**VCC-Assignment3**

**M23AID053-Bhuvaneswari J**

**Installation of Virtual box:**

Downloaded windows host version from below link and installed Virtual box.

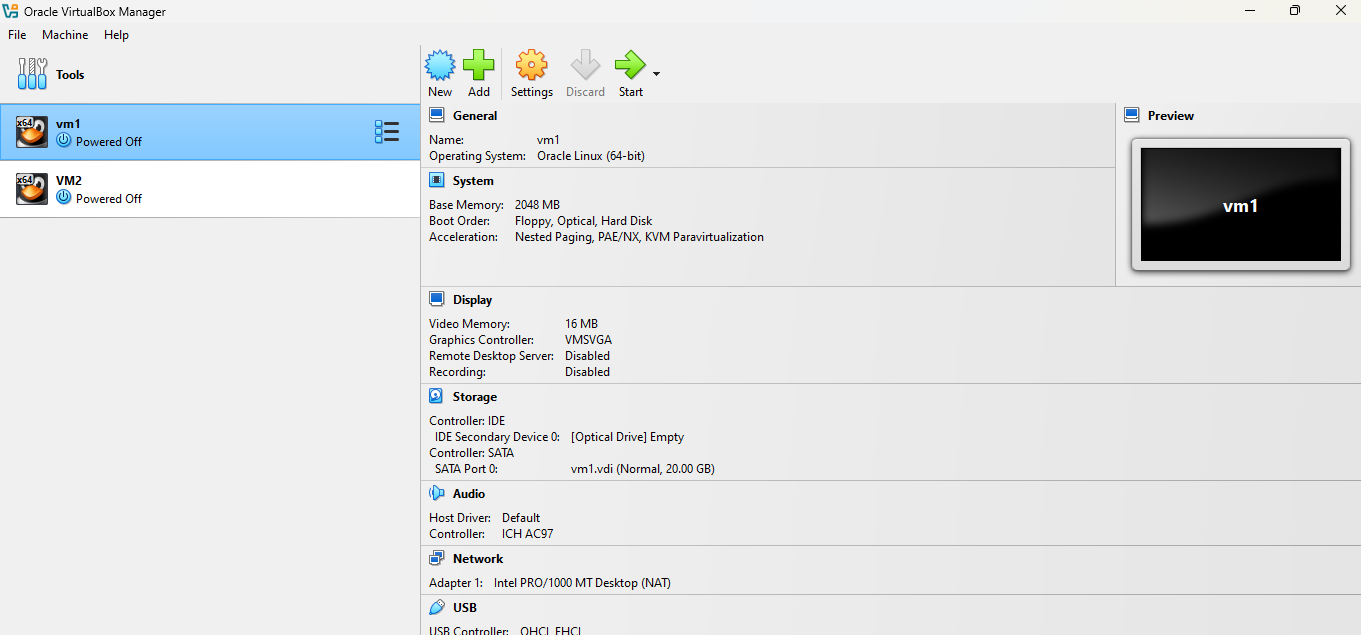
[**https://www.virtualbox.org/wiki/Downloads**](https://www.virtualbox.org/wiki/Downloads)

**Creation of VMs:**

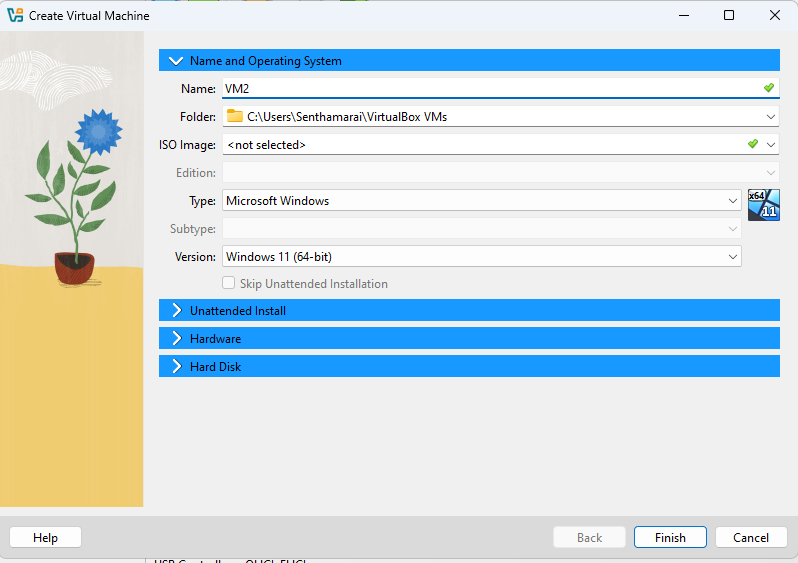
Downloaded puppy linux iso file (20.04 - [F96-CE](https://f96.puppylinux.com/) (64 bit)) from below link before VM creation.

[**https://forum.puppylinux.com/puppy-linux-collection**](https://forum.puppylinux.com/puppy-linux-collection)

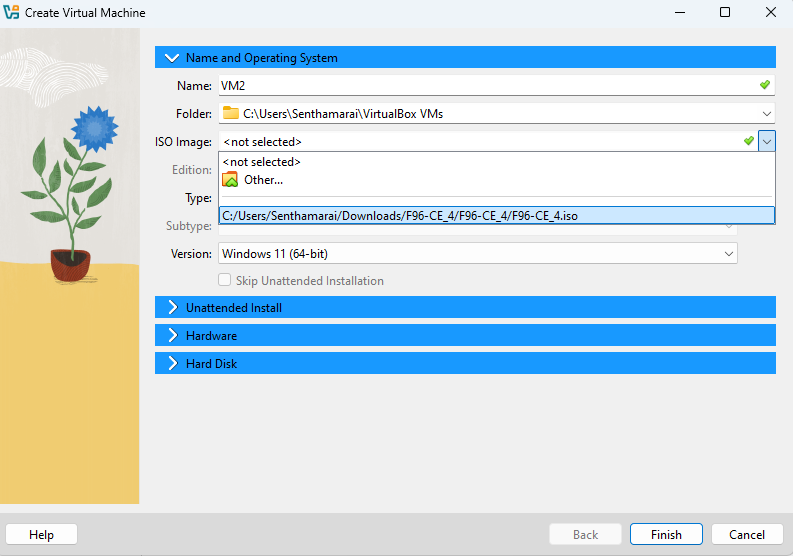
Click New:

