

---

This is the **published version** of the bachelor thesis:

Robles Fabregat, Xavier. *Disseny i desplegament d'una arquitectura cloud per a executar la simulació de Montecarlo*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

---

This version is available at <https://ddd.uab.cat/record/334453>

under the terms of the  license.

# Disseny i desplegament d'una arquitectura Cloud per a executar la simulació De Montecarlo

Xavier Robles Fabregat

29 de juny de 2026

## Resum

Aquest projecte té com a objectiu investigar el disseny d'una arquitectura cloud escalable i replicable orientada a l'execució de simulacions de Monte Carlo per a l'estimació aproximada del MTBF (Mean Time Between Failures). Es pretén analitzar els requisits tècnics, els patrons d'arquitectura distribuïda i els mecanismes d'escalabilitat necessaris per optimitzar el rendiment computacional i el cost.

A més, s'estudiarà el desplegament automatitzat de la infraestructura mitjançant Infrastructure as Code (IaC), amb l'objectiu de garantir la traçabilitat, la reproductibilitat i la seguretat en entorns cloud. Finalment, es validarà el funcionament de la infraestructura mitjançant l'execució dels processos de Terraform.

## Paraules clau

Terraform, AWS, Monte Carlo, MTBF, Python, AWS Batch, AWS ECS, AWS Fargate

## Abstract

This project aims to investigate the design of a scalable and replicable cloud architecture oriented to the execution of Monte Carlo simulations for the approximate estimation of MTBF (Mean Time Between Failures). The goal is to analyze the technical requirements, distributed architecture patterns, and scalability mechanisms required to optimize computational performance and cost.

In addition, the automated deployment of the infrastructure will be studied using Infrastructure as Code (IaC), with the aim of ensuring traceability, reproducibility, and security in cloud environments. Finally, the correct operation of the infrastructure will be validated through the execution of Terraform processes.

## Index Terms

Terraform, AWS, Monte Carlo, MTBF, Python, AWS Batch, AWS ECS, AWS Fargate



Juny de 2026, Escola d'Enginyeria (UAB)

fallada. No obstant això, determinar el MTBF d'un sistema, microcontrolador o component no és una tasca senzilla, ja que depèn de variables altament fluctuants, com la temperatura d'operació, el desgast físic, errors de configuració entre altres.

## 1 INTRODUCCIÓ

En l'actualitat, la fiabilitat és un factor determinant, especialment en aplicacions crítiques com l'automoció, l'aeroespacial o la indústria tecnològica. Una de les mètriques clau per mesurar aquesta fiabilitat és el MTBF (Mean Time Between Failures) [1][2][19], el qual és una mètrica de fiabilitat que indica el temps promig esperat en què un sistema o component funcioni correctament abans de produir-se una

Per predir aquest valor amb precisió, es pot utilitzar la **Simulació de Monte Carlo** [6][12], la qual permet estimar els possibles comportaments d'un sistema sota condicions d'incertesa mitjançant la generació de múltiples escenaris probabilístics. Aquesta càrrega de treball, per la seva naturalesa iterativa i massiva, requereix una infraestructura de computació d'alt rendiment que pugui escalar sota demanda.

- E-mail de contacte: Xavier.Robles@autonoma.cat
- Menció realitzada: Enginyeria de Computadors
- Treball tutoritzat per: Jordi Castilla Miro (Departament de microelectrònica i sistemes electrònics)
- Curs 2025/26

## 2 STATE OF THE ART

### 2.1 El paradigma de la computació al núvol

En l'actualitat, una gran part de les infraestructures tecnològiques modernes es basen en l'ús de serveis cloud, un model que permet accedir a recursos computacionals com càlcul, emmagatzematge, xarxa i serveis gestionats a través d'Internet. Al llarg dels anys, les infraestructures han evolucionat des d'un model basat exclusivament en centres de dades on-premise, on les organitzacions gestionaven íntegrament el seu propi maquinari, fins a un paradigma en què els recursos poden consumir-se com a servei a través d'Internet. Aquest enfocament elimina la necessitat de disposar d'infraestructura física pròpia i facilita l'escalabilitat sota demanda, la flexibilitat operativa i un model de pagament basat en l'ús real dels recursos.

Dins d'aquest context, actualment existeixen diversos proveïdors de serveis cloud que ofereixen plataformes amb característiques i serveis similars, com ara Amazon Web Services (AWS), Microsoft Azure o Google Cloud Platform (GCP). Entre tots ells, Amazon Web Services (AWS) s'ha consolidat com un dels principals proveïdors de serveis cloud, oferint una àmplia gamma de solucions que abasten des de recursos bàsics de computació fins a serveis avançats d'orquestració, automatització i emmagatzematge distribuït

### 2.2 L'evolució cap a la Infraestructura com a Codi (IaC)

La tecnologia actual ha evolucionat des de l'ús de scripts d'automatització imperatius (com Bash o Python), que requereixen especificar pas a pas les accions a executar, cap a enfocaments declaratius, en els quals es defineix l'estat final desitjat de la infraestructura i el sistema s'encarrega de determinar com assolir-lo. Terraform [11] destaca com l'eina de referència per la seva capacitat de mantenir un "fitxer d'estat" (state file), que permet a l'enginyer conèixer en tot moment la diferència entre la infraestructura real i la desitjada. Aquesta traçabilitat és crítica en entorns de simulació on els recursos (CPU/RAM) han de ser configurats amb precisió.

### 2.3 Orquestració de Recursos al cloud amb AWS Batch i AWS ECS

Dins l'ecosistema d'AWS, les càrregues de treball distribuïdes es poden implementar mitjançant serveis com AWS Batch i Amazon ECS [12], que constitueixen solucions àmpliament utilitzades per a l'orquestració de recursos en entorns cloud. AWS Batch és un servei gestionat orientat a l'execució de càrregues de treball batch que abstrau part de la infraestructura subjacent, utilitzant Amazon ECS com a capa d'execució i recursos d'Amazon EC2 o AWS Fargate per al còmput. Aquest enfocament permet centrar el disseny de l'arquitectura en la definició de job queues i compute environments, que escalen automàticament en funció de la demanda, fet especialment adequat per a algorismes paral·lels com Montecarlo. Alternativament, Amazon ECS permet dissenyar arquitectures basades en contenidors, amb un major control sobre l'orquestració de les càrregues de treball. Aquest servei gestiona l'execució de tasks dins dels

clústers de contenidors, i permet seleccionar el motor de còmput subjacent, ja siguin instàncies EC2 o AWS Fargate, oferint flexibilitat en la gestió de recursos.

## 3 OBJECTIUS

Aquest Treball de Final de Grau proposa el disseny i desplegament d'una infraestructura automatitzada a Amazon Web Services (AWS) [4] utilitzant Terraform [13]. Mitjançant el servei AWS Batch [10] i AWS ECS, s'orquestraran les simulacions de fiabilitat de manera eficient, permetent obtenir càlculs de MTBF precisos en un entorn totalment definit com a codi (IaC), garantint així la reproductibilitat i l'optimització de recursos.

La infraestructura s'ha pensat com una solució robusta que prioritza l'eficiència operativa i de costos. Les seves característiques principals són:

- **Reproductibilitat i Consistència:** Gràcies a l'ús de Terraform, tota la infraestructura està definida de forma declarativa. Això garanteix que l'entorn es pugui destruir i tornar a aixecar exactament igual en qualsevol moment, eliminant la configuració manual, els errors humans i els sobre costos per mantenir l'arquitectura desplegada.
- **Escalabilitat Elàstica:** Els sistemes utilitzen les eines que els serveis d'AWS ofereixen per a gestionar els recursos de còmput de manera dinàmica. La infraestructura creix automàticament per absorbir la càrrega de les simulacions de Montecarlo i es contrau fins a zero quan no hi ha tasques pendents, optimitzant així l'ús del maquinari.
- **Alta Paral·lelització:** L'arquitectura està dissenyada per a l'execució de càrregues de treball *embarrassingly parallel*. Mitjançant l'orquestració de **contenidors Docker**, el sistema pot llançar milers d'iteracions de la simulació simultàniament, reduint dràsticament el temps necessari per obtenir els càlculs de MTBF.

