Notite licenta

Avem 2 variante:

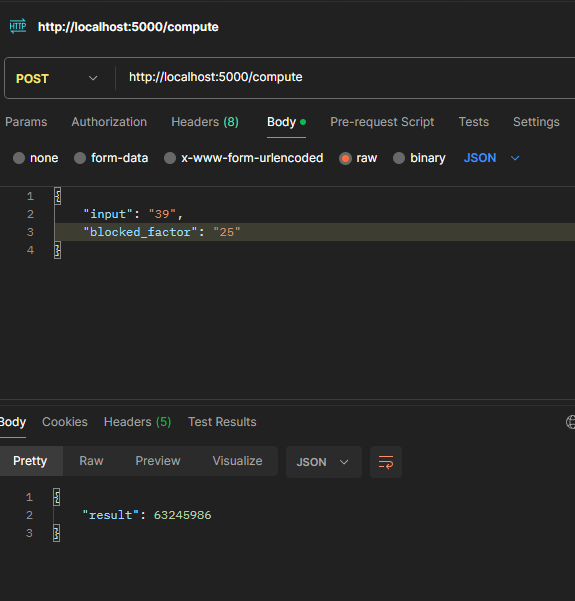
* Avand niste poduri predefinite, incercam sa asignam poduri random pe cluster si dupa sa antrenam agentul pe alte poduri deja puse pe cluster, astfel determinand un comportament al schedulerului in functie de imaginile de docker associate si alte taints ale manifesturilor
* Creem niste containere care blocheaza resurse si incercam sa antrenam pod-ul in conditii dinamice, generand astfel un comportament general al podului care nu depinde de celelalte containere deja puse pe cluster

Mergem pe a 2 a varianta pentru a facilita un comportament general si pentru a antrena modelul sa invete fiecare manifest cum s-ar comporta independent.

Avem 4 resurse pe care vrem sa le blocam:

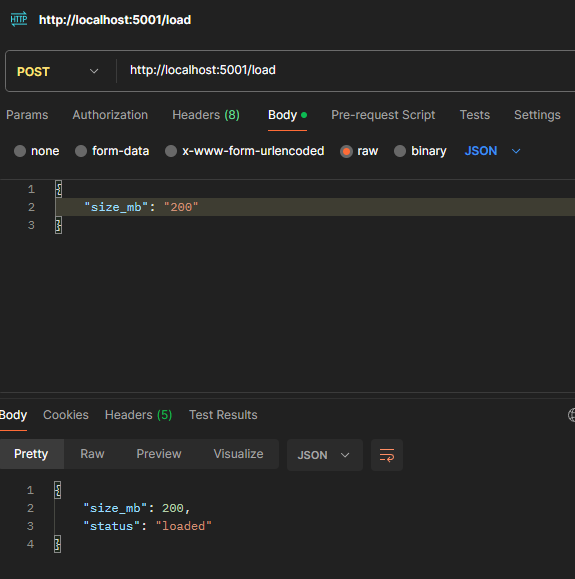
CPU – calculam Fibonacci si blocam core-uri

<http://localhost:5000/compute> POST - json cu input ( iteratii de fibonacii ) si blocked\_factor ( ce procent de cpu sa fie blocat aproximativ depinde de nr de core-uri )

 trebuie mai multe ajustari pe cluster, probleme cu docker stats sau cu docker container verificam pe cluster unde punem si monitorizari pe noduri

Am incercat sa blochez doar anumite core-uri pentru a putea simula un usage de 50% al procesorului dar metricile docker vad ca afiseaza tot 100%. Trebuie switch-uit la un cluster unde sa fie deployate imaginile alaturi de un container care sa genereze metrici de pe nod pentru a vedea mai in detaliu situatia

RAM – facem liste mari de numere float care sa ocupe spatiu determinat de noi, avand drept parametru spatiul pe care noi vrem sa il ocupam

 exista mici variatii fata de ce e ca parametru

--- acestea au fost facute pe 28.02

DISK – scrierea pe disk care s-ar face cu scrierea de numere intr-un fisier si citirea acestora in 2 thread-uri diferite din 2 fisiere diferite – lucram cu docker volume cel mai probabil

* O sa avem 3 moduri – write, read si readwrite care sa asigure ca scheduler-ul este atent la read si write si aloca pod-uri bazandu-se pe necesitatile acestuia

