```python
#OpenStack business model simulator. Run with "streamlit run thisfile.py".
#Written by Richard Neill (with some initial AI guidance); Free software, licensed under the GNU GPL version 3+.

import streamlit as st
import pandas as pd
import numpy as np
import plotly.graph_objects as go

# Constants
setup_months = 2
setup_servers = 8               #needed to build out the basic system.
server_price = 26100            #this includes the server's share of networking equipment.
server_annual_opex = 3100       #data centre costs.
monthly_revenue_per_server_base = 5500   #the AWS price, from benchmarking.
depreciation_years = 3
deferred_payment_interest = 1.08
deferred_payment_months = 9
admin_salary = 120000
min_admins = 6
servers_per_admin = 60
server_cpu_cores = 64
server_ram_gb = 256
server_disk_tb = 72/3  #after ceph redundancy
server_bandwidth_tb = 32
server_power_kw = 0.7
servers_per_rack = 20   #because each of these servers is quite small - we could normally double-their loading or more.
exponential_growth_2_dc_sanitycheck = 10000 * servers_per_rack  #2 DCs. A large enterprise data-centre typically has 500-5000 racks.

#Titles
st.set_page_config(page_title="OpenStack Business Model Simulator", layout="wide")
st.title("📊 OpenStack Business Model Simulator")
st.caption("For help, please scroll down to the end.")


# Sidebar inputs
st.sidebar.header("⚙ Parameters")
runtime_months = st.sidebar.slider("Simulation Duration (Months)", 12, 84, 60, 3)
st.sidebar.subheader("Business:")
initial_investment = 1000 * st.sidebar.slider("Initial Investment (£k)", 100, 5000, 1000, 50)
discount_vs_aws = 0.01 * st.sidebar.slider("Sales Price discount vs. AWS (%)", 0.0, 25.0, 10.0, 0.1)
overhead_cost_fraction = 0.01 * st.sidebar.slider("Business Overheads (%)", 5.0, 30.0, 10.0, 0.1)
st.sidebar.subheader("Marketing:")
marketing_cost_base = 1000 * st.sidebar.slider("Fixed Base Cost (£k/month)", 0,100,10,1)
marketing_cost_fraction = 0.01 * st.sidebar.slider("Variable Costs (% of sales)", 0.0, 20.0, 5.0, 0.1)
st.sidebar.subheader("Servers:")
server_lifetime_years = st.sidebar.slider("Server Lifetime (years)", 3.0, 8.0, 5.0, 0.2)
utilisation = st.sidebar.slider("Server Utilisation Fraction", 0.7, 1.0, 0.9, 0.01)
cpu_overcommit = ((st.sidebar.slider("CPU Overcommit Ratio", 1.0, 6.0, 1.8,0.05) - 1)/2)+1              #Halved, as a heuristic, since we
```

```
can't overcommit RAM.
st.sidebar.subheader("Capacity Growth:")
initial_capacity = st.sidebar.slider("Initial Capacity (servers)", setup_servers, 200, 20)      #in month 3.
monthly_growth_rate = st.sidebar.slider("Monthly Growth Rate (factor)", 1.0, 1.3, 1.10)
st.sidebar.subheader("Advanced:")
growth_flatten_month = st.sidebar.slider(f"Growth Flattens in Month\n\n({(runtime_months+1)} means never.)", 0, runtime_months,
runtime_months+1)
deferred_payments = st.sidebar.toggle("270-day deferred payments")


# Simulation setup
servers = []            #each cohort of servers (by month) has {'count':n, 'age':m} .
data = []
deferred_payment_queue = [0] * deferred_payment_months  # For deferred payments
deferred_servers_queue = [0] * deferred_payment_months
cumulative_cost = 0
cumulative_revenue = 0
cumulative_profit = 0
scrapped_servers_total = 0
cash_balance = initial_investment
bankrupt_month = None
server_purchase_cost = server_price * (deferred_payment_interest if deferred_payments else 1.0)
effective_capacity_per_server = utilisation * cpu_overcommit

for month in range(1, runtime_months + 1):
    if month ⩽ setup_months:
        required_servers = setup_servers
    else:
        months_since_sales = month - setup_months  #month since sales started.
