**Install once (on your laptop):**

pip install simpy

**1) Create resources (fog nodes)**

import simpy

import random

class FogNode:

"""A fog node with limited capacity + simple metrics."""

def \_\_init\_\_(self, env, name, capacity):

self.env = env

self.name = name

self.capacity = capacity

self.resource = simpy.Resource(env, capacity=capacity)

# metrics

self.served = 0

self.wait\_sum = 0.0

self.service\_sum = 0.0

def \_\_str\_\_(self):

return f"{self.name}(cap={self.capacity})"

**2) Generate processes (user requests)**

def pick\_least\_loaded(fogs):

"""Pick the fog with the smallest current load = in\_service + queued."""

loads = [(f.resource.count + len(f.resource.queue), idx)

for idx, f in enumerate(fogs)]

loads.sort()

return fogs[loads[0][1]]

def user(env, user\_id, fogs, service\_time):

"""A user arrives, picks a fog, waits if needed, gets served, then leaves."""

arrival = env.now

fog = pick\_least\_loaded(fogs)

print(f"[{env.now:>4}] User-{user\_id} arrives → {fog.name} (load={fog.resource.count + len(fog.resource.queue)}/{fog.capacity})")

# request one slot on this fog

with fog.resource.request() as req:

yield req

start = env.now

wait = start - arrival

fog.wait\_sum += wait

fog.served += 1

print(f"[{env.now:>4}] User-{user\_id} starts on {fog.name} (wait={wait:.2f})")

# service (CPU time). here we just sleep for service\_time units

yield env.timeout(service\_time)

fog.service\_sum += service\_time

print(f"[{env.now:>4}] User-{user\_id} finishes on {fog.name} (service={service\_time:.2f})")

**3) Track waiting times & resource use**

* **Average wait per fog** = wait\_sum / served
* **Utilization** ≈ total busy time / (capacity × total simulation time).  
  Here, busy time = sum of all service times on that fog (each job uses one slot).

def fog\_metrics(fog, sim\_time):

avg\_wait = fog.wait\_sum / fog.served if fog.served else 0.0

utilization = fog.service\_sum / (fog.capacity \* sim\_time) # 0..1

return {

"fog": fog.name,

"capacity": fog.capacity,

"served": fog.served,

"avg\_wait": round(avg\_wait, 2),

"total\_service\_time": round(fog.service\_sum, 2),

"utilization": round(utilization, 2)

}

**4) Toy simulation: 5 users compete for 2 fog nodes**

Two versions below:

* **A)** Everyone arrives at time **0** (maximum contention).
* **B)** Users arrive one-by-one (staggered arrivals).

Use either A or B by commenting/uncommenting.

def run\_experiment(num\_users=5, sim\_time=30, seed=7, arrivals="all\_at\_once"):

random.seed(seed)

env = simpy.Environment()

# make two fog nodes

fogA = FogNode(env, "Fog-A", capacity=2)

fogB = FogNode(env, "Fog-B", capacity=1)

fogs = [fogA, fogB]

# ----- A) all 5 arrive at time 0 (compete immediately)

if arrivals == "all\_at\_once":

for u in range(num\_users):

service\_time = random.uniform(4, 8) # pretend CPU time

env.process(user(env, u, fogs, service\_time))

# ----- B) staggered arrivals (every 1 time unit)

if arrivals == "staggered":

def source(env, n\_users):

for u in range(n\_users):

service\_time = random.uniform(4, 8)

env.process(user(env, u, fogs, service\_time))

yield env.timeout(1) # next user arrives 1 time unit later

env.process(source(env, num\_users))

# run the sim

env.run(until=sim\_time)

# print metrics

print("\n=== METRICS ===")

for fog in fogs:

print(fog\_metrics(fog, sim\_time))

# ---- run it

if \_\_name\_\_ == "\_\_main\_\_":

# Try all-at-once to see competition

run\_experiment(num\_users=5, sim\_time=30, arrivals="all\_at\_once")

# Or try staggered arrivals:

# run\_experiment(num\_users=5, sim\_time=30, arrivals="staggered")

**What you’ll see in the console**

* Lines showing when each user **arrives**, **starts service**, and **finishes** on a fog node.
* At the end, a **metrics** block with:
  + how many users each fog served,
  + average wait per fog,
  + total service time,
  + **utilization** (how busy each fog was).