Per assolir les característiques descrites, s'estableixen els següents objectius específics centrats en la infraestructura:

- **Dissenyar una arquitectura de xarxa segura i escalable:** Definir mitjançant Terraform una Virtual Private Cloud (VPC) que inclogui una subxarxa pública, així com els mecanismes de seguretat (IAM i Security Groups) necessaris per protegir els recursos de còmput.
- **Implementar l'orquestració de tasques amb AWS Batch:** Configurar els entorns de computació, les definicions de treballs i les cues de processament per automatitzar el cicle de vida de les simulacions de Montecarlo.
- **Automatitzar el desplegament amb eines de IaC:** Desenvolupar i modularitzar codi en Terraform per garantir que tota la infraestructura sigui replicable,

versionable i lliure d'intervencions manuals en la consola d'AWS.

- **Avaluar el rendiment i l'eficiència de la solució:** Validar que la infraestructura respon correctament a la demanda elàstica de la simulació, assegurant la correcta comunicació entre els serveis de còmput i emmagatzematge.
- **Realitzar un estudi detallat de costos operatius:**
  - Analitzar la despesa derivada de mantenir els serveis base de la infraestructura.
  - Analitzar el cost variable d'execució segons els recursos (vCPU i RAM) assignats a cada contenidor de simulació.
  - Comparar l'impacte econòmic de l'ús d'instàncies *On-Demand* davant d'instàncies *Spot* [14] per optimitzar el cost per iteració de la simulació de Montecarlo.

## 4 PLANIFICACIÓ

En aquest apartat s'explicaran tots els punts necessaris per a la planificació temporal i de tasques del projecte. En el diagrama de la figura 1 s'aprecia com s'han planificat totes les tasques que conformen l'elaboració d'aquest treball al llarg de totes les setmanes disponibles. S'han dividit les diverses tasques que conformen l'elaboració d'aquest treball per setmanes i amb la durada de cadascuna de les tasques.

Alguns KPIs (Key Performance Indicators) que s'han pensat per a l'elaboració d'aquest treball són els següents:

- **No superar els 10€ de costos mensuals d'infraestructura:** Per a la completa elaboració de les infraestructures que es construïran en aquest treball s'ha pensat que es dissenyaran, en la mesura del possible, tenint en compte un cost màxim de 10€ mensuals en infraestructura cloud.
- **Infraestructura reproducible i reutilitzable:** Dissenyar ambdues infraestructures amb la reproduïbilitat i la reutilitzabilitat en ment. S'han de poder desplegar i usar les infraestructures que es creen fent el mínim de modificacions possibles via la consola d'AWS; les modificacions s'intentaran fer via Terraform sempre que sigui possible. Es dissenyaran les infraestructures per intentar mantenir funcionalitats iguals però adaptant-les als serveis necessaris pertinents.

## 5 DISSENY DE LA INFRAESTRUCTURA: ANÀLISI DE REQUERIMENTS

Les arquitectures que s'estan dissenyant per a l'execució de les simulacions es basen en un model de computació per lots. L'objectiu és separar la lògica de computació de la persistència de dades, garantint la seguretat en tot moment.

Els components de les arquitectures dissenyades són:

- **Xarxa segura (VPC):** Una infraestructura de xarxa basada en una única VPC amb una subxarxa pública on es despleguen tots els recursos del sistema. Tant AWS Batch com ECS/Fargate executen les càrregues de treball dins d'aquest entorn públic, igual que els serveis associats als quals s'hi accedeix mitjançant Internet Gateway.
- **Orquestració amb AWS Batch i Amazon ECS:** En el cas d'AWS Batch, el servei gestiona de manera automàtica els entorns de còmput, escalant instàncies EC2 de tipus Compute Optimized (família C), adequades per a càrregues intensives com les simulacions de Monte Carlo. Alternativament, amb Amazon ECS, les simulacions s'executen mitjançant task definitions que es desplegaran sobre AWS Fargate, permetent un control més fi de la infraestructura.
- **Persistència en S3:** Un *bucket* centralitzat per a l'entrada de paràmetres i la recollida dels resultats del MTBF, amb polítiques de cicle de vida per optimitzar costos.
- **Seguretat IAM:** Aplicació del principi de "mínim privilegi", definint rols específics per a les instàncies que executen els contenidors Docker, limitant el seu accés només al bucket S3 corresponent.
- **Cua de treballs:** Un sistema de prioritats que permeti llançar múltiples simulacions simultàniament sense saturar els recursos disponibles.
- **Logging:** Fitxers de registre actualitzats per poder veure la traça d'execució de cadascuna de les simulacions.
- **Budget:** Control sobre les despeses mensuals de la infraestructura per tal de no superar el líndiar establert pel pressupost del projecte.
- **Alerta de budget:** Sistema que s'encarrega d'alertar per correu electrònic quan s'està arribant al límit designat al pressupost del projecte.
- **Repositori d'imatges Docker:** Repositori en el qual es guarden les imatges de Docker que el sistema emprarà prèviament per a les simulacions.

## 6 DESENVOLUPAMENT

En aquest apartat s'explicaran les fases de desenvolupament realitzades durant el transcurs del projecte.

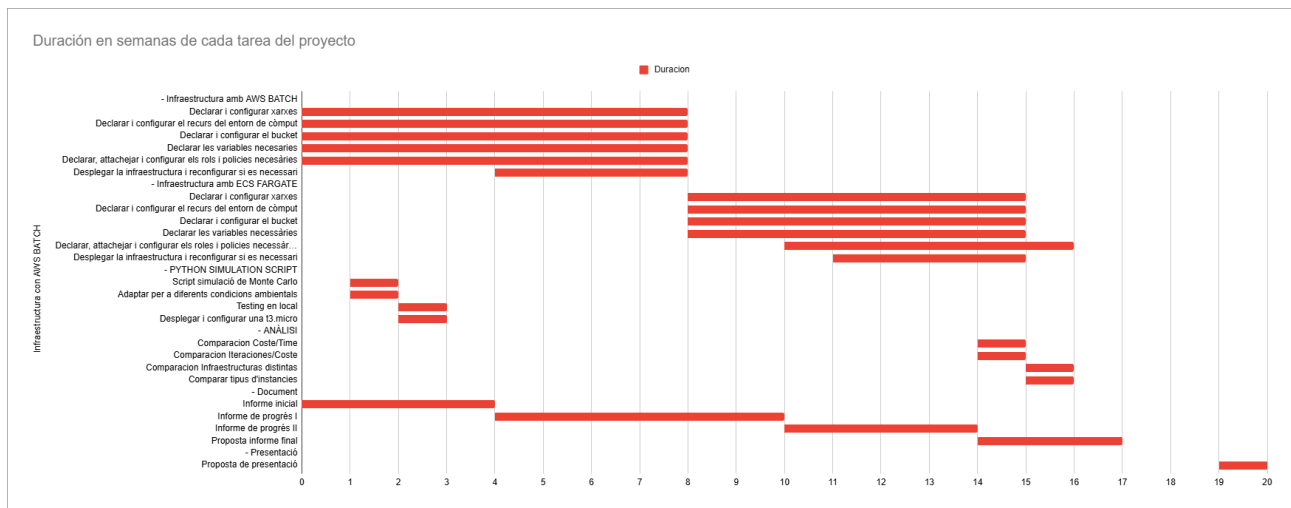


Fig. 1: Planificació del projecte de l'apartat 4

## 6.1 AWS Batch

AWS Batch és un servei gestionat d'Amazon Web Services que permet executar treballs per lots (batch jobs) de forma eficient, escalable i sense necessitat d'administrar directament la infraestructura subjacent. Automatitza l'aprovisionament de recursos de còmput en funció de les necessitats dels treballs enviats.