        growth_months = min(months_since_sales, growth_flatten_month - setup_months)
        required_servers = int(np.ceil( initial_capacity * (monthly_growth_rate ** (growth_months-1))))

    for s in servers:
        s["age"] += 1
    scrapped = [s for s in servers if s["age"] ⩾ server_lifetime_years * 12]
    scrapped_servers = sum(s['count'] for s in scrapped)
    scrapped_servers_total += scrapped_servers
    servers = [s for s in servers if s["age"] < server_lifetime_years * 12]

    current_servers = sum(s['count'] for s in servers)
    new_servers = max(0, required_servers - current_servers)
    servers.append({"age": 0, "count": new_servers})
    total_servers = sum(s['count'] for s in servers)
    active_capacity = total_servers * effective_capacity_per_server

    capex_now = new_servers * server_purchase_cost
    if deferred_payments:
        deferred_payment_queue.append(capex_now)
```

```
            capex = deferred_payment_queue.pop(0)
            deferred_servers_queue.append(new_servers)
            paid_off_servers = deferred_servers_queue.pop(0)
        else:
            capex = capex_now

        opex = total_servers * (server_annual_opex / 12)
        required_admins = max(min_admins, int(np.ceil(total_servers / servers_per_admin)))
        sysadmin_cost = required_admins * (admin_salary / 12)
        revenue = active_capacity * monthly_revenue_per_server_base * (1 - discount_vs_aws)
        marketing_cost = revenue * marketing_cost_fraction + marketing_cost_base
        overhead_cost = revenue * overhead_cost_fraction
        total_cost = capex + opex + sysadmin_cost + marketing_cost + overhead_cost
        if month ⩽ setup_months:    #initial revenue is zero, but we still use it to model overheads.
            revenue = 0

        profit = revenue - total_cost

        cumulative_cost += total_cost
        cumulative_revenue += revenue
        cumulative_profit += profit
        cash_balance += revenue - total_cost

        if cash_balance < 0 and bankrupt_month is None:
            bankrupt_month = month

        server_asset_value = 0                                          #the value of all the servers (even if we don't own them yet,
    because of deferred payments.)
        for s in servers:
             age = s['age']
             if age < depreciation_years * 12:
                 remaining = 1 - (age / (depreciation_years * 12))          #We are depreciating monthly (not yearly).
                 server_asset_value += server_price * remaining * s['count']

        unpaid_servers = sum(deferred_servers_queue) if deferred_payments else 0
        unpaid_value = sum(deferred_payment_queue) if deferred_payments else 0
        business_value = cash_balance + server_asset_value - unpaid_value

        data.append({
            "Month": month,
            "Total Servers": total_servers,
            "Effective Active Servers": active_capacity,
            "Unpaid Servers": unpaid_servers,
            "Scrapped Servers": scrapped_servers_total,
            "System Admins": required_admins,
            "Monthly System Admins Cost (£M)": sysadmin_cost / 1_000_000,
            "Monthly Marketing Costs (£M)": marketing_cost / 1_000_000,
            "Monthly Overheads Costs (£M)": overhead_cost / 1_000_000,
```

```
        "Monthly Server Costs (£M)": capex / 1_000_000,
        "Monthly Unpaid Server Costs (£M)":  (capex_now if deferred_payments else 0) / 1_000_000,
        "Monthly Cost (£M)": total_cost / 1_000_000,
        "Monthly Revenue (£M)": revenue / 1_000_000,
        "Cumulative Expenditure (£M)": cumulative_cost / 1_000_000,
        "Cumulative Revenue (£M)": cumulative_revenue / 1_000_000,
        "Cumulative Profit (£M)": cumulative_profit / 1_000_000,
        "Asset Value (£M)": server_asset_value / 1_000_000,
        "Creditors (£M)": unpaid_value / 1_000_000,
        "Cash Balance (£M)": cash_balance / 1_000_000,
        "Business Value (£M)": business_value / 1_000_000,
    })

# DataFrame
df = pd.DataFrame(data)

breakeven_month = df[df["Cumulative Profit (£M)"] ⩾ 0].head(1)["Month"].values
cashflow_positive_month = df[df["Monthly Revenue (£M)"] ⩾ df["Monthly Cost (£M)"]].head(1)["Month"].values