Am facut componenta de disk blockage dar docker metrics nu vede scrierea si citirea din fisier. De asemenea, pentru a nu consuma tot hdd-ul am facut o citire repetitiva peste un fisier de 100mb si scrierea si stergerea repetata a unui fisier de 10mb. De asemenea, pare sa blocheze cpu-ul, cred ca voi scrie aceleasi lucruri statice pentru a nu consuma cpu generand numere random

Bandwidth – trimiterea de pachete mari intr-un request spre undeva sa fac ddos ( ma gandesc la o varianta spre router )

Am scris componenta de network blockage care genereaza date random si face un stream cu ele pe un request de https catre container

TO DO:

* Terminat containere de mockuit resurse
* De facut cluster si de verificat functionalitate si de adaugat solutii de metrici
* De facut pod uri de testat against ( server python fullstack pt cpu ram si bandwidth de generat imagini, etc)
* de stabilit algoritmul de rl si de schitat environment cu metricile obtinute
* de legat environment rl de cluster prin requesturi
* de scris scheduler si de adaugat la k8 bazandu-se pe rezultatul rl
* de comparat rezultate

am creat scripturi pentru containerizarea rapida a blockerelor de resurse si pentru pushuirea acestora intr-un registry local pentru a nu le buildui de fiecare data ( economisim timp si resurse ) si am marit capacitatea wsl2 pentru a putea sustine cluster-ul de tip kind cu 3 noduri.

Initial voiam kind dar nu expune metrici low level ca un nod cu un intreg linux os

O sa folosim mai multe kernel-based virtual machine (KVM) pentru a simula niste noduri cu linux care sa ne dea si metrici low leve si kubdeadm.

In fiecare nod kvm o sa avem linux care sa ruleze pod-uri asignate de kubernetes master node. Docker registry cred ca trebuie mutat in docker hub pentru a putea permite kvm-ului sa ajunga la el.

Ca os pentru kvm-uri avem imaginea de linux folosita de cloud provideri pentru diversi agenti ( lightweight menita sa ruleze si pe bare metal).

Avem cloud init pentru configurarea serviciilor generale pentru kvm

Unul pentru master node si unul pentru worker nodes

in cadrul acestor cloud init config files avem configurari pentru os, pachete de instalat pe acesta si comenzi de rulat in acesta. Astfel, printr-o comanda de initializare a kvm-ului cu acest config file, se pot rula automat toate comenzile de pornire si configurare a cluster-ului.

Vor fi un master node si 3 worker nodes, fiecare cu 2gb ram si 2 core-uri. Pentru a facilita resursele pentru kvm uri, in cadrul configurarilor windows am marit resursele alocate catre wsl2, astfel acesta avand in total 10 core uri si 10gb ram.

Ma bat cu greu cu kvm-uri, probleme la configurare cu cloud init – initiativa canonical, metoda prin care sunt creeate toate vm-urile din cloud. Am luat log-urile din creearea vm-ului pe masina locala pentru a verifica pasii si pentru a vedea erorile

Am reusit sa configurez kvm-urile pe o interfata de internet creata odata cu ele, am instalat pachetele de administrare kubernetes si serviciul de rulare a containerelor Docker. Toate configurarile se realizeaza din script-uri de start, join, deleteContext si stop.

Trecem pe linux. Pe master node containerele de kube system fail-uiesc destul de random, semn ca nu ar avea destule resurse. Mutam pe linux pentru a putea asigura mai multe core-uri si mai mult ram.

Am trecut env de dev pe linux nativ pe laptop pt mai multe resurse libere. Avem o comunicare intre noduri, merg cum trebuie kvm-urile doar ca containerele de kubeadm pica din motive random. Am găsit ca kubeapi-server crapa din cauza etcd din loguri

W0328 22:19:17.035054 1 logging.go:55] [core] [Channel #22 SubChannel #23]grpc: addrConn.createTransport failed to connect to {Addr: "127.0.0.1:2379", ServerName: "127.0.0.1:2379", }. Err: connection error: desc = "transport: Error while dialing: dial tcp 127.0.0.1:2379: connect: connection refused"

și în etcd am găsit asta în loguri:

{"level":"info","ts":"2025-03-28T22:19:00.900839Z","caller":"osutil/interrupt\_unix.go:64","msg":"received signal; shutting down","signal":"terminated"}