L'usuari defineix quins jobs vol executar, els seus requisits (CPU, memòria, temps d'execució, dependències, etc.) i AWS Batch s'encarrega de planificar i distribuir aquests treballs en els recursos disponibles, escalant automàticament quan augmenta la càrrega i reduint recursos quan ja no són necessaris.

AWS Batch permet usar màquines virtuals o AWS Fargate com a base de còmput per executar els contenidors Docker que duen a terme la tasca definida.

## 6.2 Arquitectura de la Infraestructura amb AWS Batch

Com s'ha comentat anteriorment, l'objectiu d'aquesta infraestructura és disposar d'un entorn automatitzat per a l'execució de treballs per lots a Amazon Web Services.

Aquesta infraestructura està pensada per a un escenari de càlcul intensiu i repetible, l'escenari que necessitem per executar la simulació de Montecarlo, en el qual cada execució genera resultats que s'emmagatzemen al servei pertinent d'Amazon (S3). Les traces de l'execució queden registrades en logs (CloudWatch Logs) i es permet el versionat de diferents versions del codi mitjançant un registre d'imatges de Docker propi d'Amazon (ECR).

També s'ha habilitat un servei d'alertes de control de despeses i de notificació via correu electrònic (AWS Budget i SNS, respectivament).

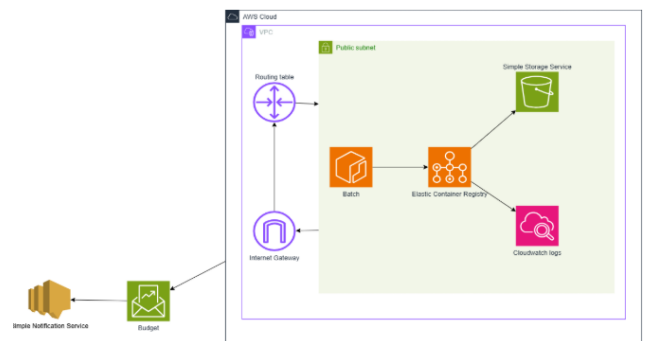


Fig. 2: Arquitectura de la infraestructura amb AWS Batch

### 6.2.1 AWS Batch: compute environment, job queue i job definition

El servei principal que conforma la infraestructura és l'esmentat AWS Batch. El nucli d'aquest servei està format per tres recursos: AWS Compute Environment, AWS Job Queues i AWS Job Definition. Aquests tres recursos són imprescindibles, ja que defineixen respectivament on s'executen els treballs, com s'organitzen i quin és el comportament exacte de cada execució.

El compute environment representa la capacitat real de processament on s'executen els contenidors. En aquest cas, s'ha configurat com un entorn *managed*, és a dir, administrat pel mateix servei d'AWS. S'ha escollit aquesta opció ja que evita gestionar manualment alguns dels recursos que són necessaris per a aquesta infraestructura (màquines virtuals d'AWS o EC2 i polítiques d'escalat o Auto Scaling Groups).

Per a base de còmput del compute environment s'ha decidit utilitzar instàncies EC2, ja que és més senzill gestionar el tipus d'instància que es vol utilitzar per a la infraestructura i evitar incórrer en costos extra. La tipologia o tipus d'instància escollida és una c4.large, la qual disposa d'un processador amb 2 vCPUs (virtual CPUs o cores), arquitectura x86\_64 i un total de 3,75 GB de memòria principal.

També s'ha definit que l'entorn de còmput tingui un `min_vcpus` (mínim de cores disponibles) de 0 vCPUs i un `max_vcpus` (màxim de cores disponibles) de 2 vCPUs, de tal forma que quan no hi hagi cap treball a executar l'entorn de còmput no crearà cap màquina virtual i, quan arribi un job, aprovisionarà els recursos definits per a la seva correcta execució. D'aquesta manera s'evita tenir màquines virtuals actives que no compleixen cap funcionalitat i es redueixen costos.

Les instàncies desplegades pel compute environment segueixen un conjunt de regles (Security Group) per tal de quedar encapsulades dins de l'entorn definit per la VPC (Virtual Private Cloud). Finalment, el compute environment fa ús d'un conjunt de permisos necessaris per interactuar amb els altres serveis que conformen la infraestructura.

La cua de treballs actua com el punt de recepció dels jobs definits. És el punt on s'envien les execucions i des d'on el servei decideix quan i sobre quin entorn s'han de llançar. En la infraestructura definida només s'ha creat una cua, ja que per a aquest cas d'ús només cal una i això simplifica la gestió i fa que l'arquitectura sigui més fàcil d'entendre i mantenir.

La cua s'ha configurat amb `state = "ENABLED"` amb una prioritat `priority = 1`. Això vol dir que està activada i preparada per rebre treballs, i que no competeix amb altres cues de prioritat diferent si aquestes existissin. Aquesta simplicitat és adequada quan només hi ha un tipus de treball o un únic flux de processament, com és el cas d'una simulació de Montecarlo. Si en el futur es volguessin introduir diferents categories de tasques, es podrien afegir més cues amb prioritats diferents per donar preferència a execucions més urgents.

La Job Definition és la part més important des del punt de vista funcional, perquè defineix exactament com s'executa cada contenidor. En aquest recurs es concreta la imatge que s'ha d'utilitzar, quants recursos s'han d'assignar, quina comanda s'ha d'executar i com s'han de registrar els logs.

La definició del job fa ús d'una imatge emmagatzemada a Amazon ECR. Aquesta decisió és clau perquè separa el codi de la infraestructura: el contenidor conté el programa de simulació i totes les dependències necessàries, mentre que AWS Batch s'encarrega únicament d'executar-lo. Aquesta separació simplifica el manteniment i fa que el sistema sigui més reproducible, ja que el mateix job es pot executar sempre amb la mateixa imatge.

Pel que fa als recursos computacionals, en la configuració apareixen `vcpus = 1` i `memory = 1024`. En canvi, per a una execució en paral·lel s'ha definit una variant del job amb més capacitat, per exemple `vcpus = 2` i `memory = 2048`, per si es vol executar una càrrega més exigent dins del mateix contenidor. Aquesta distinció és important perquè mostra que la definició del job no només descriu el programa, sinó també el perfil de recursos que necessita.

La definició del job incorpora la configuració de logs amb CloudWatch Logs. Aquí es defineix el log group `/aws/batch/job` i la regió en la qual s'emmagatzemen. Aquesta configuració és essencial per poder fer seguiment de l'execució i diagnosticar errors.

La definició també especifica un `jobRoleArn`, és a dir, un rol associat amb els permisos necessaris per a l'execució del contenidor. Aquest rol és el que permet al codi interactuar amb serveis com S3. És una peça fonamental, perquè el contenidor no hauria de tenir accés indiscriminat a l'entorn, sinó només als recursos estrictament necessaris per a la seva funció.

