PLATAFORMA DE GESTION DE RECURSOS EN LA NUBE

202300451 – Ever Alexander Hernández Lemus
202113309 - Noé Yahir Mazariegos Gramajo

Resumen

Crear una plataforma la cual realice la gestión de recursos en la nube mediante la implementación de estructuras de datos como las listas simples, listas enlazadas, el procesamiento de los datos de archivos .XML como su nueva generación con la programación orientada a objetos.

CloudSync es un sistema de gestión distribuida en recursos de la nube el cual permite a usuarios a alquilar una maquina virtual para utilizar sus recursos teniendo el beneficio de variar sus recomendaciones para cada necesidad que tenga el usuario.

CloudSysnc cuenta con los componentes fundamentales los cuales son Centros de datos, maquinas virtuales, contenedores y solicitudes de procesamiento los cuales van a ir programados con un conjunto de estructuras de datos creadas por uno mismo sin utilizar las listas nativas de Python creando un mejor orden al saber cómo manipular los datos.

Palabras clave
Estructuras TDAs

Deploy

Backup

VM

Nube

Abstract

Create a platform that manages cloud resources by implementing data structures such as simple lists, linked lists, and processing XML file data using object-oriented programming.

CloudSync is a distributed cloud resource management system that allows users to rent a virtual machine to access its resources, with the benefit of customized recommendations tailored to each user's needs.

CloudSync includes fundamental components such as data centers, virtual machines, containers, and processing requests. These components are programmed using a set of custom data structures, bypassing Python's native lists, resulting in better organization and a clearer understanding of data manipulation.

Keywords
Desplegar

Respaldo

Estructuras de Datos de Tipo Abstracto

Máquinas Virtuales

Introducción

Se presentará los fundamentos que se utilizaron para poder completar el programa requerido de la plataforma de gestión de recursos en la nube dando breves resúmenes de cada uno mostrando bloques de código de los mas importantes para dar una idea general de como es que funciona el programa y como se maneja iniciando con diagramas de flujo y también de UML.

CloudSync utiliza estructuras de datos entrelazados utilizando archivos .XML buscado como objetivo es crear un sistema que crea estructuras de datos personalizados creación de reportes y visualización de las gráficas mediante graphiz.

La gestión de recursos en la nube

Los computadores de la nube han revolucionado el mercado de como se organiza con sus gestiones y escalas de sus recursos tecnológicos, dando modelos que permiten acceder bajo demanda de las personas que lo necesitan con un precio razonable dependiendo de las configuraciones que necesita el cliente realizando un conjunto compartido de recursos computacionales como redes, servidores, almacenamiento, aplicaciones y servicios. El proyecto CloudSync se enmarca en la simulación de un sistema de gestión distribuida de recursos, donde la eficiencia en la asignación de CPU, RAM y almacenamiento se convierte en un factor critico para el rendimiento del sistema.

El agregar las estructuras de datos entrelazados como son las listan simples, dobles, circulares, pilas y colas de prioridad permiten representar de una manera dinámica y flexible los componentes del sistema que son: centros de datos, maquinas virtuales, conenedores y solicitudes sin utilizar las listas nativas de Python dando como resultado un mejor entendimiento con el código del proyecto

Procesamiento de archivos XML como ingreso y reportes

Al utilizae los archivos XML en CoudSync cumple un propósito servir como mecanismo de inicio del sistema y reportar en un formato de salida con la misma estructura para tener un informe de como es que se han organizado los datos obtenidos, si en un caso se requiera iniciar el sistema con los datos previamente modificados del sistema se ingresa el archivo de salida en el archivo de entrada para que se siga trabajando con el mismo requisito previos.

Estructuras de datos abstratos(TDAs)

Las Estructuras de Datos Abstractas (TDAs) son modelos conceptuales que definen un conjunto de datos y las operaciones permitidas sobre ellos, independientemente de su implementación concreta. Proporcionan una abstracción que permite organizar, almacenar y manipular información de manera eficiente, facilitando el

diseño de algoritmos y sistemas software robustos y escalables.

