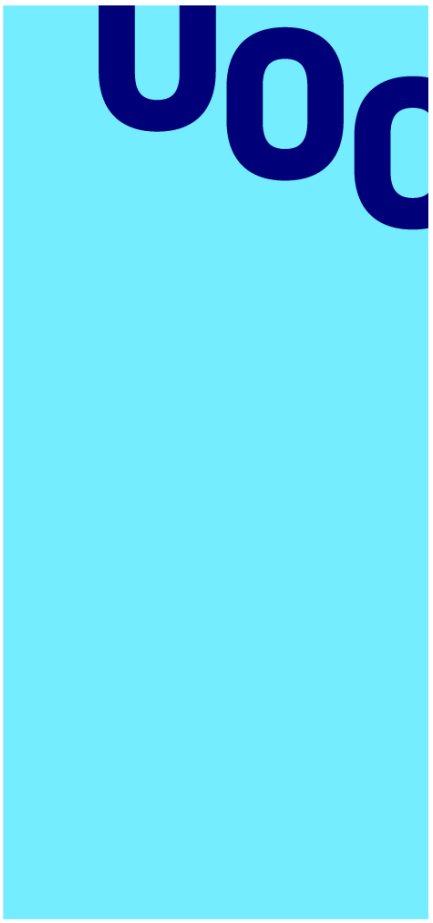
Disseny i implementació d'un clúster de K8s en AWS

Una guia pràctica enfocada en l'automatització i l'*IaC* per a empreses tecnològiques



Universitat Oberta de Catalunya

Albert Jubany Juarez

Grau d'Enginyeria Informàtica
Aplicacions i sistemes distribuïts

Tutor/a de TF
Amadeu Albós Raya

Professor/a responsable de l'assignatura
Joan Manuel Marquès Puig

Data d'entrega
11/06/2024

FITXA DEL TREBALL FINAL

Títol del treball:	*Disseny i implementació d'un clúster de Kubernetes en AWS: Un enfocament pràctic aplicant automatització i IaC per a empreses tecnològiques*
Nom de l'autor:	*Albert Jubany Juàrez*
Nom del consultor/a:	*Amadeu Albós Raya*
Nom del PRA:	*Joan Manuel Marquès Puig*
Data de lliurament (mm/aaaa):	*06/2024*
Titulació o programa:	*Grau d'Enginyeria Informàtica*
Àrea del Treball Final:	*Aplicacions i sistemes distribuïts*
Idioma del treball:	*Català*
Paraules clau	*Kubernetes, IaC, AWS*

Resum del Treball

Aquest treball de fi de grau té com a finalitat desenvolupar una solució tecnològica per millorar l'eficiència, l'escalabilitat i la gestió de recursos de les aplicacions d'una empresa mitjançant Kubernetes i Amazon EKS. El context de l'aplicació és la necessitat de modernitzar una arquitectura d'aplicacions per aprofitar millor les capacitats d'automatització i gestió eficient que ofereix Kubernetes.

La metodologia seguida inclou l'ús d'eines de infraestructura com a codi (IaC) per automatitzar el desplegament i gestió de la infraestructura a AWS, així com l'estudi i implementació de components clau un gestor de xarxa, resolutor de noms, i diferents eines per a escalar i desescalar ràpidament la infraestructura.

Els resultats del projecte demostren que la implementació del clúster de Kubernetes en AWS ha millorat significativament l'eficiència operativa i l'escalabilitat de les aplicacions amb les que s'han realitzat les proves. Les proves funcionals han mostrat una millor gestió dels recursos, major flexibilitat i una reducció de costos operatius causada per els punts anteriors.

Les conclusions del treball indiquen que la solució desplegada no només satisfà les necessitats actuals de l'empresa, sinó que també estableix una base sòlida per a futurs creixements i adaptacions tecnològiques, promovent una infraestructura més dinàmica i eficient. Això ofereix una major agilitat operativa i una millor disponibilitat dels serveis per als usuaris finals.

Abstract

The purpose of this bachelor's thesis is to develop a technological solution to improve the efficiency, scalability, and resource management of a company's applications using Kubernetes and Amazon EKS. The context of application is the need to modernize an application architecture to better leverage the automation and efficient management capabilities offered by Kubernetes.

The methodology followed includes the use of Infrastructure as Code (IaC) tools to automate the deployment and management of infrastructure on AWS, as well as the study and implementation of key components such as a network manager, name resolver, and various tools to scale the infrastructure.

The project's results demonstrate that the implementation of the Kubernetes cluster on AWS has significantly improved the operational efficiency and scalability of the company's applications. Functional tests have shown better resource management, greater flexibility, and a reduction in operational costs due to these improvements.

The conclusions of the work indicate that the deployed solution not only meets the current needs of the company but also establishes a solid foundation for future growth and technological adaptations, promoting a more dynamic and efficient infrastructure. This provides greater operational agility and better service availability for end users.

Agraïments

Ara que finalment estic arribant al final del grau, no puc acabar-lo sense abans expressar tots els meus agraïments que no he pogut formular en tots aquests anys del grau.

Primer de tot, moltes mercès a la UOC per apostar per un format d'estudis no presencial, que permet a gent com jo que es troba treballant poder estudiar. Al personal de la UOC, professors i consultors, mercès per el vostre treball i suport.

A l'empresa on treballo, Socialpoint, per permetre'm realitzar aquest projecte que estic segur que beneficiarà ambdues parts. I sobretot als meus companys (i ex-companys), Andrés, Lolo, Alex, Jaume, Ludo i LuisMi.

Als meus grans amics, JoanJi, Jan, Silvestre, Otger i Kim, per la vostra paciència i ànims, en els bons i mals moments, i per ajudar-me sempre que ho he necessitat.

I sobretot a la meva família i la meva parella, la Sara. Moltíssimes gràcies per sempre ser allà, ajudant-me i animant-me, suportant als meus nervis i dies de cansament. De veritat que sense vosaltres res d'això hagués estat possible.