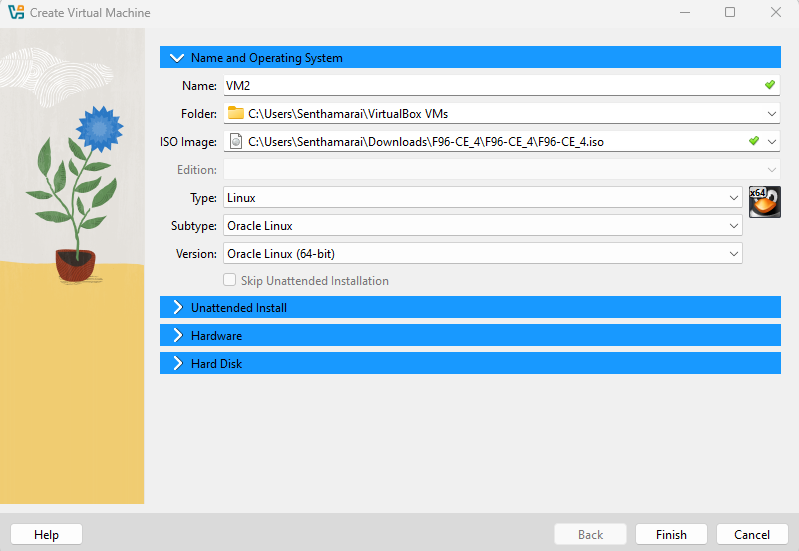
****

Add VM name

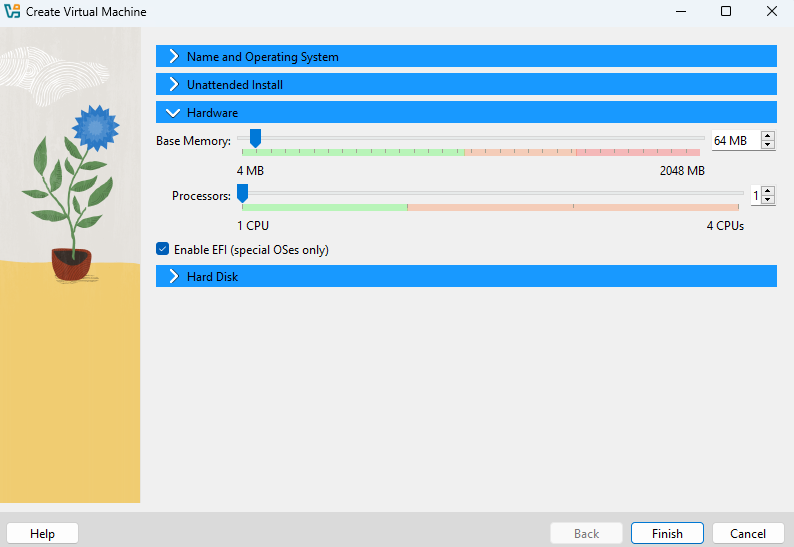
****

Upload puppy linux ISO file

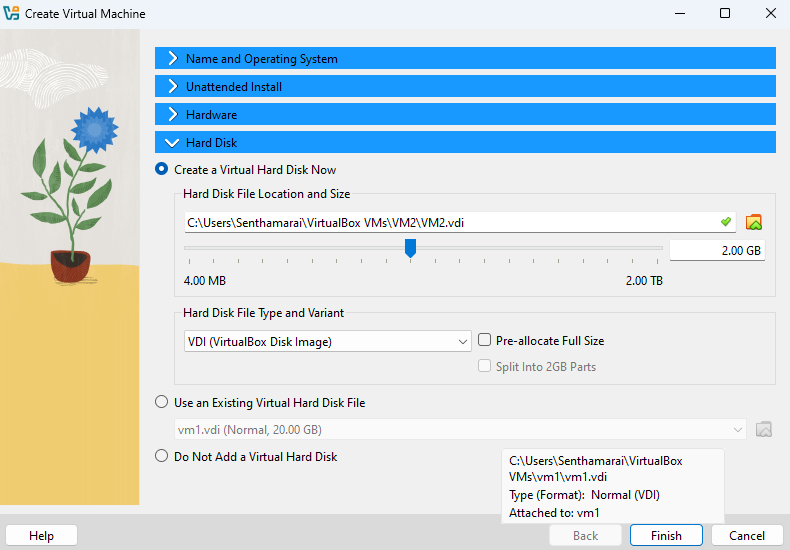
****

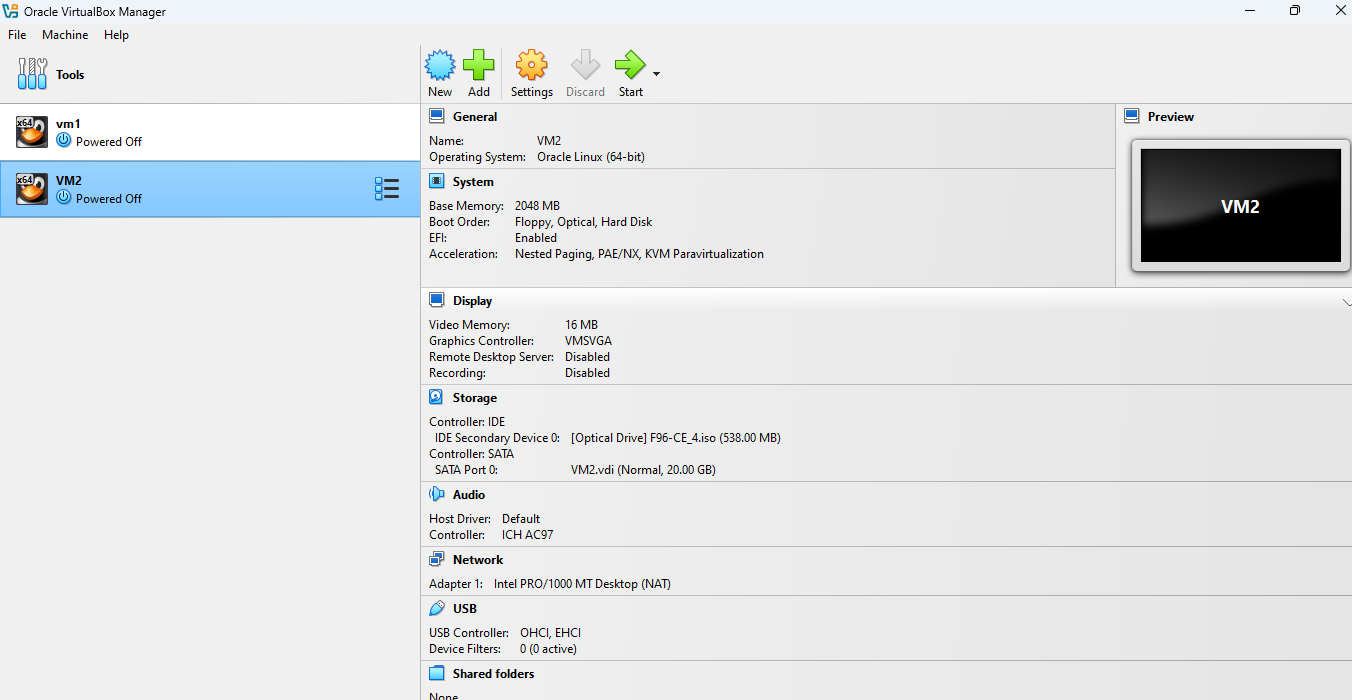