Deci cineva îmi omoară etcd-ul dar cine

gata l am găsit

Mar 28 22:41:05 k8s-node-master kubelet[3234]: E0328 22:41:05.946229 3234 event.go:359] "Server rejected event (will not retry!)" err="Timeout: request did not complete within requested timeout - context deadline exceeded" event="&Event{ObjectMeta:{etcd-k8s-node-master.18311780621bdbaa kube-system 1281 0 0001-01-01 00:00:00 +0000 UTC <nil> <nil> map[] map[] [] [] []},InvolvedObject:ObjectReference{Kind:Pod,Namespace:kube-system,Name:etcd-k8s-node-master,UID:f05bda3a2f28b4df60c28cfabc1b1940,APIVersion:v1,ResourceVersion:,FieldPath:spec.containers{etcd},},Reason:Unhealthy,Message:Readiness probe failed: Get \"http://127.0.0.1:2381/readyz\": dial tcp 127.0.0.1:2381: connect: connection refused,Source:EventSource{Component:kubelet,Host:k8s-node-master,},FirstTimestamp:2025-03-28 22:19:30 +0000 UTC,LastTimestamp:2025-03-28 22:40:31.943344257 +0000 UTC m=+1291.217463830,Count:5,Type:Warning,EventTime:0001-01-01 00:00:00 +0000 UTC,Series:nil,Action:,Related:nil,ReportingController:kubelet,ReportingInstance:k8s-node-master,}"

asta am găsit în log-urile os-ului de pe master node. Deci pica proba de readiness și de asta se tot restarteaza etcd-ul și nu merge nimic

în logurile de la kubelet am găsit aceasta linie:  
Mar 31 14:05:55 k8s-node-master kubelet[3246]: E0331 14:05:55.924835 3246 event.go:368] "Unable to write event (may retry after sleeping)" err="Post \"https://192.168.122.136:6443/api/v1/namespaces/kube-system/events\": dial tcp 192.168.122.136:6443: connect: connection refused" event="&Event{ObjectMeta:{etcd-k8s-node-master.1831e840c85f4d82 kube-system 0 0001-01-01 00:00:00 +0000 UTC <nil> <nil> map[] map[] [] [] []},InvolvedObject:ObjectReference{Kind:Pod,Namespace:kube-system,Name:etcd-k8s-node-master,UID:0be9da42a461eb89b3f7e0e7f21d6ab0,APIVersion:v1,ResourceVersion:,FieldPath:spec.containers{etcd},},Reason:Killing,Message:Stopping container etcd,Source:EventSource{Component:kubelet,Host:k8s-node-master,},FirstTimestamp:2025-03-31 14:04:55.706332546 +0000 UTC m=+0.179260185,LastTimestamp:2025-03-31 14:04:55.706332546 +0000 UTC m=+0.179260185,Count:1,Type:Normal,EventTime:0001-01-01 00:00:00 +0000 UTC,Series:nil,Action:,Related:nil,ReportingController:kubelet,ReportingInstance:k8s-node-master,}"

în care practic încearcă să se noteze faptul ca s-a trimis semnalul de kill către etcd însa faptul ca etcd e mort nu lasa api-ului sa noteze și e o chestie circulara.

Am rezolvat cu etcd-ul, adaugand un delay în comenzile de startup din cloud init config sa aștept pana este ready prin endpoint. Celelalte containere continua sa faileze și acum sa vedem de ce.

Sunt probleme la PLEG: pod lifecycle event generator. Practic nu mi se sync-uiesc pod-urile sa poată fi urmărite de containerd și e puțin haos

O luam de la capăt cu creearea cluster-ului și încerc sa fac de mana manual un cluster și după încerc sa automatizez.

Pt KVM-uri trebuie dezactivat swap-ul pentru k8s.