Índex

Llista de figures

1. Introducció

1.1. Context i justificació del Treball

L'evolució tecnològica actual exigeix que les empreses desenvolupin estratègies per a la modernització de les seves arquitectures d'aplicacions, amb l'objectiu de millorar l'escalabilitat, la disponibilitat i l'eficiència operativa. En aquest sentit, l'empresa on es desenvolupa aquest treball de fi de grau es troba davant del repte de migrar la seva arquitectura d'aplicacions des de un model basat en instàncies AWS EC2 i ASG (*Auto Scaling Groups*) cap a una solució més dinàmica i escalable mitjançant els contenidors d'imatges, i aquí entra en joc Kubernetes.

Aquesta necessitat de migració respon a una problemàtica concreta: l'arquitectura actual no aprofita plenament les possibilitats d'automatització, escalabilitat, flexibilitat i gestió eficient d'aplicacions que ofereix un entorn orquestrat per contenidors. La resposta a aquesta situació és el disseny i la implementació d'un clúster de Kubernetes dins d'AWS, que pot oferir una infraestructura més flexible i adaptable a les necessitats canviants de l'empresa.

El projecte té com a objectiu principal el desenvolupament d'una solució tecnològica que permeti a l'empresa superar els límits de l'arquitectura actual, mitjançant la creació d'un entorn gestionat de Kubernetes a través d'Amazon EKS. Això implicarà la planificació detallada i l'execució d'un disseny que integri pràctiques d'infraestructura com a codi (IaC), així com la selecció i configuració de components essencials per a la gestió correcta del clúster.

El treball pretén no només solucionar la necessitat immediata de modernització de la infraestructura d'aplicacions de l'empresa, sinó també establir una base sòlida per al futur creixement i adaptació tecnològica d'aquesta. La implementació d'un clúster de Kubernetes en AWS es planteja, doncs, com una solució estratègica que facilitarà una major agilitat operativa, una gestió més eficient dels recursos i una millora en la disponibilitat dels serveis oferts als usuaris finals.

1.2. Objectius del Treball

Aquest projecte té com a objectiu principal explorar el procés de disseny i implementació d'un clúster de Kubernetes en AWS (EKS), centrant-se en la selecció de eines i components que millorin l'escalabilitat, eficiència i modernització de l'arquitectura on s'executaran les aplicacions de l'empresa. Dins d'aquest marc, els objectius específics són:

1. **Aprofundir en el coneixement d'AWS EKS i Kubernetes**: Adquirir una comprensió detallada de com aquestes tecnologies poden ser aplicades per millorar l'arquitectura de les aplicacions d'una empresa, enfocant-me en l'escalabilitat i l'automatització.
2. **Desenvolupar habilitats en Infraestructura com a Codi (IaC)**: Aprendre i aplicar eines d'IaC com poden ser Terraform o CloudFormation per automatitzar el desplegament i gestió de la infraestructura, així com del clúster de Kubernetes, millorant les competències en gestió de la infraestructura al núvol.
3. **Analitzar i seleccionar objectivament components de Kubernetes**: Adquirir coneixement sobre els diferents components i solucions disponibles per a la gestió del clúster de Kubernetes, amb l'objectiu de poder triar les millors opcions basades en criteris de rendiment, seguretat i cost.
4. **Avaluar el rendiment i la seguretat del clúster**: Desenvolupar la capacitat d'avaluar críticament el rendiment i la seguretat d'un clúster de Kubernetes en producció, amb l'objectiu de comprendre millor els desafiaments i solucions en entorns escalables i segurs.

Superar aquests objectius no només em permetran abordar la problemàtica específica dins de l'empresa sinó també permetran ampliar les meves habilitats i coneixements en àrees clau de la computació en el núvol i la gestió d'aplicacions a gran escala.

1.3. Impacte en sostenibilitat, ètic-social i de diversitat

Aquest TFG s'ha realitzat tenint en compte l'impacte de la implementació d'un clúster de Kubernetes en AWS amb les perspectives de sostenibilitat, responsabilitat ètica i social, i la promoció de la diversitat, el gènere i els drets humans. L'objectiu és assegurar que el projecte no només avança en la innovació tecnològica sinó que també es compromet amb valors de sostenibilitat ambiental, equitat social i inclusivitat.

Sostenibilitat [1]

Impactes positius:

- **Eficiència energètica**: La migració a una arquitectura basada en Kubernetes pot millorar l'eficiència energètica, ja que -utilitzat correctament-, permet una millor gestió dels recursos, minimitzant l'ús innecessari de la infraestructura.
- **Consum responsable**: En línia amb l'ODS 12, aquesta solució promou un consum i producció responsables, optimitzant l'ús dels recursos disponibles i reduint la necessitat de recursos físics per a noves instal·lacions.
- **Acció climàtica**: La reducció de la petjada de carboni és inherent a l'eficiència energètica millorada, contribuint indirectament a l'ODS 13.

Impactes negatius:

- **Consum d'energia durant la implementació**: La fase d'implementació i proves pot implicar un augment temporal del consum energètic, tot i que es tracta d'un impacte de curt termini.

Ètic-social i responsabilitat social

Impactes positius:

- **Innovació i infraestructura resilient**: L'ús de Kubernetes està alineat amb l'ODS 9, fomentant la innovació i una infraestructura resilient que pot suportar el creixement econòmic i el desenvolupament sostenible.
- **Treball decent i creixement econòmic**: La modernització de la infraestructura pot portar a una millora en l'eficiència operativa, contribuint a l'ODS 8 per mitjà de la creació d'oportunitats de treball qualificat en l'àmbit tecnològic.

Impactes negatius:

- **Risc de la privacitat amb les dades**: El maneig inadequat de la configuració i la gestió de dades en un entorn distribuït pot implicar riscos per a la privacitat i la seguretat de les dades, requerint una atenció detallada als aspectes de seguretat i compliment normatiu.

Diversitat, gènere i drets humans

Impactes positius:

- **Reducció d'inequitats**: Amb l'ús d'eines tecnològiques avançades i de codi obert, aquest projecte pot contribuir a la reducció d'inequitats (ODS 10), possibilitant que més organitzacions, incloent-hi aquelles de països en desenvolupament, accedeixin a infraestructures tecnològiques punteres.

Impactes negatius:

- Potencialment, no hi ha impactes negatius directes en aquesta dimensió, sempre que el projecte es desenvolupi amb una consideració conscient de la inclusivitat i l'accessibilitat.