A continuación, una breve explicación de cada TDAs

- **Listas enlazadas:** Es un conjunto de nodos enlazados secuencialmente, donde cada nodo contiene datos.
- **Pilas:** Realiza estructuras tipo LIFO el ultimo eleméntenlo en ser ingresado es el primero en ser retirado.
- **Colas:** Realiza estructuras tipo FIFO el primer elemento en ser ingresado es el primero en ser retirado.
- **Arboles:** Estructuras jerárquicas compuestas con nodos conectados por lazos que tiene nodo raíz e hijos.

Explicación de los bloques de códigos más importantes

```python
class Nodo:
    def __init__(self, data = None):
        self.data = data
        self.siguiente = None
```

Se utiliza bloque de construcción para las listas que uno comience a construir, este almacena cualquier dato que uno valla a ingresar.

```python
class ListaSimple:
    def __init__(self):
        self.primero = None

    def insertar(self, dato):
        nuevo = Nodo(dato)
        if self.primero is None:
            self.primero = nuevo
        else:
            actual = self.primero
            while actual.siguiente is not None:
                actual = actual.siguiente
            actual.siguiente = nuevo

    def contar(self):
        actual = self.primero
        c = 0
        while actual:
            c += 1
            actual = actual.siguiente
        return c
```

Realiza los llamados utilizando los nodos y construyendo una lista donde se insertan los nuevos datos y el que no tenga siguiente el puntero apunta a None.

```python
def cargar_xml(tipo):
    nombre_archivo = input(f"Ingrese el nombre del archivo XML de {tipo} (ejemplo: {tipo}.xml): ").strip()
    if nombre_archivo == '':
        return ''
    ruta_archivo = f'./entrada/{nombre_archivo}'

    try:
        with open(ruta_archivo, 'r', encoding='utf-8'):
            pass
        return ruta_archivo

    except Exception as e:
        print(f'archivo no encontrado: {ruta_archivo}')
        return ''
```

Verifica que el archivo XML existe para devolver la ruta

```python
# LEEENDO ETIQUETA CONFIGURACION
configuracion = root.find('configuracion')
if configuracion is not None:
    centros = configuracion.find('centrosDatos')
if centros is not None:
    print('*' * 45)
    print("CENTROS DE DATOS:")

for centro in centros.findall('centro'):
    id = centro.get('id', '')
    nombre = centro.get('nombre', '')
    ubicacion = centro.find('ubicacion', '')
    pais = ubicacion.find('pais').text
    ciudad = ubicacion.find('ciudad').text
    capacidad = centro.find('capacidad')
    cpu = capacidad.find('cpu').text
    ram = capacidad.find('ram').text
    almacenamiento = capacidad.find('almacenamiento').text

    print('* ' * 40)
    print('Centro De Datos')
    print('id: ', id)
    print('Nombre: ', nombre)
    print('Ciudad', ciudad)
    print('Cpu: ', cpu)
    print('Ram :', ram)
    print('Almacenamiento: ', almacenamiento)
```

Lector de etiquetas del archivo XML

```python
def cargar_centros_a_lista(ruta, lista_centros):
    tree = ET.parse(ruta)
    root = tree.getroot()

    configuracion = root.find('configuracion')
    if configuracion is None:
        return

    centros = configuracion.find('centrosDatos')
    if centros is None:
        return

    for c in centros.findall('centro'):
        id = c.get('id')
        capacidad = c.find('capacidad')

        cpu = int(capacidad.find('cpu').text)
        ram = int(capacidad.find('ram').text)
        almacenamiento = int(capacidad.find('almacenamiento').text)

        centro = CentroDatos(id, cpu, ram, almacenamiento)
        lista_centros.insertar(centro)
```