# Plot 1
st.subheader("📈 Cumulative Financials")
fig1 = go.Figure()
fig1.add_trace(go.Scatter(x=df["Month"], y=df["Cumulative Expenditure (£M)"], name="Expenditure", line=dict(color="red",dash="dot")))
fig1.add_trace(go.Scatter(x=df["Month"], y=df["Cumulative Revenue (£M)"], name="Revenue", line=dict(color="blue",dash="dot")))
fig1.add_trace(go.Scatter(x=df["Month"], y=df["Cumulative Profit (£M)"], name="Profit", line=dict(color="green",dash="solid")))
fig1.add_trace(go.Scatter(x=df["Month"], y=df["Cash Balance (£M)"], name="Cash", line=dict(color="purple",dash="dash",width=1)))
fig1.add_trace(go.Scatter(x=df["Month"], y=df["Asset Value (£M)"], name="Assets", line=dict(color="orange",dash="dash",width=1)))
if deferred_payments:
    fig1.add_trace(go.Scatter(x=df["Month"], y=-df["Creditors (£M)"], name="Debts", line=dict(color="red",dash="dash",width=1)))
if breakeven_month.size:
    fig1.add_vline(x=breakeven_month[0], line_dash="dash", line_color="gray", annotation_text="Break-even")
if bankrupt_month:
    fig1.add_vrect(x0=bankrupt_month, x1=runtime_months, fillcolor="red", opacity=0.1, line_width=0)
    st.markdown(f"Red zone shows that the business went bankrupt in month {bankrupt_month}. (Try increasing the initial investment).")
fig1.update_layout(hovermode="x unified", xaxis_title="Month", yaxis_title="£M")
st.plotly_chart(fig1, use_container_width=True)
st.caption("")  #vertical space

# Plot 2
st.subheader("📈 Monthly Revenue vs. Cost (P&amp;amp;L)")
fig2 = go.Figure()
fig2.add_trace(go.Scatter(x=df["Month"], y=df["Monthly Revenue (£M)"], name="Revenue", line=dict(color="blue")))
fig2.add_trace(go.Scatter(x=df["Month"], y=df["Monthly Cost (£M)"], name="Cost", line=dict(color="red")))
if cashflow_positive_month.size:
    fig2.add_vline(x=cashflow_positive_month[0], line_dash="dash", line_color="gray", annotation_text="Cashflow+")
fig2.update_layout(hovermode="x unified", xaxis_title="Month", yaxis_title="£M")
st.plotly_chart(fig2, use_container_width=True)
```

```
st.caption("")


# Plot 3
st.subheader("📈 Monthly Cost breakdown by category)")
fig3 = go.Figure()
fig3.add_trace(go.Scatter(x=df["Month"], y=df["Monthly System Admins Cost (£M)"], name="System Admins", line=dict(color="blue")))
fig3.add_trace(go.Scatter(x=df["Month"], y=df["Monthly Overheads Costs (£M)"], name="Overheads", line=dict(color="purple")))
fig3.add_trace(go.Scatter(x=df["Month"], y=df["Monthly Marketing Costs (£M)"], name="Marketing", line=dict(color="orange")))
fig3.add_trace(go.Scatter(x=df["Month"], y=df["Monthly Server Costs (£M)"], name="Servers", line=dict(color="green")))
if deferred_payments:
    fig3.add_trace(go.Scatter(x=df["Month"], y=df["Monthly Unpaid Server Costs (£M)"], name="Unpaid Servers",
line=dict(color="red",dash="dot")))
fig3.update_layout(hovermode="x unified", xaxis_title="Month", yaxis_title="£M")
st.plotly_chart(fig3, use_container_width=True)
st.caption("")


# Plot 4
st.subheader("📈 Business Value (Balance)")
fig4 = go.Figure()
fig4.add_trace(go.Scatter(x=df["Month"], y=df["Business Value (£M)"], name="Business Value", line=dict(color="green")))
# fig4.add_trace(go.Scatter(x=df["Month"], y=df["Cumulative Profit (£M)"], name="Cumulative Profit", line=dict(dash="dot")))
fig4.add_trace(go.Scatter(x=df["Month"], y=df["Asset Value (£M)"], name="Asset Value", line=dict(color="orange",dash="dot")))
fig4.add_trace(go.Scatter(x=df["Month"], y=df["Cash Balance (£M)"], name="Cash Balance", line=dict(color="purple",dash="dot")))
if deferred_payments:
    fig4.add_trace(go.Scatter(x=df["Month"], y=-df["Creditors (£M)"], name="Creditors (Unpaid Capex)", line=dict(color="red",dash="dot")))
fig4.add_trace(go.Scatter(x=df["Month"], y=[initial_investment / 1_000_000]*len(df), name="Initial Investment",
line=dict(color="grey",dash="dot",width=1)))
fig4.update_layout(hovermode="x unified", xaxis_title="Month", yaxis_title="£M")
st.plotly_chart(fig4, use_container_width=True)
st.caption("")

# Plot 5
st.subheader("📈 Server Fleet")
st.caption(f"""
* **Effective** servers vs. **Physical** servers accounts for utilisation-fraction and CPU-overcommit.