1.4. Enfocament i mètode seguit

Per desenvolupar aquest projecte, s'ha decidit seguir una sèrie d'etapes ben definides que ens ajudaran a abordar el projecte des de l'inici fins a la seva conclusió. Això inclou tot, des de la planificació inicial fins a l'anàlisi final dels resultats. A continuació es descriuran les principals fases del projecte:

1. **Realització del pla de treball**: Aquesta fase inicial consisteix en la definició detallada dels objectius, l'abast del projecte, i l'elaboració d'un pla de treball que inclogui terminis i recursos necessaris.
2. **Estudi de components i preparació de la infraestructura d'AWS**: Aquesta etapa implica una investigació exhaustiva dels components disponibles dins d'AWS i Kubernetes que poden ser utilitzats per a la construcció del clúster. Basant-nos en aquesta investigació, es prepararà la infraestructura d'AWS necessària, optimitzant per a l'escalabilitat, seguretat i eficiència de costos.
3. **Instal·lació del clúster i configuració de components**: Després de preparar la infraestructura, es procedirà amb la instal·lació del clúster de Kubernetes en AWS, seguit de la instal·lació i configuració dels components seleccionats en el pas anterior. Aquesta fase inclou també la realització de proves funcionals per garantir que tot funciona segons el previst.

4. **Tancament del projecte i anàlisi de resultats**: L'última fase del projecte consisteix en el tancament formal del treball, que inclou l'anàlisi dels resultats obtinguts, la redacció de conclusions, l'elaboració d'un glossari i altres retocs finals. A més, es prepararà una demostració del sistema implementat per mostrar-ne el funcionament.

Cada una d'aquestes etapes està dissenyada per assegurar que el projecte avança de manera estructurada i coherent, des de la conceptualització inicial fins a la implementació final, facilitant així la gestió dels riscos i permetent ajustos dinàmics segons les necessitats que sorgeixin durant el desenvolupament.

1.5. Planificació del treball

Per a planificar les tasques a realitzar, s'ha dissenyat un diagrama de Gantt amb una estimació en dies de la duració de cada tasca. La taula amb les dates exactes és així:

Nom	Inici	Final	Duració
Fita 1 - Pla de treball	2024-03-01	2024-03-12	12 dies
Objectius del Treball	2024-03-01	2024-03-12	12 dies
Impacte en sostenibilitat, ètic-social i de diversitat	2024-03-01	2024-03-12	12 dies
Enfocament i mètode seguit	2024-03-01	2024-03-12	12 dies
Breu sumari de productes obtinguts	2024-03-01	2024-03-12	12 dies
Pla de treball	2024-03-01	2024-03-12	12 dies
Context i justificació del Treball	2024-03-01	2024-03-12	12 dies
Breu descripció dels altres capítols de la memòria	2024-03-01	2024-03-12	12 dies
Fita 2 - Estudi de components i infraestructura AWS	2024-03-13	2024-04-09	28 dies
Materials i mètodes	2024-03-13	2024-03-15	3 dies
Decidir metodologia per al projecte	2024-03-13	2024-03-14	2 dies
Què és k8s? Per a què serveix i per a què no serveix.	2024-03-14	2024-03-15	2 dies
Què és EKS? En què es diferencia?	2024-03-16	2024-03-17	2 dies
La xarxa a k8s – AWS VPC CNI, Cilium, Calico	2024-03-17	2024-03-24	8 dies
El DNS a k8s - Kube-dns, CoreDNS, NodeLocal DNS Cache	2024-03-22	2024-03-24	3 dies
L'escalabilitat de pods - Keda, HPA i VPA	2024-03-25	2024-03-28	4 dies
L'escalabilitat de nodes - Karpenter i cluster-autoscaler	2024-03-26	2024-03-28	3 dies
Investigació IaC AWS - Terraform, Cloudformation i Sceptre	2024-03-30	2024-03-31	2 dies
Investigació infraestructura necessària per clúster k8s	2024-04-01	2024-04-03	3 dies
Creació infraestructura per clúster	2024-04-04	2024-04-09	6 dies

Fita 3 - Clúster de K8s i proves tècniques	2024-04-10	2024-05-05	26 dies
Investigació IaC cluster K8s - Terraform, Eksctl	2024-04-10	2024-04-11	2 dies
Creació del cluster de k8s	2024-04-11	2024-04-14	4 dies
Investigació IaC paquets k8s - Helm	2024-04-14	2024-04-15	2 dies
Instal·lació i configuració del gestor de xarxa	2024-04-16	2024-04-20	5 dies
Instal·lació i configuració del gestor de DNS	2024-04-21	2024-04-22	2 dies
Instal·lació i configuració de l'escalador de pods	2024-04-23	2024-04-24	2 dies
Proves funcionals escalabilitat de pods	2024-04-25	2024-04-27	3 dies
Instal·lació i configuració de l'escalador de nodes	2024-04-28	2024-04-29	2 dies
Proves funcionals escalabilitat de nodes	2024-04-29	2024-05-01	3 dies
Instal·lació i configuració del gestor de secrets	2024-05-02	2024-05-03	2 dies
Proves funcionals gestor de secrets	2024-05-04	2024-05-05	2 dies
Fita 4 - Lliurament final	2024-05-06	2024-06-11	37 dies
Anàlisi dels resultats obtinguts	2024-05-06	2024-05-14	9 dies
Conclusions	2024-05-15	2024-05-23	9 dies
Glossari	2024-05-24	2024-05-28	5 dies
Revisió i retocs finals de la memòria	2024-05-29	2024-06-11	14 dies
Bibliografia	2024-03-01	2024-06-11	103 dies
Annexos	2024-03-01	2024-06-11	103 dies

Mentre que el diagrama de seria el següent:

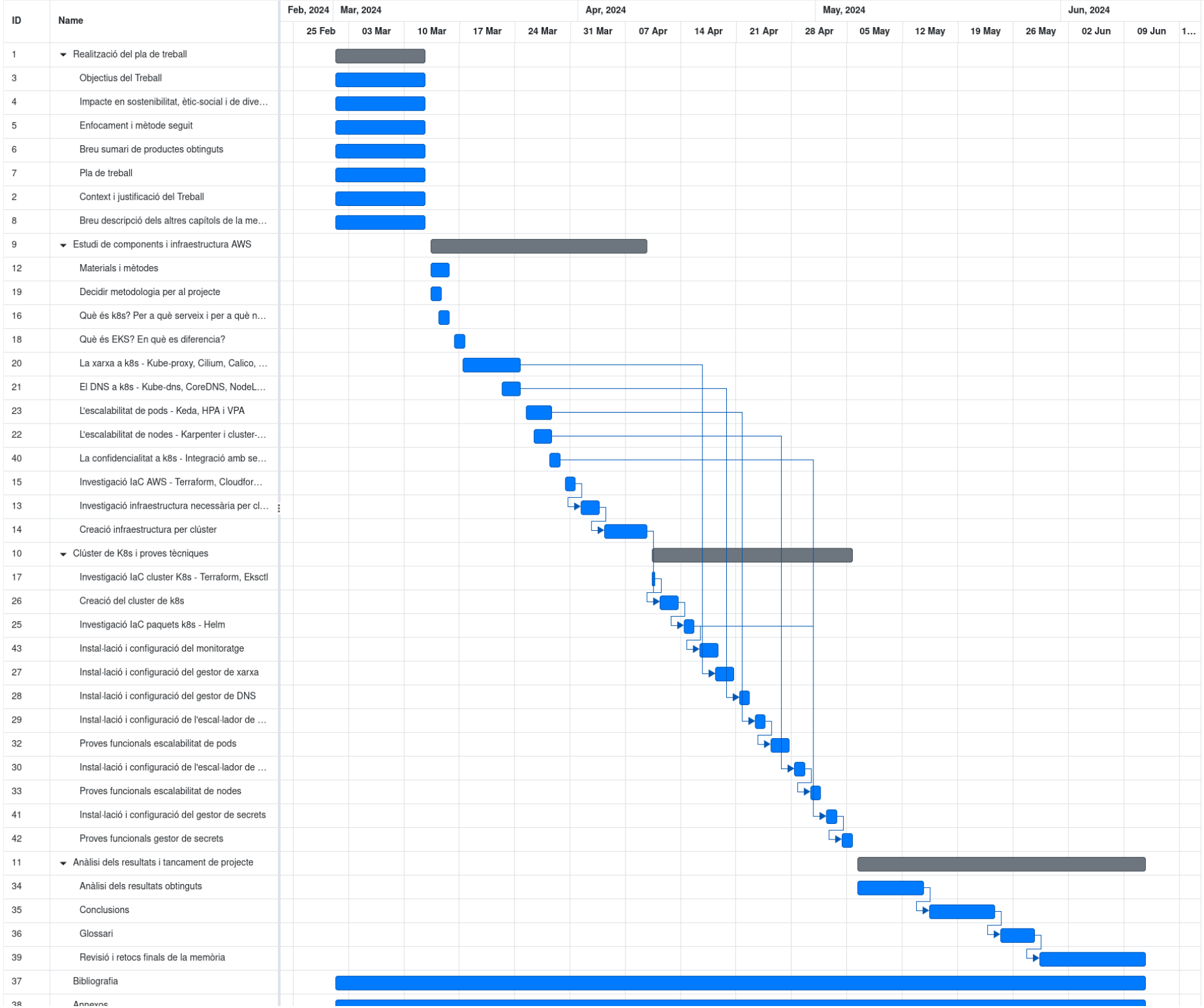


Figura 1: Diagrama de Gantt

1.6. Breu sumari de productes obtinguts

El producte final d'aquest treball es centra en la creació i desenvolupament d'un clúster de Kubernetes en AWS, demostrant l'abast i profunditat de la investigació i l'execució realitzades. Els principals *entregables* del projecte són:

1. **Disseny del clúster de Kubernetes en AWS**: Inclou una descripció detallada del disseny arquitectònic del clúster, justificant les eleccions tecnològiques i configuracions adoptades per a satisfer les necessitats específiques del projecte.
2. **Implementació del clúster de Kubernetes**: Documentació exhaustiva que cobreix cada pas del procés d'implementació, des de la configuració inicial de la infraestructura fins a la integració i ajust final dels components del clúster. Inclou codi font, scripts d'Infraestructura com a Codi (IaC) i una guia detallada d'implementació.

A més d'aquests productes clau, el projecte també proporcionarà els següents *entregables* complementaris:

- **Guia d'implementació i configuració**: Ofereix un conjunt d'instruccions detallades per facilitar la replicació o l'adaptació del clúster en altres entorns, servint com a manual de referència per a futurs projectes.
- **Comparativa de components de Kubernetes**: Presenta una anàlisi detallada comparant diferents components disponibles, destacant les raons per la selecció d'alguns sobre altres basant-se en criteris com funcionalitat, rendiment, cost i compatibilitat.
- **Resultats de proves funcionals**: Inclou un informe amb els resultats obtinguts en les proves de rendiment, escalabilitat i seguretat del clúster implementat, proporcionant una valoració objectiva de la seva eficàcia en un entorn de producció.

1.7. Breu descripció dels altres capítols de la memòria

El treball estarà estructurat en capítols que reflecteixen les fases clau del desenvolupament del projecte, cadascun enfocat en aspectes diferents de la implementació d'un clúster de Kubernetes en AWS:

- **Materials i mètodes**: Descriu els objectius del projecte, justificació i la metodologia seleccionada per a l'execució del treball.
- **Estudi preliminar**: Presenta una visió general dels components i tecnologies clau, incloent Kubernetes, Amazon EKS, i eines d'Infraestructura com a Codi (IaC), establint el context per a l'anàlisi i selecció de les solucions adoptades.

- **Anàlisi i disseny**: Detalla el procés de planificació i disseny del clúster, incloent l'anàlisi de requisits, la selecció de components i la configuració de la infraestructura d'AWS.
- **Implementació**: Explica com s'ha dut a terme la creació del clúster de Kubernetes i la instal·lació dels components i *addons* seleccionats, així com les proves funcionals realitzades.
- **Anàlisi dels resultats**: Compila els resultats obtinguts després de la implementació i les proves, oferint una valoració de l'eficàcia de la solució desplegada.
- **Conclusions i treballs futurs**: Recapitula els principals aprenentatges, els èxits i limitacions del projecte, i proposa línies de recerca o desenvolupament futures basades en l'experiència adquirida.