****

Choose memory size

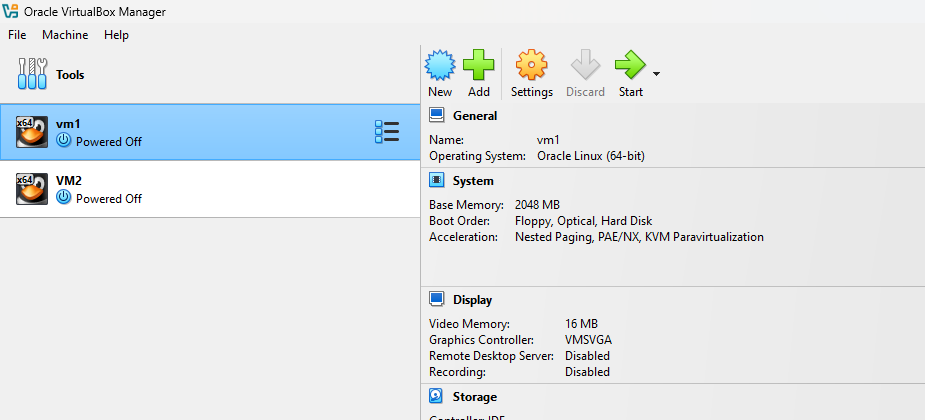
****

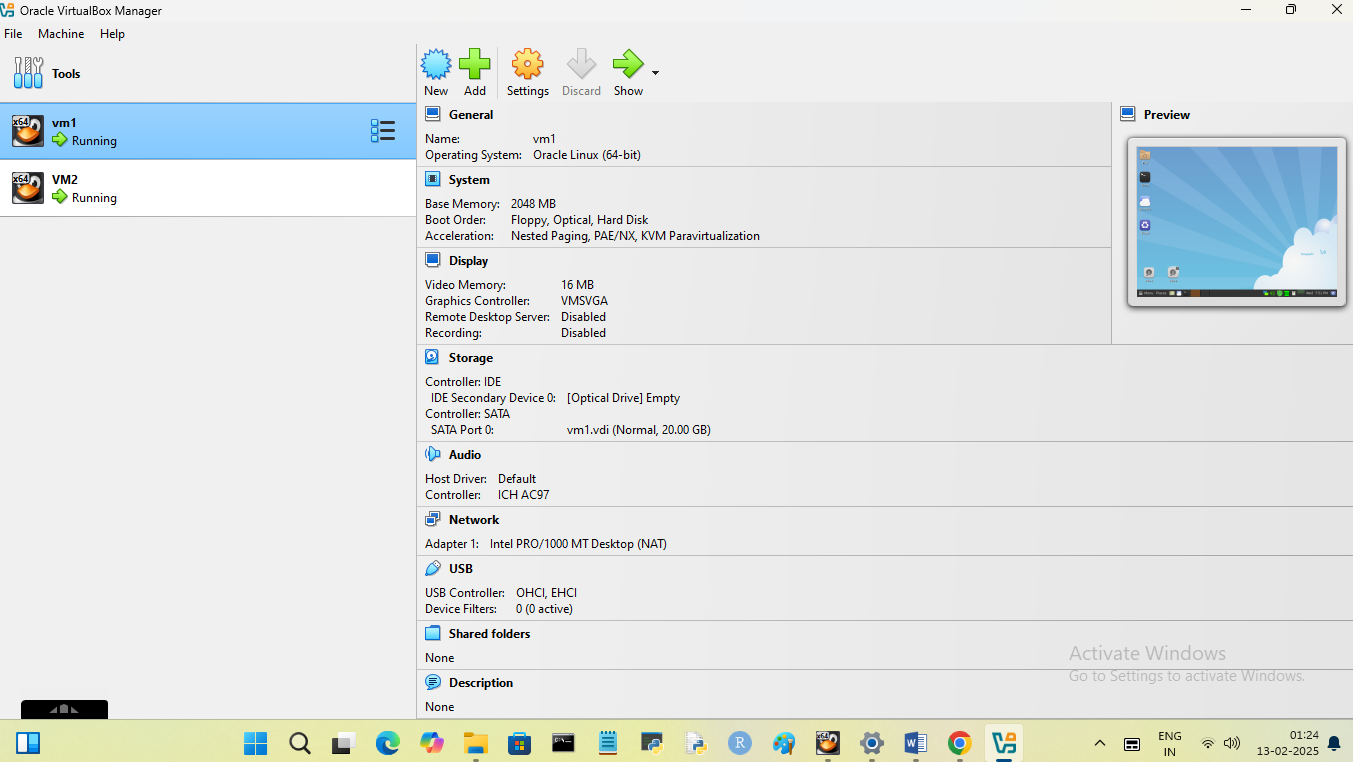
Click finish to create VM

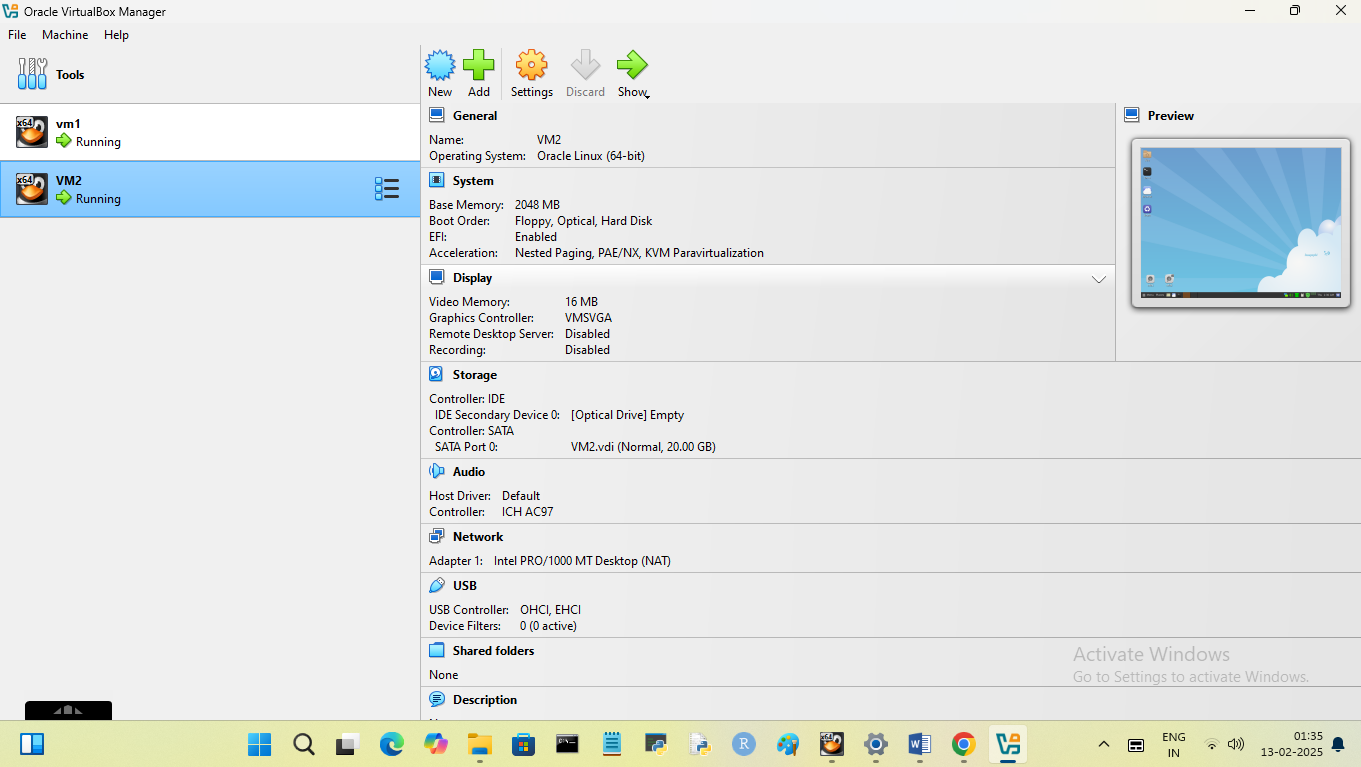
****

****

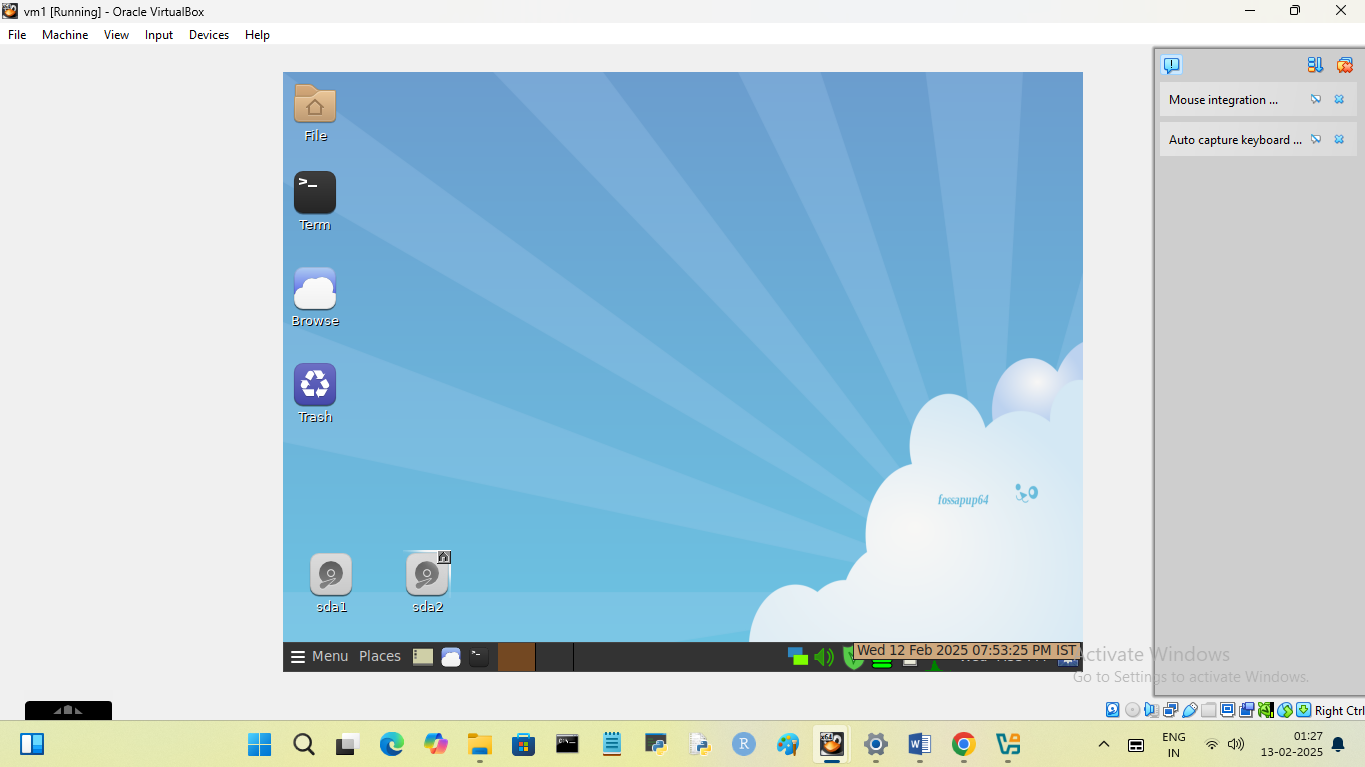