Am renunțat la cloud config pentru a rula comenzile automat, încerc sa fac un plan de bătaie pentru script-uri și pentru fisierele modificate sa le incarc de mana în kvm-uri și sa fac script-uri care sa ruleze cu sudo în cadrul acestora.

sudo apt update

sudo apt install docker.io -y

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt update

sudo apt install kubeadm kubelet kubectl -y

sudo apt-mark hold kubeadm kubelet kubectl

**# avem kubernetes pe masini - comenzile de mai sus trb pe toate vm-urile**

**# trb sa aducem containerd.conf pe toate vm-urile - il lasam la ~**

**# script de rulat pe masina**

scp utilFiles/containerd.conf kubernetes@IP:~

scp utilFiles/kubernetes.conf kubernetes@IP:~

scp utilFiles/kubelet kubernetes@IP:~

**# la fel cr ca merge si pt niste script uri de rulat in vm**

scp utilFiles/scriptMaster

scp utilFiles/scriptWorker

**# asta pt fiecare ip**

**# inapoi pe fiecare kvm**

cp ~/containerd.conf /etc/modules-load.d/containerd.conf

sudo modprobe overlay

sudo modprobe br\_netfilter

cp ~/kubernetes.conf /etc/sysctl.d/kubernetes.conf

sudo sysctl --system

am început rescrierea script-ului de creeare a mașinilor virtuale efectuand configurarea de kubernetes și alte configurari din script-uri copiate și rulate prin ssh

am rescris script-urile de creeare a kvm-urilor, avem containere stabile de kubeadm pe master node, trebuie reparat node network manager-ul, în cazul nostru flannel

gata avem script automat de creeare a unui cluster yupy, trb puțin optimizat ca mănâncă multe resurse acum

Având cluster-ul gata, acum revenim la creearea aplicatiilor de blocare a resurselor.

O să rescriu aplicatiile din 4 aplicații separate intr un singur rest api cu flask care, în funcție de call-uri, blochează anumite resurse, și astfel prin deploy-erea unui singur pod cu aceasta aplicație pe fiecare nod din cluster, pot altera resursele disponibile de pe fiecare nod. Eliminam și overhead-ul necesar sa gestionam 4 pod-uri cu 4 aplicații separate care fiecare sa blocheze o singura resursa.

Am modificat aplicația și avem un singur server de python care primește request-uri care blochează anumite resurse. Am făcut și documentatie cu swagger pentru o testare mai ușoară

Voi avea 2 functionalitati.

Întâi vom avea antrenare care include și numele podului – modelul învața comportamentul container-ului pe care îl deployaza. Aici vom avea 2 categorii:

- la nivel de aplicație întreaga unde modelul alege după numele aplicației și știe toate componentele și cum ar interactiona intre ele

Aici, practic vrem un scheduler la nivel de proiect / aplicație care sa înțeleagă comportamentul intre componente și servicii și sa poată vedea când face scheduling fiecare nod ce pod-uri are, ce pod are de deployat și astfel sa ia alegerea

- la nivel de pod unde se învața comportamentul unui singur pod

Aici, scheduler-ul învața comportamentul individual al unui pod și astfel vede la fiecare nod ce resurse are și este fine tuned pt un anume pod

Acum încep sa construiesc aplicația, care va avea 4 componente:

- frontend : react – va folosi mainly network bandwidth și niște cpu

- backend : c# - va folosi mainly cpu și niște ram

- caching: redis – va folosi mainly ram

- db: ? Încă nu stiu – va folosi mainly i/o

Am facut schita de backend in C#, trebuie reparate niste elemente de containerizare

Voi face 2 backend-uri si 2 frontend-uri, folosind tehnologii diferite menite sa adauge complexitate in plus aplicatiei. Unul pentru clienti si firme sa incarce produse, si unul pentru firme pentru inventar. Ambele vor avea acelasi db, si site-ul de comert va valida inventarul.

Revenim sa avem o singura aplicatie care sa aiba un frontend si un backend. Pentru a putea lasa spatiu de scheduling pe noduri, nu pot avea prea multe aplicatii fiindca de abia o sa am loc de cate o instanta pentru fiecare.

Scoatem jwt auth din aplicatie fiindca o sa ne incurce la testare dupa.