* **Unpaid** servers are the servers we don't own yet{", because of deferred payments" if deferred_payments else ". (deferred payments are
not enabled; this is always **zero**)"}.""")
fig5 = go.Figure()
fig5.add_trace(go.Scatter(x=df["Month"], y=df["Total Servers"], name="Physical Servers", line=dict(color="green")))
fig5.add_trace(go.Scatter(x=df["Month"], y=df["Effective Active Servers"], name="Effective Servers", line=dict(color="light blue")))
if deferred_payments:
    fig5.add_trace(go.Scatter(x=df["Month"], y=df["Unpaid Servers"], name="Unpaid Servers", line=dict(color="red",dash="dot")))
fig5.add_trace(go.Scatter(x=df["Month"], y=df["Scrapped Servers"], name="Scrapped Servers", line=dict(color="black")))
fig5.update_layout(hovermode="x unified", xaxis_title="Month", yaxis_title="Server Count")
st.plotly_chart(fig5, use_container_width=True)
```

```
st.caption("")


# Plot 6
st.subheader("📈 Provisioned Capacity")
st.caption(f"""
* This shows the physical resource we have. Log scale. Storage is triple-redundant.
* By the end (month {runtime_months}), we have {round(server_cpu_cores * df.iloc[-1]["Total Servers"])} CPUs, {round(server_ram_gb/1024 *
df.iloc[-1]["Total Servers"],1)} TB of RAM, and {round(server_disk_tb/1024 * df.iloc[-1]["Total Servers"],2)} PB of (usable) Disk.
* This consumes {round(server_bandwidth_tb * df.iloc[-1]["Total Servers"],1)} TB/month of bandwidth,  and {round(server_power_kw/1000 *
df.iloc[-1]["Total Servers"],2)} MW of electricity.
  It needs approximately {round(np.ceil(df.iloc[-1]["Total Servers"] / servers_per_rack))} racks in the data-centre.""")
fig6 = go.Figure()
fig6.add_trace(go.Scatter(x=df["Month"], y=server_cpu_cores * df["Total Servers"], name="CPU cores", line=dict(color="lightgreen")))
fig6.add_trace(go.Scatter(x=df["Month"], y=server_ram_gb * df["Total Servers"], name="RAM (GB)", line=dict(color="lightblue")))
fig6.add_trace(go.Scatter(x=df["Month"], y=server_disk_tb* df["Total Servers"], name="Disk (TB)", line=dict(color="brown")))
fig6.add_trace(go.Scatter(x=df["Month"], y=server_bandwidth_tb* df["Total Servers"], name="Bandwidth (TB)", line=dict(color="orange")))
fig6.add_trace(go.Scatter(x=df["Month"], y=server_power_kw* df["Total Servers"], name="Power (kW)", line=dict(color="red")))
fig6.update_layout(hovermode="x unified", xaxis_title="Month", yaxis_title="Resource", yaxis_type="log")
st.plotly_chart(fig6, use_container_width=True)
st.caption("")

# Summary
st.subheader("🔢 Summary")
final_value = df.iloc[-1]["Business Value (£M)"]
return_on_investment = (final_value * 1_000_000 - initial_investment) / initial_investment
return_on_investment_annualised = ((final_value * 1_000_000  / initial_investment) ** (12/runtime_months)) - 1
yearly_growth = (monthly_growth_rate ** 12 - 1) * 100
st.markdown(f"**Final Business Asset Value:** £{final_value:.2f}M")
st.markdown(f"**Return on Investment:** {return_on_investment * 100.0:.1f}%, over {runtime_months} months.")