## 6.2.2 Repositori: Amazon ECR

Amazon Elastic Container Registry, o ECR, és el servei d'AWS que s'utilitza per emmagatzemar imatges Docker de forma privada i gestionada. En aquesta arquitectura és un component essencial, perquè AWS Batch necessita una imatge per poder crear cada contenidor de treball. ECR fa el paper de repositori centralitzat d'aquesta imatge, de manera semblant a Docker Hub, però integrat de forma nativa amb la resta de serveis d'AWS. Usant ECR es busca tenir un repositori privat d'imatges per tal de que el codi no quedi exposat públicament, el qual disposa de versionat, cosa que fa més fàcil tenir disponibles versions anteriors del codi i de la configuració del container en cas necessari. En la configuració s'ha activat `scan_on_push = true`, la qual cosa habilita l'escaneig de vulnerabilitats en pujar la imatge.

La imatge conté el codi de la simulació i les diferents dependències que necessita per a la seva execució.

## 6.2.3 Xarxa: AWS VPC, AWS subnets, Internet Gateway i routing table

Per a aquesta infraestructura no s'ha fet servir una configuració de xarxa excessivament complexa ja que el cas d'ús no ho requeria. Tot i així, hi ha quatre recursos de xarxa imprescindibles: la VPC (Virtual Private Cloud), la subxarxa, l'Internet Gateway i la taula de rutes.

La VPC és l'espai lògic aïllat on es despleguen tots els recursos de la infraestructura. És la base de tota la xarxa definida, i permet controlar l'abast de les comunicacions entre recursos. La VPC s'ha configurat amb `enable_dns_support` i `enable_dns_hostnames`, ja que AWS Batch i altres recursos que conformen la infraestructura depenen de la resolució DNS interna per funcionar correctament.

La subxarxa definida és pública, i això es reflecteix en `map_public_ip_on_launch`. Aquesta opció fa que les instàncies que s'aixequin en aquesta subxarxa puguin rebre una IP pública automàticament. Aquesta configuració facilita la connectivitat cap a serveis externs com ECR, CloudWatch o altres punts d'Internet sense haver d'afegir un NAT Gateway, servei que faria que els costos totals del projecte incrementessin considerablement.

L'Internet Gateway és el component que connecta la VPC amb Internet. Sense aquest recurs, la subxarxa no podria tenir sortida pública i la resta de recursos no podrien dur a terme la seva funcionalitat correctament.

La taula de rutes defineix el camí que ha de seguir el trànsit de la subxarxa. En este caso s'ha creat una ruta per defecte `0.0.0.0/0` cap a l'Internet Gateway.

## 6.2.4 Regles de seguretat: AWS Security Group

El security group actua com a tallafoc virtual de les instàncies del compute environment. En aquesta configuració només s'ha definit regla de sortida, i no s'han obert ports d'entrada ja que la instància només necessita comunicar-se amb ECR per a descarregar la imatge de Docker, CloudWatch per a pujar els logs i S3 per a emmagatzemar els resultats.

## 6.2.5 Emmagatzematge: Amazon S3

Amazon S3 és el servei d'emmagatzematge de dades al núvol que utilitza la infraestructura per guardar els resultats de cada execució. En aquest disseny, S3 s'utilitza com a repositori de sortides i resultats de la simulació. Això permet posteriorment analitzar els fitxers, comparar execucions i extreure conclusions. A nivell de configuració, el recurs S3 o bucket té associada una lifecycle policy que elimina automàticament els objectes amb més de 60 dies d'antiguitat. S'ha activat aquesta configuració per tal d'evitar l'acumulació de dades innecessàries i reduir el cost d'emmagatzematge.

## 6.2.6 Logging: CloudWatch Logs

Amazon CloudWatch Logs s'utilitza per registrar la sortida estàndard del job i facilitar el diagnòstic de l'execució. En la configuració de la infraestructura, el log group es crea explícitament amb el nom `/aws/batch/job` i amb una retenció de 7 dies. S'ha escollit aquest número de dies perquè és el temps suficient per a recollir les dades i per evitar acumulació de logs que incorrien en més costos.

## 6.2.7 Control de costos: AWS Budget i SNS

Per a controlar el cost mensual de la infraestructura s'ha afegit AWS Budgets, el qual és un servei gratuït d'Amazon que ser-

veix per a monitoritzar i limitar els costos mensuals d'un compte d'AWS, juntament amb Amazon SNS, o Simple Notification Service, per a l'enviament d'alertes per correu electrònic. S'han creat aquests dos recursos per a poder mantenir un registre dels costos que generen els diferents recursos desplegats i així evitar sobre-costos.

### 6.2.8 Gestió de permisos: IAM

Les operacions que l'infraestructura ha de realitzar per a dur a terme la correcta execució de la simulació requereixen de permisos especials per tal de que els recursos d'Amazon creats puguin interactuar entre ells.

Els diferents grups de permisos s'agrupen en rols, i són els rols els que són assignats a cadascun dels recursos que els necessita.

El rol de la instància EC2 permet que les màquines del compute environment actuïn com a nodes de Batch i puguin descarregar la imatge de ECR i enviar logs a CloudWatch.

El rol de servei d'AWS Batch el qual permet que el servei administri la infraestructura de còmput i coordini l'execució dels treballs.

El rol del job té els permisos necessaris per a que el contenidor pugui interactuar amb S3, ja que els resultats de la simulació es pugen a S3 des del script.

## 6.3 AWS ECS

Amazon Elastic Container Service (ECS) és un servei gestionat que permet desplegar, executar i orquestrar contenidors Docker de forma escalable i integrada amb la resta de serveis cloud d'AWS. ECS actua com a capa d'orquestració de contenidors dins de l'arquitectura, és a dir, és el servei encarregat de definir quins contenidors s'han d'executar, amb quins recursos, en quines condicions i en quin entorn de xarxa. L'usuari defineix tasks i services especificant aspectes com la imatge de Docker, CPU i memòria assignades, les variables d'entorn o el nombre d'instàncies que s'han d'executar. ECS s'encarrega de coordinar l'execució dels contenidors, supervisar-ne l'estat i garantir que el nombre desitjat d'execucions es mantingui actiu.

### 6.3.1 AWS Fargate

AWS Fargate és un servei serverless d'Amazon que permet executar contenidors. Aquest servei està integrat amb ECS i altres serveis d'orquestració d'AWS. Aquest servei és l'encarregat d'aprovisionar el còmput a la nova infraestructura. Amb Fargate no és necessari aprovisionar manualment màquines virtuals o instàncies EC2, ni configurar ni administrar servidors manualment. L'usuari només defineix els recursos que necessita cada contenidor i Fargate s'encarrega d'aprovisionar-los perquè es puguin executar correctament.

### 6.3.2 AWS Batch vs AWS ECS + AWS Fargate

La combinació d'AWS ECS i AWS Fargate [5] representa una aproximació més modular i flexible respecte a AWS Batch. Mentre que Batch abstrau gran part de la gestió interna i està orientat específicament a l'execució de treballs batch, ECS + Fargate ofereix un nivell de control molt més granular sobre la infraestructura i el comportament dels contenidors. Amb ECS es poden definir manualment aspectes avançats com la configuració detallada de les tasques, les polítiques de xarxa, l'assignació de recursos, els mecanismes d'escalat, la integració amb serveis interns, la persistència, el balanceig de càrrega o les estratègies de desplegament.

En aquest sentit AWS Batch es pot entendre com una capa d'abstracció superior construïda sobre tecnologies de còmput i orquestració com ECS i Fargate. Batch simplifica l'execució de

càrregues batch automatitzant la planificació dels jobs, la gestió de les cues, les dependències entre tasques i l'escalat dels recursos. Aquesta abstracció redueix considerablement la complexitat operativa, però al mateix temps limita la capacitat de personalització i control fi sobre determinats components de la infraestructura.