Carga los centros de datos a la lista para poder manipularlos después.

```python
class CentroDatos:
    def __init__(self, id, cpu, ram, almacenamiento):
        self.id = id

        self.cpu_total = int(cpu)
        self.ram_total = int(ram)
        self.alm_total = int(almacenamiento)
        self.cpu_disponible = int(cpu)
        self.ram_disponible = int(ram)
        self.alm_disponible = int(almacenamiento)
        self.vms_activas = 0

    def mostrar_info(self):
        cpu_usado = self.cpu_total - self.cpu_disponible
        ram_usado = self.ram_total - self.ram_disponible
        alm_usado = self.alm_total - self.alm_disponible

        print('*' * 45)
        print('Centro ID:', self.id)
        print('CPU Total:', self.cpu_total, '| CPU Disponible:', self.cpu_disponible, '| CPU Usado:', cpu_usado)
        print('RAM Total:', self.ram_total, '| RAM Disponible:', self.ram_disponible, '| RAM Usada:', ram_usado)
        print('Almacenamiento Total:', self.alm_total, '| Disponible:', self.alm_disponible, '| Usado:', alm_usado)
        print('Vms Activas:', self.vms_activas)

        cpu_p = (cpu_usado * 100) / self.cpu_total
        print(f'CPU usada (%): {cpu_p:.2f}%')
        print('*' * 40)
```

Manipulación de los datos obtenidos para la visualización de los datos en la terminal.

```python
centro_id = input('ingrese el ID del centro: ').strip()

print(f"\n=== VMs en {centro_id} ===")

actual = lista_vms.primero
contador = 1
existe = False

while actual:
    vm = actual.dato

    if vm.centro_id == centro_id:
        vm.mostrar_general_vm(contador)
        contador += 1
        existe = True

    actual = actual.siguiente
```

Verificación de si existen los centros a través de su id

```python
nodo_vm = lista_vms.primero
vm_encontrado = None
while nodo_vm:
    if nodo_vm.dato.id == id_vm:
        vm_encontrado = nodo_vm.dato
        break
    nodo_vm = nodo_vm.siguiente
if vm_encontrado is None:
    print('No se encontro la VM con el id ')
    return
print(f"Vm Seleccionado: {vm_encontrado.id} (Actual centro:){vm_encontrado.centro_id}")
print(f"Recursos requeridos: Cpu{vm_encontrado.cpu}, Ram:{vm_encontrado.ram}GB, Almace

id_destino = input("ingresa el ID del centro de datos: ").strip()

if id_destino == vm_encontrado.centro_id:
    print('error es el mismo centro actual')
    return
#----------------------------------------------------------------Busca centro destino
nodo_centro = lista_centros.primero
centro_destino = None
while nodo_centro:
    if nodo_centro.dato.i  (variable) nodo_centro: Any
        centro_destino = nodo_centro.dato
        break
    nodo_centro = nodo_centro.siguiente
```

Realiza la búsqueda de Vm y los va ingresando en nodos donde se guardara su ubicación

```
#-----------------------------------------------validar si cenrto tiene espacio
if (centro_destino.cpu_disponible >= vm_encontrado.cpu and centro_destino.ram_disponible >= vm_encontrado.ram and
    centro_destino.alm_disponible >= vm_encontrado.almacenamiento):
    nodo_antiguo = lista_centros.primero
    centro_antiguo = None
    while nodo_antiguo:
        if nodo_antiguo.dato.id == vm_encontrado.centro_id:
            centro_antiguo = nodo_antiguo.dato
            break
        nodo_antiguo = nodo_antiguo.siguiente

    if centro_antiguo:  (function) cpu_disponible: Any
        centro_antiguo.cpu_disponible += vm_encontrado.cpu
        centro_antiguo.ram_disponible += vm_encontrado.ram
        centro_antiguo.alm_disponible += vm_encontrado.almacenamiento
        centro_antiguo.vms_activas -= 1

    centro_destino.cpu_disponible -= vm_encontrado.cpu
    centro_destino.ram_disponible -= vm_encontrado.ram
    centro_destino.alm_disponible -= vm_encontrado.almacenamiento
    centro_destino.vms_activas += 1

    vm_encontrado.centro_id = id_destino

    print('la accion se a realizado con exito')
else:
    print('Error el centro no tiene recursos disponibles')
```