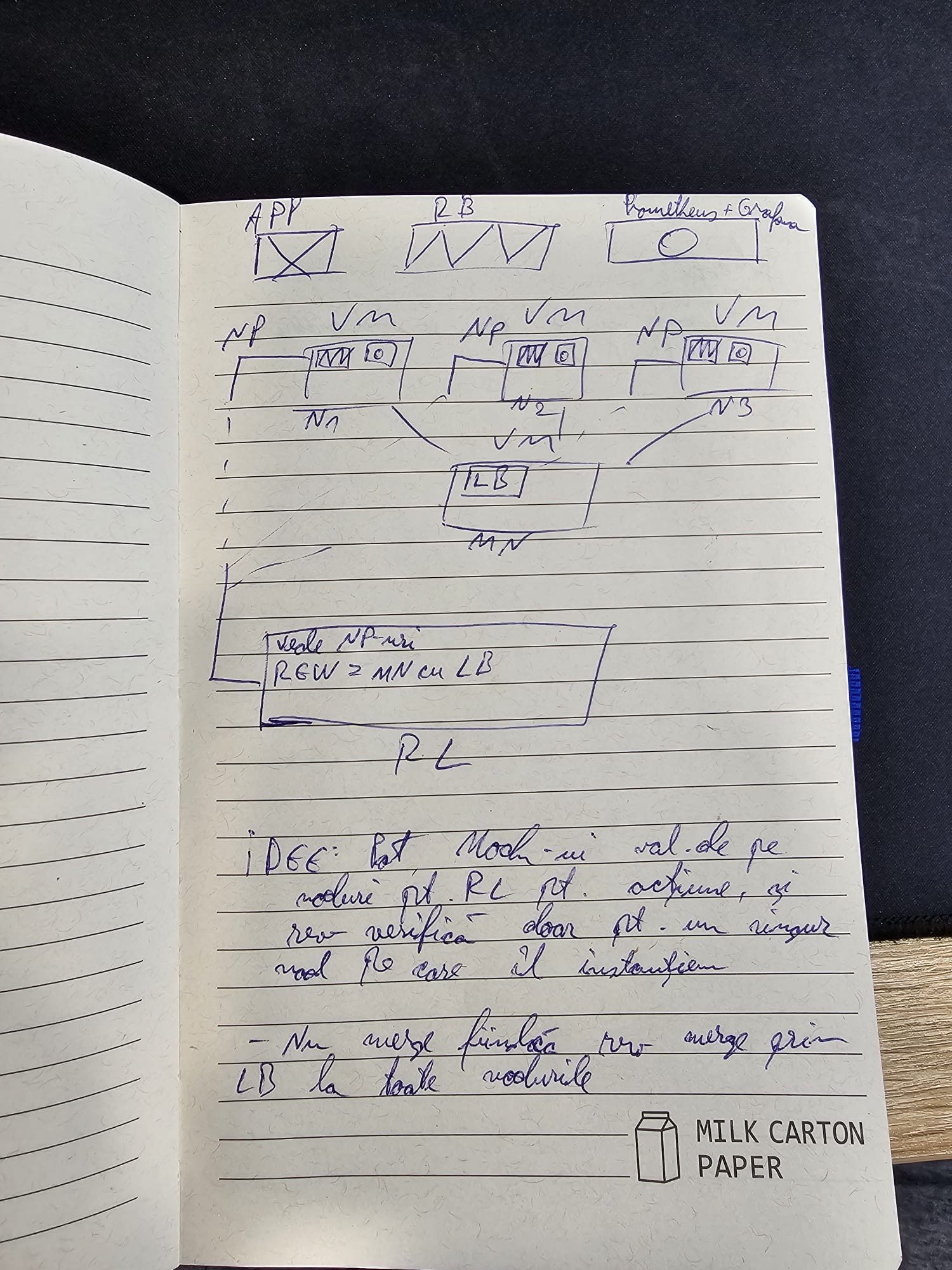
Renunțam la ideea cu mai multe componente ale unei aplicații întrucât nu o să avem puterea computationala sa o facem sa meargă. O să antrenam modelul pe câte un pod dat ca nume în state-ul modelului.

Renuntam la aplicatie complicata de testat fiindca nu ala e focus-ul si nu o sa avem resurse pe laptop sa mai si antrenam. O sa fie 2 aplicatii de demo mai simple dar tot practice.

O sa fie backend simplu in python cu flask. (lightweight)

O aplicatie va fi una unde trimit un fisier si returnez fisierul intr-o arhiva zip/rar (comprimare).

La cealalta ma mai gandesc.



Ca infrastructura, vor fi cele 3 worker nodes si master node-ul. Pe fiecare worker node va fi un pod de resource blockers si unul de prometheus. Pe master node va fi si un pod de grafana pt dashboards si vizualizare. Vom avea nevoie de metrici pt RL. Fiecare nod va avea si un nodePort pentru a putea fi accesat individual de grafana.