Per aquest motiu, substituir AWS Batch per una arquitectura basada en ECS + Fargate aporta una major flexibilitat i capacitat de personalització; no obstant això, aquest increment de control implica també una arquitectura més complexa.

## 6.4 Arquitectura de l'infraestructura amb AWS ECS + AWS Fargate

Per a aquesta nova infraestructura s'ha dissenyat una nova infraestructura equivalent amb Amazon ECS i AWS Fargate, amb l'objectiu de substituir el model d'execució de jobs basat en cues i instàncies EC2 per un model serverless basat en l'execució directa de tasks.

Aquesta nova aproximació manté la mateixa lògica funcional de l'aplicació, de manera que les simulacions de Monte Carlo continuen executant-se dins de contenidors Docker però es modifica completament la capa de còmput subjacent.

L'objectiu principal d'aquest canvi és avaluar l'impacte en el canvi d'infraestructura, en termes d'eficiència econòmica i de rendiment.

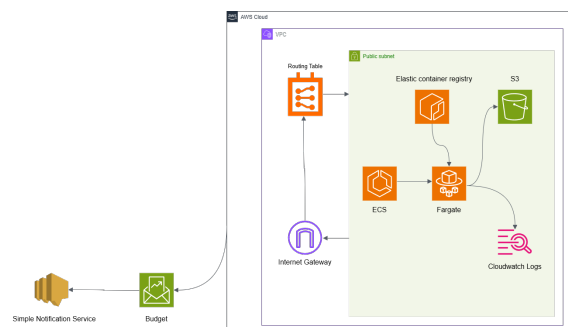


Fig. 3: Arquitectura de la infraestructura amb AWS ECS i Fargate

### 6.4.1 AWS ECS: cluster definition i task definition

En aquesta nova arquitectura, l'ECS Cluster representa la unitat lògica que agrupa les diferents execucions de les tasques. Per al nostre cas d'ús, el cluster creat no disposa de capacitat computacional pròpia ni requereix de la gestió de servidors. A diferència d'altres arquitectures basades en EC2, el cluster en Fargate no actua com a infraestructura de còmput, sino com a contenidor administratiu on s'executen les tasks.

La task definition és el component més important del sistema, ja que defineix com s'ha d'executar cadascun dels contenidors. Aquesta definició substitueix conceptualment a la definició de tasques que es feia en l'arquitectura anterior.

Per a aquest cas, s'han creat dues tasques, una paral·lela i una seqüencial, les quals s'han anat modificant amb terraform per a adaptar les diferents simulacions. Es podrien haver creat 4 tipus de tasques diferents però s'ha volgut aprofitar la simplicitat de gestió de la infraestructura que ens dona el llenguatge per a només tenir-ne dos i adaptar-les als diferents escenaris.

La tasca per a l'execució de la simulació paral·lela consta de 2048 cpu units, 2 vCPUs, i 4096 Mb de memòria principal.

La tasca per a l'execució de la simulació seqüencial ha constatat de 1024 cpu units, 1 vCPU, i 1024Mb, 2048Mb i 4096 Mb de memòria principal segons el cas que es volgués avaluar.

A les configuracions comuns a les dues task definition he hagut de definir el network mode com a awsvpc per a dotar a cada container de la seva pròpia IP privada i interfície de xarxa. Aquesta

configuració és obligatòria quan s'usa Fargate. Al paràmetre requieres compatibilitats hem definit que volem que usi Fargate com a motor de còmput.

També he hagut de definir un rol específic per a que el clúster ECS pugui executar tasks i un rol específic per a que cadascuna de les tasks tingués els permisos necessaris per a executar correctament la simulació. Aquests permisos s'expliquen amb més detall a l'apartat 5.4.7.

Finalment al paràmetre container definitions és on s'ha definit quins paràmetres tindran cadascun dels contenidors que s'executaran dins de les tasques Fargate.

Dins d'aquest JSON de configuració es defineix el nom que rebrà cadascun dels containers, d'on obtindran la imatge Docker que usaran, si són crítics per a l'execució de la tasca que els conté, quina comanda executaran, variables d'entorn necessàries per a l'execució del codi que contenen i la configuració dels logs necessària.

#### 6.4.2 Repositori: Amazon ECR

Per al repositori de les imatges s'ha emprat la mateixa configuració per al repositori abans creat. En aquest cas, amb l'objectiu de reutilitzar les imatges de Docker ja existents i evitar la seva recreació el recurs ha estat importat per Terraform, permetent gestionar-lo dins de la nova infraestructura sense modificar el seu contingut previ.

#### 6.4.3 Xarxa: AWS VPC, AWS subnets, Internet Gateway i routing table

Per a la configuració de xarxa s'ha reutilitzat pràcticament la mateixa arquitectura emprada en la infraestructura basada en AWS Batch, mantenint la mateixa topologia de VPC, subxarxes i taules de rutes. Aquesta reutilització ha estat possible gràcies a la gestió de la infraestructura mitjançant Terraform, ja que l'entorn anterior va poder ser destruït completament abans del desplegament de la nova infraestructura basada en ECS i Fargate.

D'aquesta manera, no ha estat necessari definir una nova VPC ni crear nous blocs CIDR o rangs d'adreçament IP, permetent reutilitzar la mateixa configuració de xarxa base.

#### 6.4.4 Regles de seguretat: AWS Security Group

Per al Security Group de la infraestructura, s'ha definit una configuració de sortida adaptada per a que cada task que s'executi amb ECS pugui accedir al repositori ECR i obtenir la imatge de docker necessària per a la simulació, pujar els logs de la simulació al log group corresponent de CloudWatch i per a poder pujar els resultats de la simulació al bucket S3.

#### 6.4.5 Emmagatzematge: Amazon S3

En quant a l'emmagatzematge, no hi ha canvis significatius a la configuració. S'ha decidit crear un bucket S3 separat per a emmagatzemar els resultats de les proves amb la nova infraestructura. En aquest cas, s'ha configurat la retenció del contingut del bucket a 60 dies, ja que l'output de la simulació són fitxers JSON que pesen molt poc.

#### 6.4.6 Logging: CloudWatch Logs

En la configuració de la infraestructura, s'ha creat un nou log group amb el nom /aws/ecs/ecs-fargate-logs amb una retenció dels logs de 7 dies. S'ha escollit aquest número de dies per evitar acumulació de logs que incorrerien en costos extra.

#### 6.4.7 Gestió de permisos: IAM

AWS per a aquesta nova infraestructura s'han hagut de crear dos rols completament diferents: el rol per a que el servei ECS pugui executar noves tasks (ECS task execution role) i el rol per a que el codi que hi ha a cadascuna de les tasques tingui els permisos necessaris per a poder dur a terme la seva execució correctament (ECS task role)

El ECS task execution rol garanteix al servei ECS els permisos per a autenticar-se al repositori ECR i descarregar la imatge amb tot el que necessita la task per a executar la simulació i enviar logs d'execució a CloudWatch.

El ECS task role garanteix permisos d'escriptura i de lectura a S3 a cadascuna de les tasques per tal de poder emmagatzemar correctament els resultats de les simulacions de Monte Carlo.

### 6.5 Resultats

Per avaluar el comportament de la infraestructura desplegada, s'han realitzat diverses execucions de la simulació de Monte Carlo sota diferents configuracions. L'objectiu principal d'aquestes proves és analitzar l'impacte del paral·lisme i de la disponibilitat de recursos (CPU i memòria) no només sobre el temps d'execució dels jobs, sinó també sobre els costos associats a l'ús de la infraestructura desplegada. Cal mencionar que els valors emprats per al càlcul de la simulació de Monte Carlo —la taxa de fallada per hora de la CPU, la taxa de fallada per hora de la memòria principal i la taxa de fallada per hora dels dispositius d'E/S— són valors aproximats i no reals, ja que no ha estat possible obtenir-los a partir de la fitxa tècnica de cap microcontrolador. Per aquest motiu, s'han considerat suficients per a la realització del càlcul en la infraestructures desplegadas.