Cada secció de la memòria aporta una visió integral del procés seguit en el projecte, des de les fases inicials de conceptualització fins a l'execució pràctica i la posterior anàlisi de resultats.

2. Materials i mètodes

2.1. Aspectes més rellevants del disseny i desenvolupament del treball

Per a l'èxit del projecte de disseny i implementació d'un clúster de Kubernetes en AWS, és crucial atendre meticulosament als aspectes clau del seu disseny i desenvolupament. Aquests elements fonamentals no només defineixen l'estructura i funcionalitat del clúster, sinó que també asseguren que la solució final sigui robusta, escalable i segura, alineant-se amb les necessitats específiques de l'empresa. A continuació, detallem els components més significatius d'aquest procés. [2] [3]

Selecció de la plataforma i serveis: El primer pas crític involucra l'elecció d'AWS EKS com a plataforma per al desplegament del clúster de Kubernetes, una decisió que destaca per la seva capacitat de simplificar significativament l'administració del clúster i la seva integració amb altres serveis AWS. Aquesta elecció ha de ser justificada, ponderant avantatges com la gestió automàtica del clúster, seguretat reforçada i escalabilitat, front a altres opcions del mercat.

Infraestructura com a codi (IaC): Implementar la infraestructura com a codi mitjançant eines com Terraform o AWS CloudFormation és un pilar fonamental d'aquest projecte. Això no només facilita la gestió i replicació de l'entorn de manera eficient i amb menys errors, sinó que també promou pràctiques de desenvolupament sostenible i escalable. La selecció d'una eina sobre l'altra dependrà de factors com la compatibilitat amb els serveis AWS, la facilitat d'ús i la comunitat de suport.

Seguretat i conformitat: Garantir la seguretat del clúster des de la seva concepció és imprescindible. Això inclou la configuració detallada de la gestió d'identitats i accés (IAM), les polítiques de xarxa i el compliment de les normatives aplicables. La seguretat ha de ser tractada com una capa integral del disseny del clúster, no com un afegitó posterior.

Selecció i configuració de components: L'anàlisi detallada i la selecció de components específics de Kubernetes com solucions de xarxa (Cilium, Calico), serveis de resolució de noms (CoreDNS), i eines per a l'escalabilitat (Keda, HPA per pods i Karpenter o Cluster Autoscaler per nodes) són essencials. Cada component ha de ser avaluat en termes d'eficiència, compatibilitat amb l'entorn AWS EKS i les necessitats específiques de l'empresa, buscant trobar el millor equilibri entre prestacions, costos i mantenibilitat.

2.2. Metodologia del projecte

Per afrontar els reptes específics d'un projecte de migració a l'arquitectura de Kubernetes dins d'AWS, s'ha seleccionat una metodologia *agile*. Aquest enfocament pot ser particularment adequat per al desenvolupament i implementació de clústers de Kubernetes en entorns del núvol, on la flexibilitat i l'adaptabilitat són clau. Les consideracions preses en compte són les següents:

- **Iteracions curtes i una entrega continua**: En concordança amb els principis d'*agile*, el projecte es desenvoluparà en esprints curts, permetent una entrega anticipada i regular de valor. Aquest enfocament facilita la implementació incremental de la infraestructura i els components de Kubernetes, ajustant-se a les necessitats específiques de l'entorn d'AWS i Kubernetes.
- **Adaptabilitat als canvis**: La naturalesa d'un clúster de Kubernetes, que es caracteritza per la seva escalabilitat i la gestió dinàmica de recursos, ressona amb la flexibilitat que *agile* ofereix. Això permet realitzar ajustos en el disseny i la configuració del clúster a mesura que s'identifiquen noves necessitats o millores, aprofitant la capacitat d'AWS per adaptar-se ràpidament als canvis en les càrregues de treball i en les demandes del mercat.
- **Automatització i integració continua**: La implementació d'estratègies de *CI* es complementa perfectament amb la metodologia *agile*, especialment en el context de Kubernetes. L'enfocament en la automatització, tant en el desplegament de la infraestructura com a codi (IaC) com en les proves i la gestió de configuracions, permet una integració i lliurament continu de canvis, minimitzant els riscos i millorant la qualitat del clúster final.

Aquesta metodologia *agile*, aplicada al context de la migració i gestió de clústers de Kubernetes en AWS, ofereix un marc de treball robust per al desenvolupament ràpid i adaptatiu, garantint al mateix temps que el projecte pot respondre eficientment als canvis i desafiaments inherents als entorns de núvol i la orquestració de contenidors amb Kubernetes.

3. Estudi dels components

3.1. Què és Kubernetes?

K8s, o Kubernetes, és un sistema d'orquestració de contenidors obert i extensible que permet automatitzar el desplegament, l'escalat i la gestió d'aplicacions contenidoritzades. Va ser originalment desenvolupat per Google i ara és mantingut per la Cloud Native Computing Foundation. Kubernetes es converteix en una eina clau per a les empreses que busquen aprofitar l'escalabilitat i l'agilitat ofertes per la computació en el núvol. [4]

Figura 2: Logotip de Kubernetes

Per a què serveix Kubernetes:

1. **Automatització de desplegaments**: Kubernetes permet als usuaris desplegar automàticament les seves aplicacions contenidoritzades en un clúster de servidors, gestionant la distribució i l'execució dels contenidors.
2. **Escalabilitat**: Ofereix eines per escalar les aplicacions de manera automàtica o manual, permetent ajustar els recursos necessaris segons la demanda.
3. **Gestió de càrrega de treball**: Permet distribuir la càrrega de treball de manera eficient entre els contenidors i els nodes del clúster, assegurant l'alta disponibilitat i l'eficiència.
4. **Descoberta de serveis i balanceig de càrrega**: Kubernetes pot gestionar el tràfic d'entrada als contenidors, proporcionant serveis de descoberta i balanceig de càrrega per a les aplicacions.
5. **Automatització de la recuperació davant errors**: Pot reiniciar automàticament els contenidors que fallen, substituir-los i reubicar-los en altres nodes si és necessari.