Alg de RL se va uita prin grafana la toate nodurile si va lua metrici pt a configura starile agentului AI, iar dupa aceea reward function-ul va avea la baza si metrici din grafana si metrici din request-uri prin LoadBalancer pentru a accentua beneficiile resimtite de utilizator.dd

În continuare, am refacut cluster-ul și am început sa configurez expunerea de metrici prin node exporter spre un server de prometheus.

Ficare nod va avea câte un pod node exporter și un pod de resource blocker. Metricile din node exporter vor fi trimise către prometheus pentru o vizualizare mai umana. Trebuie văzut daca în algoritmul de RL vor fi folosite metrici din prometheus sau vor fi extrase metricile direct din node exporter-ul fiecarui nod.

Dupa ce am pus pe fiecare nod un node exporter și am lansat un server de prometheus, am văzut ca laptop-ul nu tine atâtea cerinte. Astfel, am decis sa decentralizez nod-urile și sa le distribui pe alte doua laptopuri.

Modificam script-urile sa ruleze și remote și să lanseze câte un vm care să fie worker node, distribuind astfel munca. Sper sa meargă :))

Intrucat ideea cu mai multe laptop-uri a prezentat o provocare prin faptul ca nu putem expune vm-urile de pe fiecare laptop la LAN, am incercat si cu google cloud platform. Problema cu 3 laptop-uri este faptul ca vm-ul de pe laptopul 1 nu poate comunica cu v mul de pe laptopul 2, chiar daca laptopul 1 poate comunica cu laptopul 2.

O solutie incercata si sugerata pe internet a fost creearea unui vpn intre toate vm-urile si host machine-urile, insa nici asta nu sunt sigur ca ar functiona. Am facut fisiere de configurare dinamice cu wireguard, insa intervin multe probleme de permisiuni si de sincronizare.

Am pastrat ideea unui script central de startKVM care sa porneasca cluster-ul, insa acum acel script trebuie sa mute fisiere pe laptopurile celelate si sa faca ssh pe celelalte masini pentru a rula comenzi sau pentru a face alt ssh pentru masina virtuala.

Treburile deveneau urate, asa ca am incercat sa validez si teoria faptului ca metricile pe care mi le expun nodurile dintr-un cluster oferit de cloud provideri nu sunt destule pentru a putea ghida un algoritm de reinforcement learning pe baza lor.

Odata ce am facut un cluster pe google cloud platform, am extras metricile pe care mi le dadea node exporter si nu includeau nimic legate de consumul de curent. De asemenea, nici in fisierele de pe nod-uri nu am gasit referinte catre senzori sau metrici legate de consumul de curent. Sunt atasate metricile expuse de node exporter de pe nod in folderul cloud.

Încerc sa clarific situația sa vad daca pot expune în KVM metrici de curent. Daca da, încerc sa leg sa simulez un cluster cu un worker node pe thinkpad cu intel fiindcă am rapl. Încerc fiindcă acesta este raspunsu chat pt asta:

Why You're Seeing turbostat Fail in the KVM Guest

Inside the KVM guest (e.g., your worker3 VM), turbostat fails because:

* It accesses **MSRs (Model-Specific Registers)** directly via /dev/cpu/\*/msr
* These **are not virtualized or passed through by default** in KVM for security reasons
* So even though the **guest sees an Intel CPU** (CPUID is passed through), it cannot **read real power/energy state**, hence:

text

CopiazăEditează

msr offset 0xe2 read failed: Input/output error

In short: **you cannot access real hardware power metrics like RAPL from inside a KVM guest** without explicit passthrough (which is rarely done because of security concerns and kernel support issues).

## ✅ Where You **Can** Access Power Metrics: On the **KVM Host**

You should run turbostat, powertop, or check /sys/class/powercap/ **on the physical machine** running KVM (your "hypervisor host").

### ✅ Check on the host:

1. SSH into your KVM **host machine** (the physical Linux server)
2. Run:

bash

CopiazăEditează

sudo turbostat

You should now see output like:

yaml

CopiazăEditează

Package Core CPU Avg\_MHz Busy% Bzy\_MHz Pkg\_W Cor\_W

0 - - 340 12.00 2800 14.2 4.5

1. Check for RAPL files:

bash

CopiazăEditează

ls /sys/class/powercap/

If you see:

makefile

CopiazăEditează

intel-rapl:0 intel-rapl:0:0

You're good — the host supports RAPL.