#### 6.5.1 Resultats: AWS Batch

S'han definit tres escenaris diferenciats en funció del nombre d'iteracions: 15 milions, 30 milions i 45 milions, comparant directament entre execució seqüencial i execució paral·lela.

En la primera prova, corresponent a 15 milions d'iteracions, s'ha utilitzat una configuració amb 1024 Mb de memòria tant per a l'execució seqüencial com per a la paral·lela. La diferència entre ambdues radica exclusivament en la capacitat de computació:

- Execució seqüencial: 1 vCPU
- Execució paral·lela: 2 vCPUs

En la segona prova, amb 30 milions d'iteracions, s'ha mantingut la mateixa configuració de vCPUs, però s'ha incrementat la memòria a 2048 Mb. Aquesta decisió ve donada per l'obtenció d'errors OutOfMemoryError durant les proves preliminars amb 1024 Mb.

Finalment, en la tercera prova, amb 45 milions d'iteracions, s'ha augmentat la memòria fins a 3072 Mb. Aquesta configuració s'ha escollit tenint en compte el límit de la instància utilitzada (aproximadament 3,7 Gb de memòria), aprofitant així gairebé tota la capacitat disponible sense comprometre l'estabilitat del sistema.

Els resultats obtinguts mostren que, en tots els escenaris analitzats, l'execució en paral·lel presenta un millor rendiment respecte a l'execució seqüencial. Tal com es pot observar a la gràfica, aquesta millora és progressivament més significativa a mesura que augmenta la càrrega de treball, fet que indica una millor explotació del paral·lisme en escenaris de càlcul més intensiu.

No obstant això, la diferència entre ambdós modes d'execució no arriba a ser proporcional al nombre de vCPUs utilitzades, cosa que evidencia l'existència de factors que limiten l'escalabilitat del sistema.

Tenint en compte que s'han fet 10 execucions per a cada cas en paral·lel i cada cas en seqüencial, els speedups obtinguts en temps promig de les 10 execucions en els diferents escenaris es defineixen com:

$$S_{15M} = \frac{0,94}{0,82} \approx 1,146$$

$$S_{30M} = \frac{1,99}{1,66} \approx 1,198$$

$$S_{45M} = \frac{3,20}{2,56} = 1,25$$

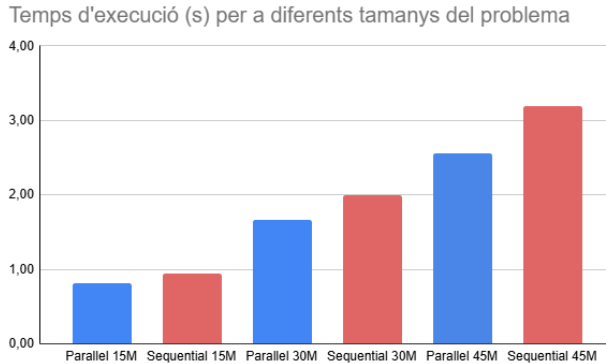


Fig. 4: Gràfica de temps d'execució de la simulació de Monte Carlo a l'infraestructura AWS Batch

### 6.5.2 Resultats: AWS ECS + AWS Fargate

Per avaluar el comportament de la infraestructura basada en AWS ECS i AWS Fargate, s'han realitzat diverses execucions de la simulació de Monte Carlo sota les mateixes configuracions de càrrega utilitzades en el cas d'AWS Batch (15, 30 i 45 milions d'iteracions). L'objectiu principal d'aquestes proves és comparar el rendiment i el comportament del sistema respecte a la infraestructura anterior, mantenint el mateix tipus de càrrega de treball.

En aquest cas, les configuracions de recursos han estat condicionades per les restriccions pròpies de Fargate. Per a la configuració de 2 vCPUs, la plataforma requereix un mínim de 4 GB de memòria, motiu pel qual aquesta capacitat s'ha mantingut constant en tots els escenaris corresponents a aquesta configuració. En el cas d'1 vCPU, les proves s'han realitzat amb configuracions d'1 GB, 2 GB i 4 GB de memòria, adaptant-se als valors permesos pel servei, ja que no es permet l'ús de configuracions intermèdies com 3 GB.

A diferència del cas d'AWS Batch, aquestes restriccions limiten parcialment la capacitat d'ajust fi dels recursos, fet que pot influir en l'eficiència del paral·lelisme i en el temps d'execució en escenaris de major càrrega. No obstant això, en el context d'aquest treball, aquesta limitació no ha tingut un impacte significatiu en la comparabilitat global dels resultats.

Els resultats obtinguts mostren un comportament consistent amb l'observat en AWS Batch: la configuració paral·lela presenta millors temps d'execució que la seqüencial, especialment a mesura que augmenta la càrrega de treball. Tanmateix, la diferència de rendiment no escala de manera estrictament proporcional al nombre de vCPUs, fet que suggereix l'existència de colls d'ampolla associats a la pròpia naturalesa de la càrrega de treball i a la sobrecàrrega de paral·lelització.

Tenint en compte que s'han fet el mateix nombre d'execucions que a l'apartat anterior per als dos casos, els speedups obtinguts en temps promig de les 10 execucions són els següents:

$$S_{15M} = \frac{1,10}{0,77} \approx 1,431$$

$$S_{30M} = \frac{2,08}{1,54} \approx 1,350$$

$$S_{45M} = \frac{3,31}{2,21} \approx 1,498$$

Temps d'execució (s) per a diferents tamanyos del problema

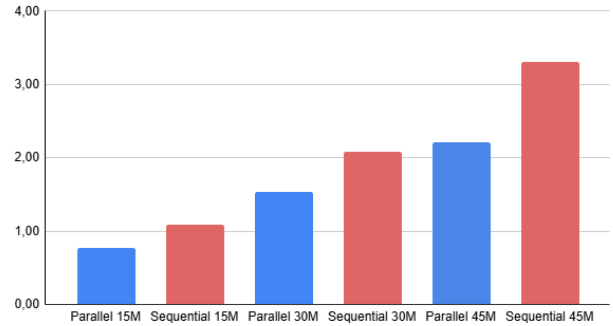


Fig. 5: Gràfica de temps d'execució de la simulació de Monte Carlo a l'infraestructura AWS ECS amb Fargate

### 6.5.3 Resultats: Costos

En aquest apartat es presenta l'anàlisi de costos associat a les dues arquitectures estudiades durant aquest treball considerant dos modes d'aprovisionament de recursos de còmput: on demand i spot.

#### 6.5.4 Resultats: Costos on demand

El model on demand és el model d'aprovisionament de recursos de còmput per defecte que té AWS. L'usuari paga pel temps d'ús dels recursos sense necessitat de compromisos previs ni reserves de capacitat. Aquest model garanteix disponibilitat immediata dels recursos, sempre que existeixi capacitat suficient a la regió seleccionada.