Per a què no serveix Kubernetes:

1. **Gestió de dades específiques de l'aplicació**: Kubernetes gestiona la infraestructura a nivell de contenidors, però no s'encarrega de la lògica interna o la gestió de dades específiques de les aplicacions. Aquestes tasques continuen sent responsabilitat dels desenvolupadors de l'aplicació.
2. **Substitut directe de màquines virtuals**: Encara que Kubernetes pot executar contenidors que aïllen les aplicacions igual que les màquines virtuals, no substitueix la funcionalitat completa o la seguretat ofertes per aquestes últimes a nivell de sistema operatiu.

3. **Solucions simples d'allotjament web**: Per a petits projectes o aplicacions web simples, l'ús de Kubernetes pot ser més complex i costós del necessari. En aquests casos, solucions d'allotjament més simples o plataformes com a servei (PaaS) podrien ser més adequades.

La gestió de Kubernetes es duu a terme mitjançant un servidor API HTTP que el propi clúster exposa, amb la qual tant els usuaris finals com les aplicacions es podran comunicar els uns amb els altres. Aquestes interaccions amb la API es podran dur a terme utilitzant simples trucades REST, però ja existeixen eines que ens permetran simplificar aquestes trucades, com *kubectl*, o altres llibreries. [5]

Resumint tot, Kubernetes és una poderosa eina per a la gestió de contenidors a gran escala, però és important entendre els seus límits i assegurar-se que s'adapta a les necessitats específiques de cada projecte abans de decidir implementar-lo.

3.1.1. L'arquitectura de Kubernetes

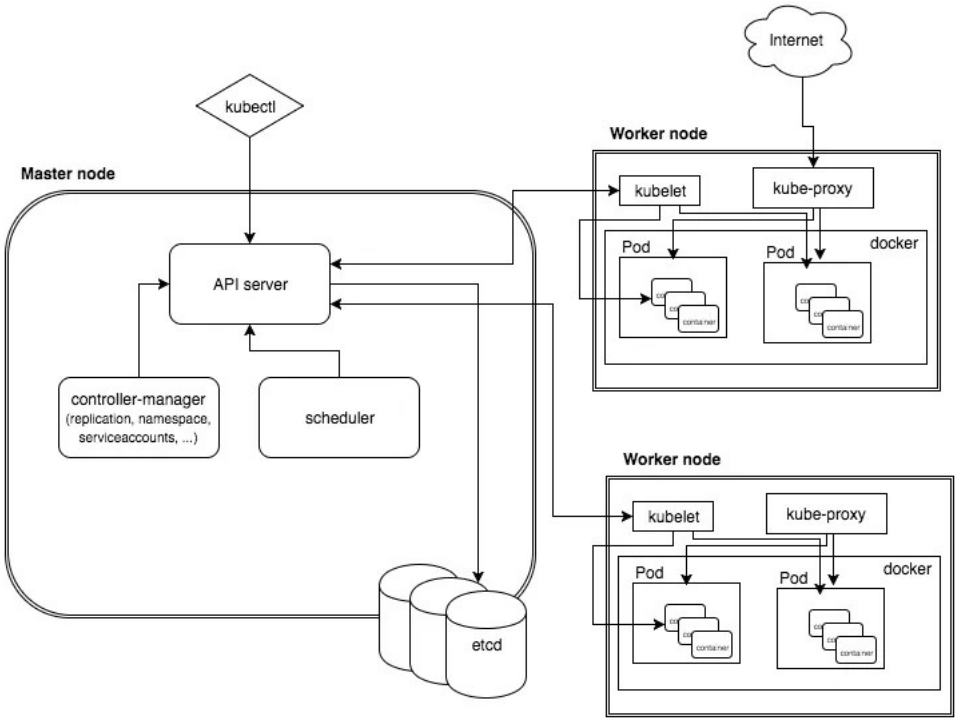


Figura 3: Arquitectura de Kubernetes

Una visió molt general de l'arquitectura de Kubernetes, sense cap extensió ni modificació extra, estaria formada per els següents components:[6]

1. **Node mestre**: Actuen com els «cervells» del clúster i són els responsable de prendre decisions globals sobre el clúster, com ara quan aixecar noves aplicacions o quan gestionar esdeveniments de nodes. Un node mestre conté diversos components:
    - **Servidor API**: És el punt d'entrada de les comandes de Kubernetes. Serveix l'API de Kubernetes i gestiona i emmagatzema l'estat del clúster en etcd.
    - **Planificador**: Determina en quin Node s'han de desplegar els nous contenidors basant-se en la disponibilitat de recursos, requisits de política, afinitats, etc.
    - **Gestor de Controladors**: Executa els controladors de lògica de negocis, que monitoritzen l'estat del clúster i fan canvis cap a l'estat desitjat.
    - **etcd**: Emmagatzema tota la informació del clúster de manera persistent i consistent, actuant com a font de veritat per l'estat del clúster.
2. **Node treballador**: Són les màquines que executen les aplicacions i els serveis contenidoritzats. Cada node està gestionat per un node mestre i conté els següents components:
    - **Kubelet**: És l'agent que s'executa en cada node, s'assegura que els contenidors s'executen en un Pod.
    - **Kube-proxy**: Manté les regles de xarxa en els nodes, permetent la comunicació amb els contenidors des de dins o fora del clúster. Aquest component pot ser reemplaçat per altres gestors de xarxa.
    - **Container Runtime**: És el programari responsable d'executar els contenidors. *Dockershim* n'era el més utilitzat, però Kubernetes suporta altres *runtimes* com l'actual estàndard de mercat, *containerd*.
3. **Pod**: És la unitat més petita que es pot desplegar i gestionar en Kubernetes. Un Pod pot contenir un o més contenidors que comparteixen la mateixa adreça IP, volums de dades i altres recursos.