Click start to run VM

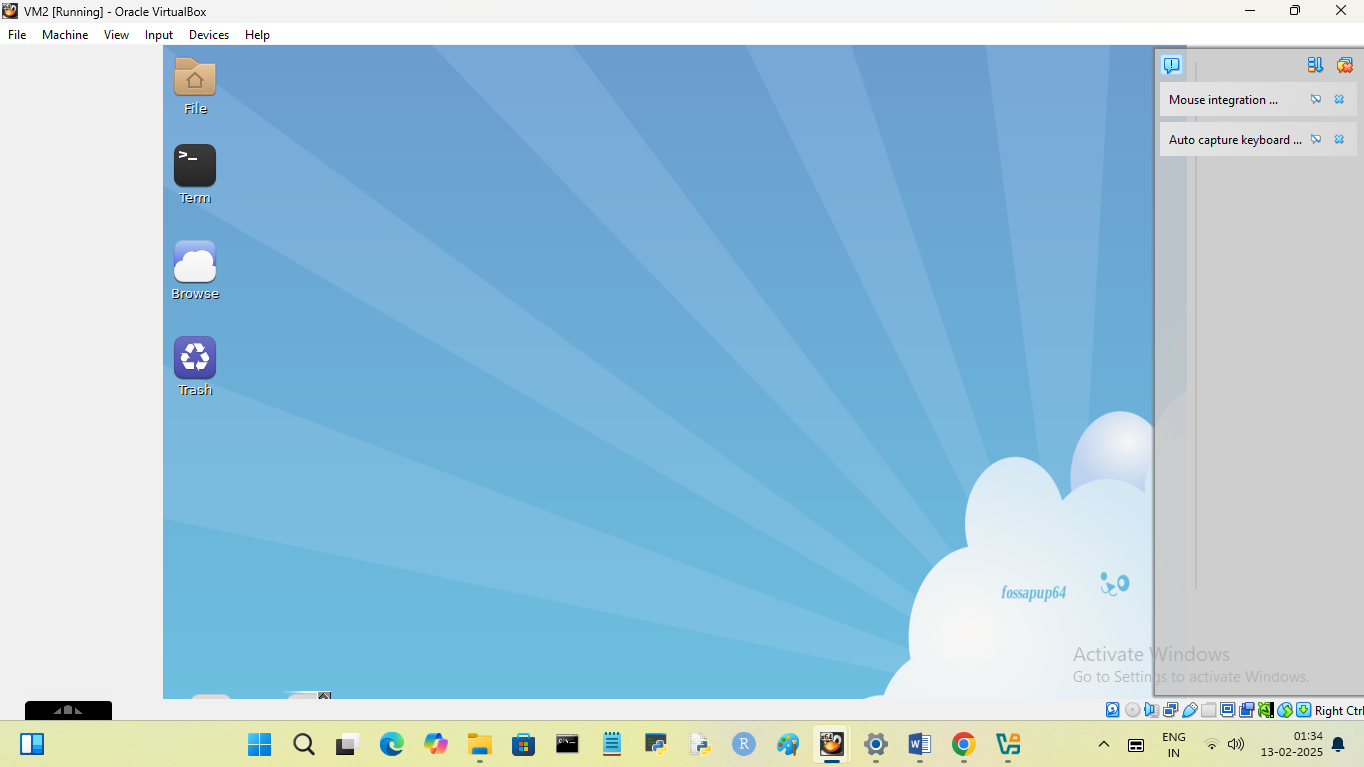
****

****

****

Change Desktop background and icon settings as required

****

****

**Manually Install pip**

**wget** <https://bootstrap.pypa.io/get-pip.py>

**python3 get-pip.py**

**Manually Install psutil**

**wget** [https://github.com/giampaolo/psutil/archive/refs/tags/release-5.9.5.tar.gz -O psutil.tar.gz](https://github.com/giampaolo/psutil/archive/refs/tags/release-5.9.5.tar.gz%20-O%20psutil.tar.gz)