En el cas d'AWS Batch amb instàncies EC2 on demand, s'ha consumit un total de 0.19\$[18] per a les 75 execucions necessàries per dur a terme l'experiment. A partir d'aquesta dada, es pot calcular el cost per execució com:

$$C_{exec} = \frac{0.19}{75} \approx 0.00253\$$$

Si fèssim un milió d'execucions, per tal d'obtenir un volum de simulació més representatiu, obtindríem els següents costos

$$C_{Mexec} = 0.00253 \cdot 1.000.000 \approx 2533\$$$

Per una altra banda, en el cas de l'arquitectura amb AWS ECS i Fargate s'han gastat 0.11\$[19] per al mateix nombre d'execucions. El cost per execució en aquest escenari és:

$$C_{exec} = \frac{0.11}{75} \approx 0.00147\$$$

Si fèssim un milió d'execucions, per tal d'obtenir un MTBF més proper a la realitat, obtindríem els següents costos

$$C_{Mexec} = 0.00147 \cdot 1.000.000 \approx 1473\$$$

En termes comparatius, l'arquitectura basada en AWS ECS i Fargate presenta un cost inferior tant per execució com per volum equivalent de càlcul, mostrant una major eficiència econòmica per a càrregues de treball altament paral·lelitzables com les simulacions de Monte Carlo. En concret, si es comparen els costos totals observats (0.19\$ per AWS Batch amb EC2 i 0.11\$ per AWS Fargate), s'obté un estalvi relatiu de

$$\frac{0.19 - 0.11}{0.19} \cdot 100 \approx 42.11\%$$

evidenciant un menor cost global en el cas de ECS amb Fargate.

### 6.5.5 Resultats: Costos spot

El model spot és un mecanisme de provisió de recursos de còmput basat en la utilització de la capacitat sobrant dins la infraestructura d'Amazon Web Services. AWS disposa d'una gran quantitat de recursos per a clients on demand, però en determinats moments hi ha capacitat que no està sent utilitzada. El model Spot ens permet aprofitar aquesta capacitat a un cost significativament inferior, a canvi d'acceptar que aquests recursos poden ser retirats en qualsevol moment.

El funcionament d'aquest model d'aprovisionament de còmput varia segons el recurs que vulguem emprar. Per al cas de les instàncies EC2, quan AWS necessita els recursos que estem utilitzant per a provisionar-los a clients on demand avisa amb 2 segons d'antelació i la nostra instància passa a estar terminated automàticament. En el cas de les tasques ECS a Fargate spot, si un client on demand necessita el còmput que estem ocupant, AWS marcarà la nostra task com a stopped.

Per al cas del còmput en spot, la infraestructura que usa AWS Batch ha consumit 0.08\$[17] per a realitzar les 75 simulacions necessàries per a dur a terme l'experiment. A partir d'aquesta dada, es pot calcular el cost per execució com:

$$C_{exec} = \frac{0.08}{75} \approx 0.00107\$$$

Si fèssim un milió d'execucions, per tal d'obtenir un volum de simulació més representatiu, obtindríem els següents costos:

$$C_{Mexec} = 0.00107 \cdot 1.000.000 \approx 1067\$$$

Per una altra banda, en el cas de l'arquitectura amb AWS ECS i Fargate s'han gastat 0.04\$ en còmput per a dur a terme les simulacions necessàries per a dur a terme l'experiment. El cost per execució en aquest escenari és:

$$C_{exec} = \frac{0.04}{75} \approx 0.000533\$$$

Si fèssim un milió d'execucions, per tal d'obtenir un MTBF més proper a la realitat, obtindríem els següents costos:

$$C_{Mexec} = 0.000533 \cdot 1.000.000 \approx 533\$$$

En aquest cas, l'arquitectura basada en AWS ECS i Fargate presenta també un cost inferior tant per execució com per volum equivalent de càlcul, mostrant una major eficiència econòmica per a càrregues de treball altament paral·lelitzables com les simulacions de Monte Carlo. En concret, si es comparen els costos totals observats (0.08\$ per AWS Batch amb Spot i 0.04\$ per AWS Fargate), s'obté un estalvi relatiu de:

$$\frac{0.08 - 0.04}{0.08} \cdot 100 = 50\%$$

evidenciant un menor cost global en el cas de ECS amb Fargate.

## 7 CONCLUSIONS

En aquest treball s'han pogut explorar diferents maneres de crear una arquitectura per a dur a terme la mateixa funcionalitat. Cadascuna d'elles amb diferents requeriments i diferents casos d'ús.

En el cas de la primera arquitectura esmentada, AWS Batch i EC2, el servei AWS Batch ens abstrau d'entendre com funciona realment tota la infraestructura necessària per a dur a terme l'experiment, és a dir: creant un entorn de còmput amb el servei abans esmentat i aprenent com funcionen alguns dels seus components és més que suficient per a poder llançar una simulació que compleixi amb els requisits que s'havien plantejat a l'inici d'aquest treball.

Per altra banda, amb l'arquitectura desenvolupada amb AWS ECS i Fargate, s'ha hagut d'aprendre com funcionen cadascun dels

serveis necessaris, que l'anterior arquitectura administrava automàticament, per a poder desenvolupar l'arquitectura necessària per a dur a terme la simulació. En aquest cas hem augmentat la complexitat però tenim un control molt més fi sobre els diferents paràmetres de la infraestructura.

D'una altra banda, s'han pogut veure les diferències de costos monetaris entre les dues solucions. Per una banda s'ha explorat una implementació amb còmput on demand, la qual s'ha vist que resulta més cara però amb l'avantatge de poder tenir disponibilitat computacional en qualsevol moment que ho considerem necessari. A l'altra banda de l'espectre tindríem el còmput spot, el qual ha resultat molt adequat per al meu cas d'ús, ja que la duració de les simulacions ha propiciat que no patís cap tall de servei en cap de les meves simulacions. Per tant, es podria afirmar que és més adequat usar còmput spot que còmput on demand per a aquest cas d'ús.

En referència també als costos monetaris, s'ha pogut observar que en ambdós casos hi ha una diferència de preu notable entre les dues arquitectures. Aquest factor és degut a que de per si els costos de còmput Task són més assequibles que el còmput EC2, ja que en un dels casos tenim disponibilitat només per a executar scripts i a l'altra tenim una màquina virtual completa al cloud. La diferència monetària també és deguda a que, en el cas del còmput amb EC2, hi ha un overhead de temps d'ús de cadascuna de les instàncies el qual fa que els costos es disparin quantes més execucions de la simulació individuals es facin. Aquest temps d'ús extra es deu a la naturalesa de les instàncies EC2: les màquines virtuals al cloud necessiten un temps per a iniciar-se i estar operatives, start-up time, i un temps per a apagar-se i destruir-se. Aquests temps sumats entre si, per breus que siguin, fan que s'acabi consumint més temps d'ús de còmput EC2 del que realment es necessita per a les simulacions. Per tant, aquest factor també influeix en la diferència de preu significativa. Per aquests dos factors i per la naturalesa de ECS amb Fargate, l'arquitectura que utilitza les tasques és la que més s'adequa al meu cas d'ús, ja que no em cal usar una màquina virtual completa en cada execució de la simulació, sinó que amb les tasques que es creen en la segona arquitectura és més que suficient.

Pel que fa a les arquitectures desenvolupades en aquest treball de recerca, es podria haver desenvolupat una infraestructura usant el primer servei d'Amazon, AWS Batch, amb Fargate com a base de còmput. S'ha decidit no fer-la perquè no aportava res nou ni en matèria de la comparativa ni en matèria d'aprenentatge. Donada la naturalesa del servei mencionat, aquesta infraestructura hauria consumit, monetàriament parlant, exactament el mateix que l'arquitectura desenvolupada amb ECS i Fargate. L'únic avantatge que suposava respecte a la primera arquitectura era un baix cost de còmput amb un nivell d'abstracció molt elevat, cosa que li restava sentit a l'hora de comparar-la amb les altres dues.

## 8 ÚS D'INTEL·LIGÈNCIA ARTIFICIAL

Per a aquest treball la intel·ligència artificial s'ha utilitzat com a eina de suport per a la consulta de documentació, ja sigui per buscar-la o per a filtrar els diferents paràmetres de configuració dels recursos de Terraform utilitzats. També s'ha emprat com a eina de suport en la redacció del document amb la finalitat de millorar la qualitat de l'escriptura.