Aquesta és una visió molt general de l'arquitectura de Kubernetes. La part positiva d'aquest sistema rau en la seva flexibilitat i en la capacitat d'automatitzar el desplegament, l'escalat i la gestió d'aplicacions en contenidors. Més endavant analitzarem possibles reemplaçaments d'alguns d'aquests components, i d'altres completament de nous.

3.2. Què és EKS? Les principals diferències amb Kubernetes

EKS (Elastic Kubernetes Service) [7] és un servei gestionat per Amazon que permet executar Kubernetes a AWS sense necessitat d'instal·lar, operar o mantenir el teu propi control de Kubernetes al núvol. EKS s'encarrega de tasques difícils de gestió com la instal·lació, l'operació i l'actualització del software de Kubernetes, permetent així concentrar-te en el desplegament i la gestió d'aplicacions.

Aquest servei proporciona un entorn Kubernetes altament disponible i segur, que s'integra amb altres serveis d'AWS per oferir escalabilitat, rendiment i seguretat. Per exemple, pots utilitzar, Amazon ECR per emmagatzemar les imatges de Docker, i AWS Identity and Access Management (IAM) per controlar l'accés a recursos, entre altres coses.

Ara, en comparació amb Kubernetes «vanilla» o una instal·lació de Kubernetes auto-gestionada hi ha aquestes principals diferències: [8]

1. **Gestió de la infraestructura**: Amb EKS, AWS s'encarrega de la gestió del panell de control de Kubernetes, incloent l'actualització i la seguretat d'aquest. En una instal·lació pròpia de Kubernetes, has de gestionar i actualitzar tu mateix el panell de control. Tota la lògica referent als nodes mestres és gestionada per AWS.
2. **Seguretat**: EKS està integrat amb AWS IAM, oferint un mecanisme de control d'accés molt refinat i potent per als recursos de Kubernetes. En una instal·lació pròpia, hauries de configurar i mantenir els controls d'accés manualment.
3. **Escalabilitat i disponibilitat**: EKS gestiona l'alta disponibilitat del panell de control de Kubernetes a través de diverses zones de disponibilitat (availability zones), cosa que no és tan directe en una configuració pròpia, on has de planificar i implementar la teva pròpia estratègia de alta disponibilitat.
4. **Integració amb serveis AWS**: EKS està profundament integrat amb altres serveis d'AWS, com per exemple VPC, AWS Fargate, i AWS Outposts, permetent un desplegament més suau i una millor interoperabilitat entre serveis. Amb Kubernetes auto-gestionat, hauries d'integrar manualment aquests serveis.
5. **Cost**: Amb EKS, es paga per el servei de gestió del panell de control i els recursos que utilitzes (com instàncies EC2 o nodes Fargate). En una instal·lació pròpia de Kubernetes, no pagues pel software de Kubernetes en si, però els costos operatius poden ser significatius, depenent de la mida i complexitat de la teva infraestructura.

3.3. La xarxa a Kubernetes

Entendre la xarxa dins de Kubernetes és fonamental per a la implementació eficaç i la gestió d'aplicacions en contenidors d'imatges en un entorn distribuït. La xarxa de Kubernetes permet que els pods (un o més contenidors desplegats junts) comuniquin entre si i amb l'exterior de manera segura i eficient. A diferència d'altres sistemes de contenidors, on la comunicació es basa sovint en vinculacions de ports específics al host, Kubernetes assigna a cada pod una adreça IP única, la qual facilita una comunicació més directa i simplificada. [9]

La complexitat de la xarxa a Kubernetes es gestiona mitjançant l'ús de Serveis, que proporcionen un punt d'accés constant als pods, independentment de les seves ubicacions dins del clúster. Aquests serveis permeten a les aplicacions descobrir i comunicar-se entre si a través de DNS intern o balancejadors de càrrega, ocultant la complexitat i el canvi constant dels entorns de producció. [10]

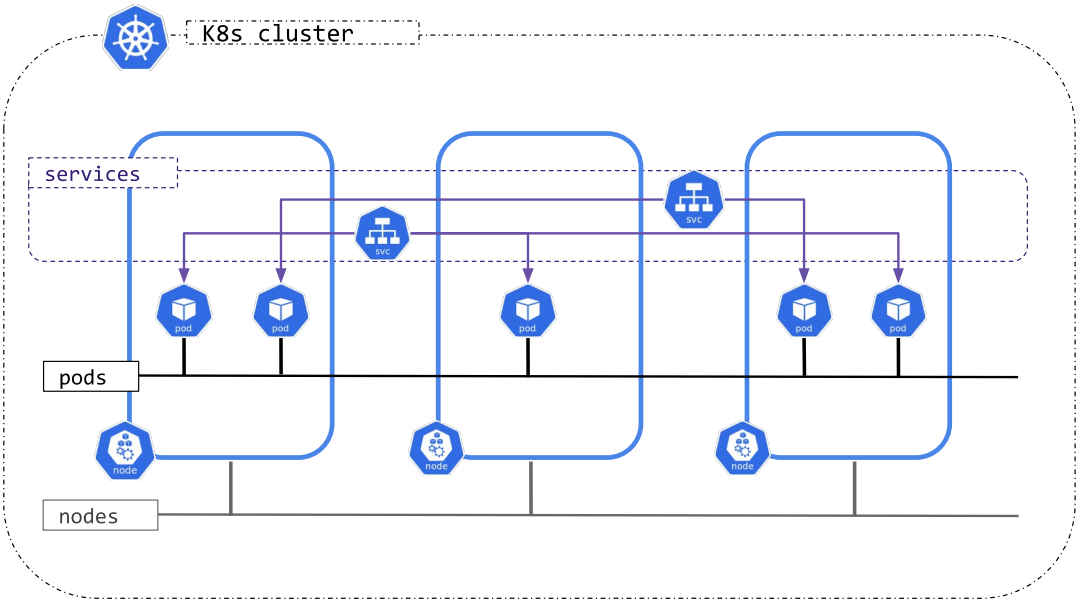


Figura 4: La xarxa en un clúster de Kubernetes

Per a la gestió i configuració d'aquest ecosistema de xarxa, s'utilitzen diverses eines que ajuden a satisfer les necessitats específiques de seguretat, observabilitat i rendiment. Algunes d'aquestes eines inclouen solucions de xarxa de superposició, *plugins* de xarxa basats en estàndards CNI (requerits per implementar el model de xarxa de Kubernetes [11]), i utilitats específiques de cada núvol que optimitzen la integració amb infraestructures d'aquests. Les opcions varien des de solucions de xarxa altament personalitzables fins a

integracions específiques de proveïdors de serveis en núvol, cadascuna amb el seu conjunt de característiques, avantatges i inconvenients. [12]

Abans de començar la revisió dels components específics de Kubernetes, és important destacar que hi ha un conjunt de components clau que són crucials per a l'èxit d'aquest projecte. L'avaluació detallada d'aquests components permetrà seleccionar les millors opcions per al desenvolupament i implementació del clúster de K8s en AWS. Així, en els apartats següents, es realitzarà un anàlisi exhaustiu de cada component, tenint en compte criteris com la funcionalitat, el rendiment, el cost i la compatibilitat amb l'entorn AWS EKS.