**tar -xvzf psutil.tar.gz**

**cd psutil-release-5.9.5**

**python3 setup.py install**

**Run sample application:**

**App.py**

**from flask import Flask, request, jsonify**

**from flask\_sqlalchemy import SQLAlchemy**

**from werkzeug.security import generate\_password\_hash, check\_password\_hash**

**import os**

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

**# Configure the database URIs**

**db\_path = os.path.join(os.getcwd(), 'combined.db')**

**app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///' + db\_path**

**app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False**

**# Initialize the database connection**

**db = SQLAlchemy(app)**

**# User Model**

**class User(db.Model):**

**id = db.Column(db.Integer, primary\_key=True)**

**username = db.Column(db.String(80), unique=True, nullable=False)**

**password = db.Column(db.String(120), nullable=False)**

**# Year Model**

**class Year(db.Model):**

**id = db.Column(db.Integer, primary\_key=True)**

**year\_value = db.Column(db.String(4), nullable=False)**

**# ExamType Model**

**class ExamType(db.Model):**

**id = db.Column(db.Integer, primary\_key=True)**

**name = db.Column(db.String(80), nullable=False)**

**# Subject Model**

**class Subject(db.Model):**

**id = db.Column(db.Integer, primary\_key=True)**

**name = db.Column(db.String(80), nullable=False)**

**exam\_type\_id = db.Column(db.Integer, db.ForeignKey('exam\_type.id'))**

**# Keyword Model**

**class Keyword(db.Model):**

**id = db.Column(db.Integer, primary\_key=True)**

**value = db.Column(db.String(80), nullable=False)**

**subject\_id = db.Column(db.Integer, db.ForeignKey('subject.id'))**

**# Question Model**

**class Question(db.Model):**

**id = db.Column(db.Integer, primary\_key=True)**

**text = db.Column(db.String(200), nullable=False)**

**exam\_year\_id = db.Column(db.Integer, nullable=False)**

**exam\_type\_id = db.Column(db.Integer, nullable=False)**

**# Root route**

**@app.route('/')**

**def index():**

**return jsonify({'message': 'Welcome to the Monolithic API!'}), 200**

**# User Registration**

**@app.route('/register', methods=['POST'])**

**def register():**

**data = request.get\_json()**

**hashed\_password = generate\_password\_hash(data['password'], method='pbkdf2:sha256')**

**new\_user = User(username=data['username'], password=hashed\_password)**

**db.session.add(new\_user)**

**db.session.commit()**

**return jsonify({'message': 'User created'}), 201**

**# User Login**

**@app.route('/login', methods=['POST'])**

**def login():**

**data = request.get\_json()**

**user = User.query.filter\_by(username=data['username']).first()**

**if not user or not check\_password\_hash(user.password, data['password']):**

**return jsonify({'message': 'Login failed'}), 401**

**return jsonify({'message': 'Logged in successfully'}), 200**

**# Add Year**

**@app.route('/year', methods=['POST'])**

**def add\_year():**

**data = request.get\_json()**

**new\_year = Year(year\_value=data['year\_value'])**

**db.session.add(new\_year)**

**db.session.commit()**

**return jsonify({'message': 'Year added'}), 201**

**# Add Exam Type**

**@app.route('/exam-type', methods=['POST'])**

**def add\_exam\_type():**

**data = request.get\_json()**

**new\_exam\_type = ExamType(name=data['name'])**

**db.session.add(new\_exam\_type)**

**db.session.commit()**

**return jsonify({'message': 'Exam type created'}), 201**

**# Add Question**

**@app.route('/question', methods=['POST'])**

**def add\_question():**

**data = request.get\_json()**

**new\_question = Question(**

**text=data['text'],**

**exam\_year\_id=data['exam\_year\_id'],**

**exam\_type\_id=data['exam\_type\_id']**

**)**

**db.session.add(new\_question)**

**db.session.commit()**

**return jsonify({'message': 'Question added'}), 201**

**# Get all Subjects**

**@app.route('/subjects', methods=['GET'])**

**def get\_subjects():**

**subjects = Subject.query.all()**

**return jsonify([{'id': subject.id, 'name': subject.name, 'exam\_type\_id': subject.exam\_type\_id} for subject in subjects]), 200**

**# Get all Keywords**

**@app.route('/keywords', methods=['GET'])**

**def get\_keywords():**

**keywords = Keyword.query.all()**

**return jsonify([{'id': keyword.id, 'value': keyword.value, 'subject\_id': keyword.subject\_id} for keyword in keywords]), 200**

**# Categorize Question**

**@app.route('/categorize/<int:question\_id>', methods=['GET'])**

**def categorize\_question(question\_id):**

**question = Question.query.get\_or\_404(question\_id)**

**question\_text = question.text**

**subjects = Subject.query.all()**

**keywords = Keyword.query.all()**

**categories = []**

**for subject in subjects:**

**if subject.name.lower() in question\_text.lower():**

**category\_info = {**

**'subject': subject.name,**

**'keywords': []**

**}**

**related\_keywords = [kw for kw in keywords if kw.subject\_id == subject.id]**

**for keyword in related\_keywords:**

**if keyword.value.lower() in question\_text.lower():**

**category\_info['keywords'].append(keyword.value)**

**categories.append(category\_info)**

**if not categories:**

**return jsonify({'message': 'No categories found for this question'}), 404**

**return jsonify({**

**'question\_id': question\_id,**

**'question\_text': question\_text,**

**'categories': categories**

**}), 200**

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

**try:**

**# Ensure tables are created within app context**

**with app.app\_context():**

**print(f"Creating database tables in: {db\_path}")**

**db.create\_all()**

**print("Database tables created successfully in combined.db")**

**except Exception as e:**

**print(f"Error creating tables: {e}")**

**# Run the Flask application**

**app.run(debug=True, host='0.0.0.0')**

**Visualize Resource Usage with Prometheus & Grafana**

**Install Prometheus:**

pkg add Prometheus

**Edit Prometheus Config:**

Vi /etc/prometheus/prometheus.yml

global:

scrape\_interval: 5s

scrape\_configs:

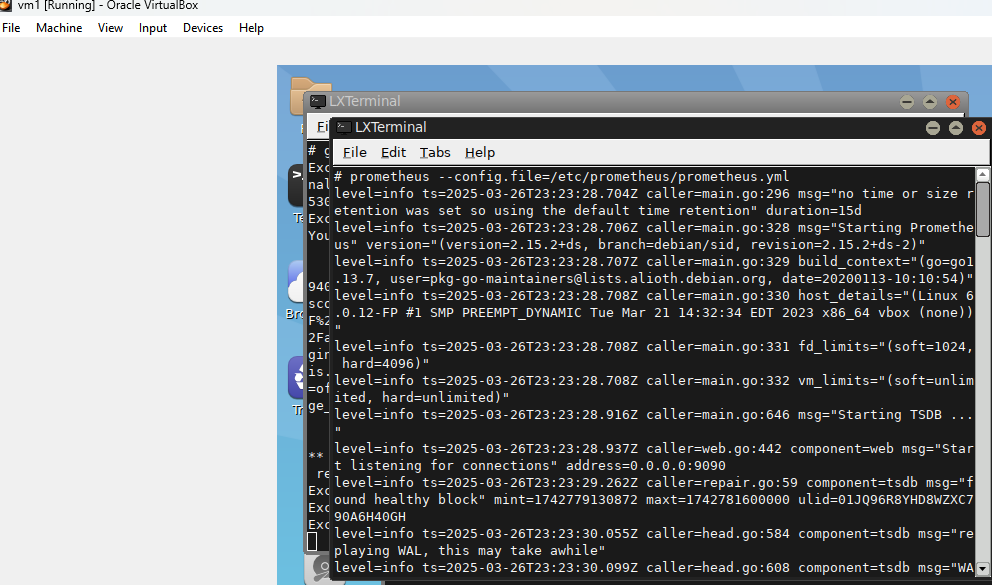
- job\_name: 'puppy\_linux\_metrics'

static\_configs:

- targets: ['localhost:9090']

**Run Prometheus:**

prometheus --config.file=/etc/prometheus/prometheus.yml



Resource usage can be monitored in localhost 9090

**Install Grafana for Visualization**

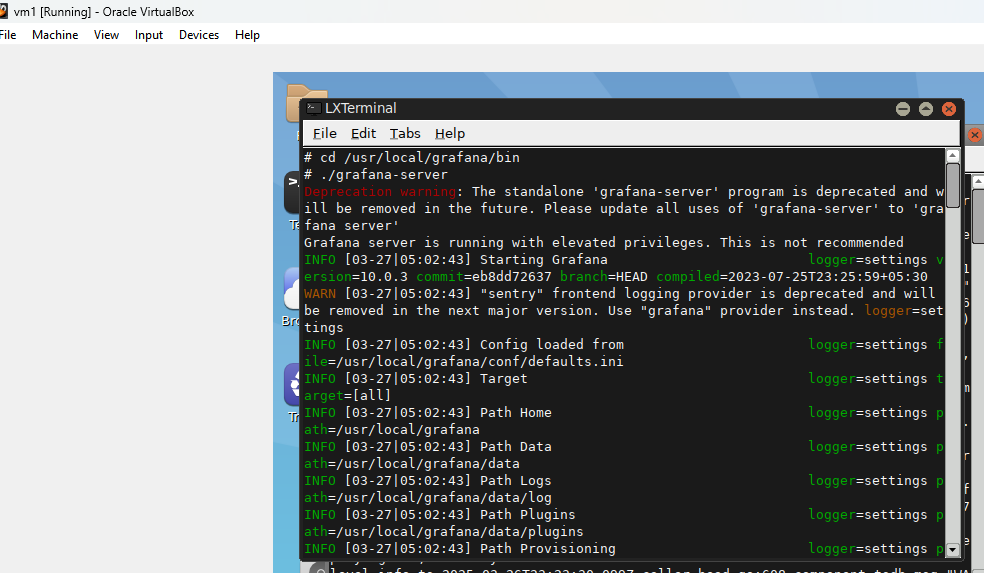
wget <https://dl.grafana.com/oss/release/grafana-10.0.3.linux-amd64.tar.gz>

tar -xvzf grafana-10.0.3.linux-amd64.tar.gz

mv grafana-10.0.3 /usr/local/grafana

cd /usr/local/grafana/bin

./grafana-server



Resource usage can be visualised in localhost 3030

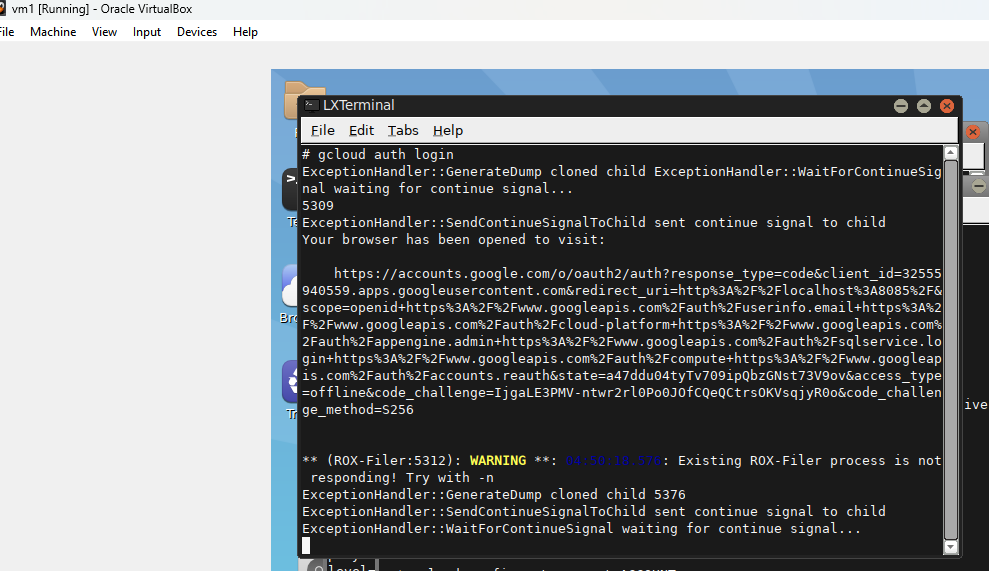
**Install Google Cloud SDK**

curl https://sdk.cloud.google.com | bash

exec -l $SHELL

gcloud init

gcloud auth login



**Create the Script:**

Vi monitor.py

import psutil

import time

import os

THRESHOLD = 75 # Auto-scaling trigger threshold

def create\_gcp\_vm():

print("Deploying new VM on GCP...")