### Bibliografia

1. Advanced Energy, "Interpretation of MTBF and its applications," Application Note. [En línia]. Disponible a: <https://www.advancedenergy.com/getmedia/bdb1ed5f-f6eb-4a53-9c79-273778638bf1/en-lv-interpretation-of-mtbf-application-note.pdf>

2. V. Avelar, "Mean Time Between Failure: Explanation and Standards," APC White Paper. [En línia]. Disponible a: [https://www.researchgate.net/publication/251895269\\_Mean\\_Time\\_Between\\_Failure\\_Explanation\\_and\\_Standards](https://www.researchgate.net/publication/251895269_Mean_Time_Between_Failure_Explanation_and_Standards)
3. L. J. Wells et al., "Reliability and availability analysis of complex systems," IFAC-PapersOnLine, vol. 52, núm. 13, pp. 25-30, 2019. [En línia]. Disponible a: <https://www.sciencedirect.com/science/article/pii/S2405896319313047>
4. Amazon Web Services, "Optimizing Monte Carlo simulations using AWS Batch," AWS HPC Blog, 2024. [En línia]. Disponible a: <https://aws.amazon.com/es/blogs/hpc/optimizing-monte-carlo-simulations-using-aws-batch/>
5. Amazon Web Services, "Deep dive into Fargate Spot to run your ECS tasks for up to 70% less," AWS Compute Blog. [En línia]. Disponible a: <https://aws.amazon.com/es/blogs/compute/deep-dive-into-fargate-spot-to-run-your-ecs-tasks-for-up-to-70-less>
6. J. Harrison, "Monte Carlo methods and their applications," PMC, 2010. [En línia]. Disponible a: <https://pmc.ncbi.nlm.nih.gov/articles/PMC2924739/>
7. GitHub Docs, "Desplegar en plataformes de tercers: Amazon Elastic Container Service," *GitHub Actions Documentation*. [En línia]. Disponible a: <https://docs.github.com/es/actions/how-tos/deploy/deploy-to-third-party-platforms/amazon-elastic-container-service>
8. Amazon Web Services, "Què és AWS Batch?," *AWS Batch User Guide*. [En línia]. Disponible a: <https://docs.aws.amazon.com/batch/latest/userguide/what-is-batch.html>
9. Amazon Web Services, "AWS Fargate a Amazon ECS," *Amazon ECS Developer Guide*. [En línia]. Disponible a: [https://docs.aws.amazon.com/es\\_es/AmazonECS/latest/developerguide/AWS\\_Fargate.html](https://docs.aws.amazon.com/es_es/AmazonECS/latest/developerguide/AWS_Fargate.html)
10. IBM, "What is Monte Carlo Simulation?," *IBM Think Topics*. [En línia]. Disponible a: <https://www.ibm.com/think/topics/monte-carlo-simulation>
11. HashiCorp, "Terraform Documentation," *Terraform by HashiCorp*. [En línia]. Disponible a: <https://developer.hashicorp.com/terraform/docs>
12. Amazon Web Services, "Best practices for AWS Batch," *AWS Batch User Guide*. [En línia]. Disponible a: <https://docs.aws.amazon.com/batch/latest/userguide/bestpractice5.html>
13. Amazon Web Services, "Amazon ECS Clusters," *Amazon ECS Developer Guide*. [En línia]. Disponible a: <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/clusters.html>
14. D. P. Landau i K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, 4a ed., Cambridge, Regne Unit: Cambridge University Press, 2014.
15. F. Schenkelberg, "The Reliability Metric: A Quick and Valuable Improvement Over MTBF," FMS Reliability, 2015.
16. Amazon Web Services, "The difference between On-Demand Instances and Reserved Instances," AWS. [En línia]. Disponible a: <https://aws.amazon.com/es/compare/the-difference-between-on-demand-instances-and-res>
17. Amazon Web Services, "Amazon EC2 Spot Instances Pricing," AWS EC2. [En línia]. Disponible a: <https://aws.amazon.com/es/ec2/spot/pricing/>
18. Amazon Web Services, "Amazon EC2 On-Demand Pricing," AWS EC2. [En línia]. Disponible a: <https://aws.amazon.com/es/ec2/pricing/on-demand/>
19. Amazon Web Services, "AWS Fargate Pricing," AWS Fargate. [En línia]. Disponible a: <https://aws.amazon.com/es/fargate/pricing/>

## A FORMULA MTBF

$$MTBF = \frac{T}{N}$$

$T$  Temps total de funcionament observat.

$N$  Nombre de fallades ocorregudes durant aquest període.

## B TAULES DE TEMPS D'EXECUCIÓ

### B.1 Taula de temps d'execució amb Batch

| P 15M       | S 15M       | P 30M       | S 30M       | P 45M       | S 45M       |
|-------------|-------------|-------------|-------------|-------------|-------------|
| 0,806986    | 0,913132    | 1,600804    | 1,987061    | 2,429225    | 3,113831    |
| 0,815354    | 0,992615    | 1,627138    | 1,992642    | 2,414141    | 3,083663    |
| 0,835163    | 0,891777    | 1,820037    | 2,002434    | 2,448145    | 3,106485    |
| 0,806997    | 0,887701    | 1,607998    | 1,994179    | 2,538201    | 3,136946    |
| 0,818447    | 0,975236    | 1,776269    | 1,990993    | 2,502496    | 3,222653    |
| 0,834902    | 0,945118    | 1,578880    | 1,983944    | 2,501944    | 3,133975    |
| 0,815006    | 0,987553    | 1,576106    | 1,995332    | 2,692941    | 3,155640    |
| 0,833121    | 0,889443    | 1,594927    | 1,985857    | 2,567440    | 3,142223    |
| 0,815006    | 0,975236    | 1,788449    | 1,994906    | 2,713574    | 3,276806    |
| 0,812342    | 0,965118    | 1,603499    | 2,000007    | 2,831389    | 3,615305    |
| <b>0,82</b> | <b>0,94</b> | <b>1,66</b> | <b>1,99</b> | <b>2,56</b> | <b>3,20</b> |

### B.2 Temps d'execució amb ECS i Fargate

| P 15M       | S 15M       | P 30M       | S 30M       | P 45M       | S 45M       |
|-------------|-------------|-------------|-------------|-------------|-------------|
| 0,890418    | 1,106181    | 1,761808    | 2,439632    | 2,599942    | 3,302354    |
| 0,879991    | 1,087896    | 1,419970    | 2,209193    | 2,519337    | 3,303764    |
| 0,510844    | 1,095442    | 1,407708    | 2,210923    | 2,596761    | 3,321486    |
| 0,709608    | 1,098993    | 1,731416    | 2,183682    | 1,515806    | 3,274391    |
| 0,880215    | 1,107970    | 1,733813    | 1,363044    | 2,076038    | 3,343338    |
| 0,936765    | 1,094060    | 1,411185    | 1,472625    | 2,078337    | 3,286937    |
| 0,936765    | 1,120014    | 1,395689    | 2,134532    | 2,583790    | 3,277616    |
| 0,710982    | 1,111464    | 1,400563    | 2,277247    | 2,046745    | 3,296852    |
| 0,524505    | 1,090427    | 1,385400    | 2,263943    | 1,922834    | 3,431131    |
| 0,705825    | 1,087301    | 1,742352    | 2,204453    | 2,170392    | 3,301965    |
| <b>0,77</b> | <b>1,10</b> | <b>1,54</b> | <b>2,08</b> | <b>2,21</b> | <b>3,31</b> |