1. View power usage over time:

bash

CopiazăEditează

cat /sys/class/powercap/intel-rapl:0/energy\_uj

sleep 1

cat /sys/class/powercap/intel-rapl:0/energy\_uj

## ⚙️ Advanced: Can I Pass RAPL to Guests?

Technically **possible**, but **not recommended** and rarely used:

* You’d need to **pass through** /dev/cpu/\*/msr via virtio or vfio
* You might expose MSR functionality using QEMU flags (-cpu host,+msr)
* Kernel in guest and host must both allow access to MSR (via msr module)

But this breaks VM abstraction and security, so it’s not standard.

## ✅ Recommendation for Power Metrics with KVM-based Clusters

If you're running a **Kubernetes cluster on KVM** and want to monitor **power usage per node**, do this:

1. Install Prometheus **Node Exporter** as a DaemonSet **only on the physical host(s)**
2. Mount /sys/class/powercap/ into the container
3. Use a custom script or exporter (e.g. rapl-exporter) to export power values
4. Collect and graph them in Prometheus/Grafana

## TL;DR

* ❌ You **cannot** read real power metrics from **inside a KVM VM**
* ✅ You **can** read them from the **physical host running the VM**
* 🔧 For Prometheus, use a custom exporter or the textfile collector on the **host**
* ⚠️ Do **not** attempt MSR passthrough unless you're doing low-level experimentation

Let me know if you want:

* A full setup to export power metrics from the host into Prometheus
* A script that logs RAPL data to disk or CSV
* A way to expose summarized power per VM (using host-side correlation)

4o

încerc sa pasez rapl pt kvm. Daca nu, voi lucra cu google cloud platform.

O idee este aceea de a vedea ce core-uri sunt folosite de kvm, și sa extragem informații despre ele de pe host machine.

Am făcut un script care expune toate metricile de curent pe care le poate lua de pe laptop, luând informații în funcție de ce core-uri sunt folosite de kvm.

Încă sunt puțin limitat de laptop sa accesez toate informațiile, trebuie schimbate niște reguli de securitate la nivel de bios.

Nvm n are suport procesorul pentru masurare per task:  
Why per-task RAPL still says “<not supported>”

Even though kernel 6.8 contains the code, Intel’s RAPL driver **still reports  
“not supported” for per-process counting on many client CPUs** (mobile  
Tiger-Lake/Comet-Lake, etc.).  
Upgrading further or rebuilding the kernel won’t change that today.

So we’ll fall back to the method that is **kernel-agnostic and works  
everywhere**:

1. Sample the **package-wide energy counter** every second.
2. Compute the VM’s **share of CPU time** during that second.
3. Attribute that fraction of the package energy to the VM.

It’s not billing-grade, but it is the best generally available on current  
hardware.

Gata, avem o varianta functionala a script-ului. Acum trebuie legate cele doua vm-uri intre mașini

RAPL = running average power limit

pentru comunicarea intre laptop-uri, am decis conectarea amândurora printr-un cablu de internet, creand astfel un bridge fizic care sa poată aloca ip-uri și mașinilor virtuale.

Avem comunicare intre laptop-uri pe interfata de cablu, acum trebuie sa configuram reteaua pentru a asigura comunicare și intre vm-uri.  
Pentru a asigura conexiunea la internet a vm-urilor, ele o să aibă doua interfete de net:  
una de bridge pentru comunicarea intre laptop-uri și vm uri

una default a libvirt pentru a putea accesa internetul

refacem script-urile sa testam puțin sa verificam conexiunea intre vm-uri

pe noua rețea:

thinkpad: 172.16.100.1

rog: 172.16.100.2

master: 172.16.100.3

worker: 172.16.100.4

gata, am realizat conectarea pe aceeași rețea a laptop-urilor și a vm-urilor. Astfel, am făcut cluster-ul de kubernetes cu un singur worker node prin intermediul caruia voi simula nodurile din algoritmul de rl. Problema era în transmiterea fisierelor de configurare către vm în partea de inițializare, partea de networking de pana acum nefiind făcută insa fiind aceeași cu cea scrisa de mine.

În momentul în care am modificat și am adaugat o noua interfata, am citit din log-uri faptul ca partea de networking nu este initializata și se facea fallback la varianta default care era folosita și pana acum. Astfel, la argumentul de cloud-init trebuia pasat separat un fisier cu network configuration.

Acum încep dezvoltarea modelului de rl.