os.system("gcloud compute instances create auto-scale-vm --machine-type=e2-medium --image-family=debian-11 --image-project=debian-cloud --zone=us-central1-a")

def check\_resources():

cpu\_usage = psutil.cpu\_percent(interval=1)

memory\_usage = psutil.virtual\_memory().percent

print(f"CPU Usage: {cpu\_usage}%")

print(f"Memory Usage: {memory\_usage}%")

with open("resource\_log.txt", "a") as log\_file:

log\_file.write(f"CPU: {cpu\_usage}%, Memory: {memory\_usage}%\n")

if cpu\_usage > THRESHOLD or memory\_usage > THRESHOLD:

print("High resource usage detected! Consider scaling up.")

create\_gcp\_vm()

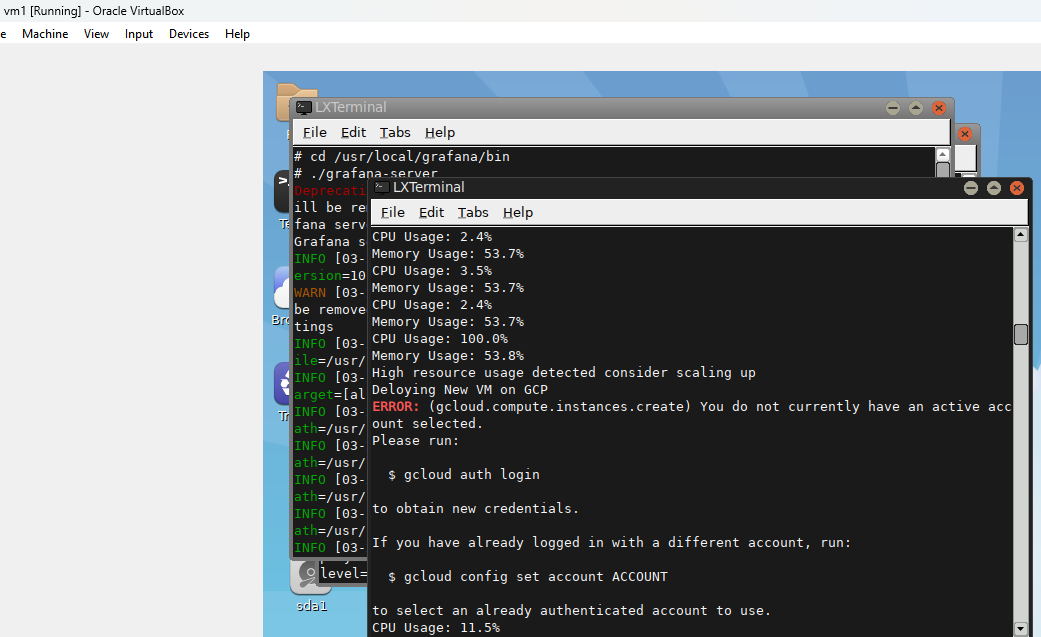
while True:

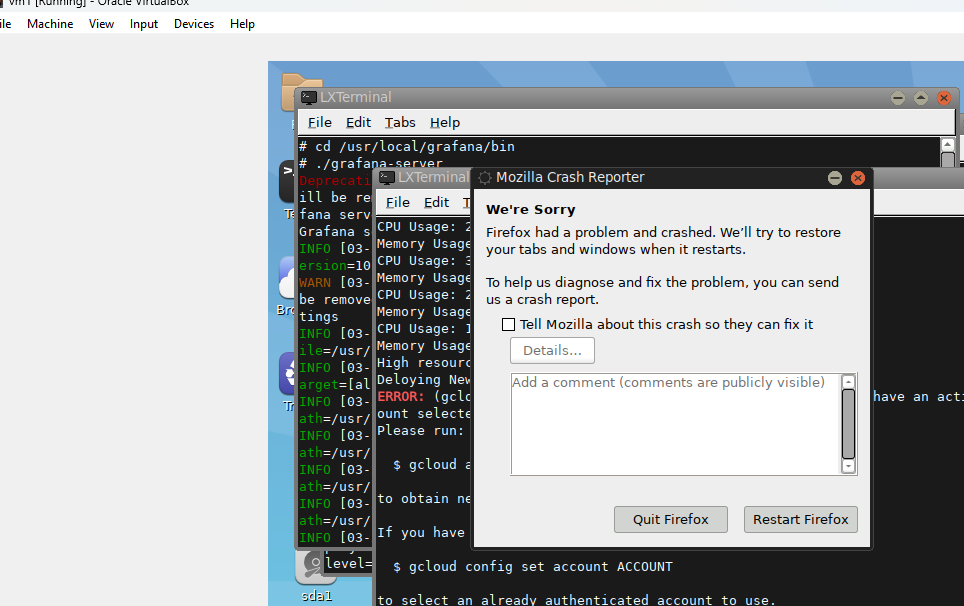
check\_resources()

time.sleep(5) # Run every 5 seconds

**Run the script**

python3 monitor.py





Unable to achieve the final step of migrating local VM to GCP due to above error.