message = f"{name} 关闭失败 "

print(e)

return message

**vmop3.py:**

import vmop1

# 定义ISO镜像文件路径

iso = "/home/lan/CentOS-7-x86\_64-DVD-2009.iso"

def vm\_check(conn, memory, cpu, disk): # 主机信息

node\_info = conn.getInfo() # 获取虚拟化主机信息

memory\_max = node\_info[1] // 1024

if memory > memory\_max:

message = "内存过载 "

return False, message

cpu\_max = node\_info[2]

if cpu > cpu\_max:

message = "CPU过载 "

return False, message

if vmop1.check\_allow\_size(conn, disk):

message = "磁盘过载 "

return False, message

message = "硬件申请成功 "

return True, message

def vm\_create\_with\_check(conn, name, memory, cpu, disk):

check, message = vm\_check(conn, memory, cpu, disk)

if check:

message += vm\_create(conn, name, memory, cpu, disk)

else:

message += "硬件资源不足 创建虚拟机失败 "

return message

def vm\_create(conn, name, memory, cpu, disk):

# 创建磁盘

message, fault = create\_img(conn, name, disk)

try:

# 检查磁盘

if fault:

raise ValueError("disk not found")

# 创建虚拟机

xml\_desc = \

f'''

<domain type='kvm'>

<name>{name}</name>

<memory unit='G'>{memory}</memory>

<vcpu placement='static'>{cpu}</vcpu>

<os>

<type arch='x86\_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>

</os>

<features>

<acpi/>

<apic/>

</features>

<cpu mode='custom' match='exact' check='partial'>

<model fallback='allow'>Broadwell-noTSX-IBRS</model>

<feature policy='require' name='md-clear'/>

<feature policy='require' name='spec-ctrl'/>

<feature policy='require' name='ssbd'/>

</cpu>

<clock offset='utc'>

<timer name='rtc' tickpolicy='catchup'/>

<timer name='pit' tickpolicy='delay'/>

<timer name='hpet' present='no'/>

</clock>

<on\_poweroff>destroy</on\_poweroff>

<on\_reboot>restart</on\_reboot>

<on\_crash>destroy</on\_crash>

<pm>

<suspend-to-mem enabled='no'/>

<suspend-to-disk enabled='no'/>

</pm>

<devices>

<emulator>/usr/libexec/qemu-kvm</emulator>

<disk type='file' device='disk'>

<driver name='qemu' type='qcow2'/>

<source file='/var/lib/libvirt/images/{name}.qcow2'/>

<target dev='hda' bus='ide'/>

<boot order='1'/>

<address type='drive' controller='0' bus='0' target='0' unit='0'/>

</disk>

<disk type='file' device='cdrom'>

<driver name='qemu' type='raw'/>

<source file='{iso}'/>

<target dev='hdb' bus='ide'/>

<readonly/>

<boot order='2'/>

<address type='drive' controller='0' bus='0' target='0' unit='1'/>

</disk>

<controller type='usb' index='0' model='ich9-ehci1'>

<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x7'/>

</controller>

<controller type='usb' index='0' model='ich9-uhci1'>

<master startport='0'/>

<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0' multifunction='on'/>

</controller>

<controller type='usb' index='0' model='ich9-uhci2'>

<master startport='2'/>

<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x1'/>

</controller>

<controller type='usb' index='0' model='ich9-uhci3'>

<master startport='4'/>

<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x2'/>

</controller>

<controller type='pci' index='0' model='pci-root'/>

<controller type='ide' index='0'>

<address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>

</controller>

<controller type='virtio-serial' index='0'>

<address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>

</controller>

<interface type='network'>

<mac address='52:54:00:ee:f2:ad'/>

<source network='default'/>

<model type='rtl8139'/>

<address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>

</interface>

<serial type='pty'>

<target type='isa-serial' port='0'>

<model name='isa-serial'/>

</target>

</serial>

<console type='pty'>

<target type='serial' port='0'/>

</console>

<channel type='spicevmc'>

<target type='virtio' name='com.redhat.spice.0'/>

<address type='virtio-serial' controller='0' bus='0' port='1'/>

</channel>

<input type='mouse' bus='ps2'/>

<input type='keyboard' bus='ps2'/>

<graphics type='spice' autoport='yes'>

<listen type='address'/>

<image compression='off'/>

</graphics>

<sound model='ich6'>

<address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>

</sound>

<video>

<model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1' primary='yes'/>

<address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>

</video>

<redirdev bus='usb' type='spicevmc'>

<address type='usb' bus='0' port='1'/>

</redirdev>

<redirdev bus='usb' type='spicevmc'>

<address type='usb' bus='0' port='2'/>

</redirdev>

<memballoon model='virtio'>

<address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>

</memballoon>

</devices>

</domain>

'''

conn.defineXML(xml\_desc)

message += "创建虚拟机成功 "

except Exception as e:

message = '创建虚拟机失败 '

print(e)

return message

def create\_img(conn, name, size):

try:

pool = conn.storagePoolLookupByName('default')

vol\_xml = f"""

<volume type='file'>

<name>{name}.qcow2</name>

<capacity unit='G'>{size}</capacity>

<allocation unit='G'>{size}</allocation>

<target>

<format type='qcow2'/>

</target>

</volume>

"""

vol = pool.createXML(vol\_xml, 0)

message = '磁盘创建成功 '

fault = False

except Exception as e:

message = '磁盘创建失败 '

fault = True

print(e)

return message, fault

def vm\_delete(conn, name):

dom = conn.lookupByName(name)

message, fault = delete\_img(conn, name)

try:

if fault:

raise ValueError("disk not found")

dom.undefine()

message += f"虚拟机{name} 删除成功 "

except Exception as e:

message = f"虚拟机{name} 删除失败 "

print(e)

return message

def delete\_img(conn, name):

try:

pool = conn.storagePoolLookupByName("default")

vol = pool.storageVolLookupByName(f"{name}.qcow2")

vol.delete(0)

message = "磁盘删除成功 "

fault = False

except Exception as e:

message = '磁盘删除失败 '

fault = True

print(e)

return message, fault