Para validar si se tiene espacio disponible

```
def procesar_solicitudes(lista_solicitudes, lista_centros):
    actual = lista_solicitudes.primero
    if actual is None:
        print("No hay solicitudes para procesar")
        return

    while actual:
        solicitud = actual.dato
        centro_actual = lista_centros.primero

        while centro_actual:
            centro = centro_actual.dato
            if (centro.cpu_disponible >= solicitud.cpu and
                centro.ram_disponible >= solicitud.ram and
                centro.alm_disponible >= solicitud.almacenamiento):

                centro.cpu_disponible -= solicitud.cpu
                centro.ram_disponible -= solicitud.ram
                centro.alm_disponible -= solicitud.almacenamiento

                print(f"Solicitud {solicitud.id} procesada en centro {centro.id}")
                break

            centro_actual = centro_actual.siguiente

        actual = actual.siguiente
```

Procesa cada solicitud asignándole el primer centro que tenga recursos suficientes

```
def generar_salida_xml(lista_centros, lista_vms, ruta="salida/salida.xml"):

    timestamp = ET.SubElement(root, "timestamp")
    timestamp.text = datetime.now().isoformat()

    estado_centros = ET.SubElement(root, "estadoCentros")

    total_vms = 0
    total_contenedores = 0

    actual_centro = lista_centros.primero
    while actual_centro:
        centro = actual_centro.dato

        centro_xml = ET.SubElement(
            estado_centros, "centro", {"id": centro.id}
        )

        nombre = ET.SubElement(centro_xml, "nombre")
        nombre.text = centro.id   # si no tienes nombre, usamos el ID

        recursos = ET.SubElement(centro_xml, "recursos")

        cpu_total = ET.SubElement(recursos, "cpuTotal")
        cpu_total.text = str(centro.cpu_total)

        cpu_disp = ET.SubElement(recursos, "cpuDisponible")
        cpu_disp.text = str(centro.cpu_disponible)

        cpu_usada = centro.cpu_total - centro.cpu_disponible
        cpu_util = ET.SubElement(recursos, "cpuUtilizacion")
        cpu_util.text = f"{(cpu_usada * 100) / centro.cpu_total:.2f}%"

        ram_total = ET.SubElement(recursos, "ramTotal")
        ram_total.text = str(centro.ram_total)
```

Reorganizar los nuevos datos en formato xml para un registro de los datos.

```
while True:
    print('*' * 20 + 'CONTROL CENTRO DE DATOS' + '*' * 20)
    print('1. Cargar XML')
    print('2. Gestion de centros de datos')
    print('3. Gestion de maquinas virtuales')
    print('4. Gestion de contenedores')
    print('5. Gestion de solicitudes')
    print('6. Menu Reportes')
    print('7. Generar Salida XML')
    print('8. Salir')

    op = input('Seleccione una opcion: ')

    if op == '1':
        ruta = cargar_xml('configuracion')
        if ruta:
            cargar_centros_a_lista(ruta, lista_centros)
            cargar_mv_a_lista(ruta, lista_vms)
            cargar_solicitudes_desde_xml(ruta, lista_solicitudes,cola)
            print("XML cargado correctamente")

    elif op == '2':
        menu_centros(lista_centros)

    elif op == '3':
        menu_mavi(lista_vms, lista_centros)

    elif op == '4':
        menu_conte(lista_vms)

    elif op == '5':
        menu_solicitudes(lista_solicitudes, lista_centros, cola)
```

El menú donde se ingresara en las funciones del sistema.

Conclusiones

En el desarrollo de sistemas de software complejos y escalables, la implementación de listas enlazadas, colas de prioridad, pilas y otras estructuras definidas por el usuario ha demostrado que la elección adecuada de un ADT no es sólo una decisión técnica sino también una estrategia fundamental para garantizar la eficiencia, la mantenibilidad y la claridad conceptual de una solución informática.