st.markdown(f"**Annualised ROI:** {return_on_investment_annualised * 100.0:.1f}%")
st.markdown(f"**Cashflow Positive Month:** {int(cashflow_positive_month[0]) if cashflow_positive_month.size else 'Not reached'}")
st.markdown(f"**Break-even Month:** {int(breakeven_month[0]) if breakeven_month.size else 'Not reached'}")
st.markdown(f"**Yearly Growth Rate in Demand:** {yearly_growth:.1f}%")
if bankrupt_month:
    st.markdown(f"**💥 Cash runs out in Month {bankrupt_month}.** (try raising the initial investment)")
else:
    st.markdown("**✔ Cash never runs out.**")
unprofitable_growth = df["Cumulative Profit (£M)"].iloc[-1] < 0 and all(df["Cumulative Profit (£M)"].diff().tail(12) < 0)
if unprofitable_growth:
    st.markdown("**⚠ Growth rate too fast to yield near-term profit.** (try enabling deferred payments)")
else:
    st.markdown("**✔ Growth rate is safe.**")
if (df.iloc[-1]["Total Servers"] > exponential_growth_2_dc_sanitycheck):
    st.markdown("**✖ Too many servers for two big data-centres** (Model may break down? Perhaps exponential growth has its limits).")
st.caption("")

# Data Table
```

```
st.subheader("🏷 Monthly Data Table")
st.dataframe(df, hide_index=True,   #rename cols to fit.
             column_config={"Effective Active Servers":"Effective Servers", "Scrapped Servers":"Scrap Servers", "System Admins":"Sysadmins",
"Monthly Revenue (£M)":"Revenue (£M)", "Monthly Cost (£M)":"Cost (£M)",
                    "Cumulative Expenditure (£M)":"Cumul. Spent (£M)", "Cumulative Revenue (£M)":"Cumul. Revenue (£M)",  "Cumulative Profit
(£M)":"Cumul. Profit (£M)", "Asset Value (£M)":"Assets (£M)", "Cash Balance (£M)":"Cash (£M)",
                    "Monthly System Admins Cost (£M)": None, "Monthly Marketing Costs (£M)": None, "Monthly Overheads Costs (£M)": None,
"Monthly Server Costs (£M)": None, "Monthly Unpaid Server Costs (£M)":  None, } )

# Footnote
st.caption("")
st.subheader("Footnotes")
st.caption(f"""
* The model excludes VAT, and is pre-tax. It and works in 2025-pounds (so we can ignore inflation).
* Financing costs and premises for the team are included within the {(100*overhead_cost_fraction)}% overhead.
* Setup (pre-sales) takes {setup_months} months, with the team of {min_admins} sysadmins, and needs {setup_servers} servers.
* Growth is then modeled as exponential (initial capacity in month {(setup_months+1)}; then monthly compound growth), followed optionally by
flattening out.
* The demand-model is modeled as smooth, ignoring real-world granularity and customer-churn.
* Consultancy sales are not modeled - this would be extra revenue (and cost), to provide applications and support.
* A server is a "standardised unit of compute power", in this case {server_cpu_cores} CPU cores, {server_ram_gb} GB RAM, {server_disk_tb} TB
(triplicated) disk, {server_bandwidth_tb} TB monthly bandwidth.
* Sysadmins provide *quorum* for 24/7 support, and initial setup. Each sysadmin can run (very-conservatively) {servers_per_admin} servers.
* The spare capacity lets us build additional services: a standard application suite, or providing bespoke services, systems and support to
customers.
* Grants, and R&D tax credits are excluded.
* Deferred payments (if enabled) mean that servers cost {round(100 * (deferred_payment_interest-1),2)}% more, paid {deferred_payment_months}
months later. (Logically identical to borrowing).
* The Final Business Asset value is calculated just as the total of the (partly-depreciated) servers + cash - debt. (It doesn't attempt to
"value the business" based on EBITDA or anything like that).
""")

#Help
st.caption("")
st.subheader("?○    Instructions and Help")
st.caption(f"""
* This dynamically models the OpenStack business. Adjust the sliders on the left, and the numbers will update.
* Reload the page to go back to the default settings.
* Best viewed on a desktop computer, with a larger monitor. (On mobile, the sidebar is hidden by default; use the chevrons (&gt;&gt;) in the
top left to show it).
* The graphs are interactive. You can zoom and pan; mouse over a trace for the values; click the legend to isolate one trace. Double-click
the graph to reset it.
* The most important parameter is the capacity-growth. By default, it is exponential, but it can be set to constant (raise the initial
capacity, set monthly growth factor = 1).
""")
```