În cadrul modelului, avem aceste caracteristici:

environment: noduri de Kubernetes „simulate”

state space: metricile aferente unui nod ( cpu usage, ram usage, network upload/download, disk read/write, current power usage) pentru toate nodurile simulate ( în situația de fata 3)

pentru state space ma gândeam și la pod name dar mergem pe ideea ca acest model este făcut special pentru aceasta imagine de docker

action space: nod-ul pe care să fie pus pod-ul

reward function: încă nu e clara dar ma interesează importanta consumului redus de curent

vom folosi un algoritm de tip deep reinforcement learning, folosind deep q network bazat pe algoritmul q learning.

Avem o problemă cu modulul de flask – primeam out of memory când făceam un pod și era din cauza modulului flask și cum era pornit. Trebuia pornit cu gunicorn pentru a putea fi compatibil cu regula kubernetes în care în fiecare pod procesul main al aplicației are pid-ul 1. De asemenea, era o eroare din cauza permisiunilor întrucât pod-ul încerca sa scrie în /data unde nu avea permisiuni și un worker node al gunicorn dadea exit care dadea exit în procesul main al gunicorn și astfel pod-ul se termina cu completed. Acum avem resource blockers activ.  
  
Am descoperit ca mai aveam erori de conexiune în cadrul pod-ului de resource blocker și din cauza unor poduri din kube-system care dadeau crash deoarece calico nu facea o rutare buna intre worker și master. Încerc sa adaug ip route pe master și worker către reteaua cluster-ului prin 172.16.100.3 (ip-ul masterului). Se pare ca Calico alegea interfata greșită de net pentru a realiza comunicarea intre noduri.

Nu reușesc sa rezolv podurile calico-node și kube-proxy pe worker node. Primesc ori completed ori crashLoopBackOff. Voi încerca sa mut tot cluster-ul, și master node și worker node, pe celălalt laptop.

Am mutat tot cluster-ul pe celălalt laptop, am automatizat creearea de vm-uri pe el insa și acolo am aceleași pod-uri care failuiesc din motive necunoscute. Ma voi muta cu un cluster pe minikube.

Am încercat sa configurez un cluster de minikube cu 2 noduri, unul de master și unul de worker, insa a intervenit aceeași problema în care mureau pod-uri de kube proxy și de calico. Astfel, am folosit minikube în configuratia de un nod cu 6 core-uri și 16gb de ram, la care am atașat doua interfete: cea de nat cu mașina host și cea de bridge cu celălalt laptop pe care vom rula modelul. Am reușit sa configurez cluster-ul și sa mut și config-ul acestuia cu token-urile specifice pe cealaltă mașina pentru a putea gestiona cluster-ul remote.

Pentru a calcula curentul folosit de kvm, voi folosi senzorul rapl la care voi aproxima în felul următor: voi vedea cat curent consuma procesorul într-o unitate de timp, și în aceeași unitate de timp vad cat timp a stat procesul meu de kvm pe procesor. Astfel, voi extrage proportional curentul consumat de kvm.

Am început sa dezvolt modelul de RL.

Voi folosi un algoritm de Deep Q learning cu deep q networks. Voi adapta env-ul creat pana acum în unul în stilul openAI gym. Am separat astfel acțiunile și datele modelului:

- avem nod-urile teoretice în care vom genera date random care vor fi după folosite la simulare

- avem cluster-ul care va reprezenta tot mediul de env pentru model

- avem resource blockers-ul care va comunica cu aplicația rest deployata pe nod

- după simularea resurselor, vom trece de la metricile simulate la metrici reale pentru a avea o corelare corecta din partea modelului a imaginii cu sistemul pe care este deployata.

Pentru a asigura acuratetea metricilor masurate, după fiecare simulare va fi resetat pod-ul de resource blockers pentru a elimina orice eroare din partea aplicației de blocare a resurselor.

De asemenea, când se alege o „acțiune” ( indicele nodului pe care va fi rulat pod-ul), după ce se aplica deployment-ul pe nod-ul simulat cu datele nodului, vor fi rulate teste dintr-un fisier jmx asociat pod-ului. Astfel, este acoperit comportamentul aplicației de model.

Pentru testare, pentru a usura structura aplicației și funcționarea scriptului de python, vom avea un container care va primi un fisier de jmeter și va putea porni rularea unui test sau opri rularea acestuia.