3.3.1. Anàlisi dels diferents plugins de xarxa a Kubernetes

En el context de desplegar un clúster Kubernetes utilitzant Amazon EKS, s'analitzen diferents tecnologies de xarxa per determinar la solució més eficaç en termes de rendiment, seguretat, i gestió de la xarxa. Les tecnologies que s'han analitzat són Amazon VPC CNI conjuntament amb Kube-proxy, Cilium, i Calico, cadascuna oferint diferents avantatges i limitacions.

Objectius i requisits tecnològics:

L'objectiu és seleccionar una implementació de xarxa que optimitzi la integració, escalabilitat, i observabilitat del clúster Kubernetes, alhora minimitzant la complexitat i els costos operacionals. Els requisits específics són:

- **Compatibilitat amb AWS:** El *plugin* ha de ser plenament compatible amb els serveis AWS i ha d'integrar-se sense problemes amb la infraestructura existent.
- **Alta disponibilitat i rendiment:** Ha de garantir una alta disponibilitat i un excel·lent rendiment per a les aplicacions en el clúster.
- **Seguretat:** Ha d'oferir característiques avançades de seguretat que protegeixin les comunicacions dins del clúster.
- **Escalabilitat:** Ha de ser capaç d'escalar de manera eficient a mesura que el clúster creix.
- **Manteniment i suport:** Preferiblement, ha de tenir un bon suport comunitari i actualitzacions regulars per part dels desenvolupadors.

Amazon VPC CNI i Kube-proxy

Amazon VPC CNI: [13]

- **Funcionalitat**: Assigna adreces IP natives del VPC d'AWS als pods, permetent una comunicació eficient tant dins com fora del clúster.
- **Avantatges**:
    - Integració nativa amb AWS, simplificant la gestió de xarxa.
    - Permet l'aplicació directa de polítiques de seguretat AWS als pods.
- **Inconvenients**:
    - Cada pod necessita una adreça IP del VPC, el que pot esgotar ràpidament l'espai d'adreçament disponible en xarxes grans, especialment en clústers de gran escala.
    - Complexitat en la gestió de múltiples ENIs i IPs per cada node.

Kube-proxy: [14]

- **Funcionalitat**: Gestiona l'accés del tràfic als serveis dins del clúster, utilitzant IPTABLES, IPVS, o altres mecanismes de redireccionament de trànsit del kernel Linux.
- **Avantatges**:
    - Configuració flexible amb suport per múltiples modes de funcionament.
    - Contribueix a l'equilibri de càrrega i la resiliència dels serveis, distribuint les peticions entrants de manera equitativa entre tots els pods d'un servei.
    - Pot establir polítiques d'accés per regular el tràfic entrant cap als serveis.
- **Inconvenients**:
    - Els modes IPTABLES, IPVS... poden ser un coll d'ampolla especialment en clústers amb tràfic de xarxa intens.
    - Dependent de característiques específiques del nucli Linux, que poden ser limitants.

Cilium

- **Funcionalitat**: Utilitza eBPF per gestionar la seguretat, xarxa, i observabilitat a nivell de kernel. [16]
- **Avantatges**:
    - Alt rendiment i escalabilitat, especialment en entorns densos i dinàmics gràcies a l'ús d'eBPF.

    - Gran capacitat d'observabilitat amb la integració nativa amb Hubble.
    - Possibilitat d'utilitzar un IPAM intern per a gestionar les IPs dels pods, eliminant així en gran mesura la limitació d'ús d'IPs en un VPC.
- **Inconvenients**:
    - Complexitat tecnològica que pot requerir coneixements especialitzats en el funcionament d'eBPF.
    - Potencialment restrictiu en termes de compatibilitat del kernel degut a l'ús d'eBPF com a base del seu funcionament.

Calico

- **Funcionalitat**: Ofereix funcionalitats de xarxa de nivell 3 i una gestió extensa de polítiques de seguretat multi-entorns. [21]
- **Avantatges**:
    - No requereix encapsulació de tràfic, potencialment millorant el rendiment.
    - Ampli suport per polítiques de seguretat detallades i configurables.
    - Integració nativa amb Prometheus per proporcionar un monitoratge detallat.
- **Inconvenients**:
    - Pot ser molt complex de configurar i gestionar, especialment en entorns multinúvol o híbrids.

En l'**Annex 1: Anàlisi detallat de les solucions de xarxa a Kubernetes** s'analitza i s'explica més en detall què és i com funciona cada component de xarxa mencionat.

3.3.2. Conclusions i decisions

Després d'analitzar les diferents opcions, s'ha escollit Cilium com la millor solució per al projecte per les següents raons:

- **Observabilitat detallada**: La integració amb Hubble permet una visibilitat en temps real que és crítica per diagnosticar i resoldre problemes ràpidament.

- **Rendiment i escalabilitat**: Cilium, amb la seva base en eBPF, proporciona un rendiment significativament superior sense els colls d'ampolla associats amb solucions basades en iptables, com Kube-proxy.
- **Simplificació de la gestió de xarxa**: Cilium elimina la necessitat d'utilitzar múltiples ENIs per node, una limitació clau de l'Amazon VPC CNI en clústers grans.
- **Seguretat avançada**: Les polítiques de seguretat basades en identitats de Cilium són més flexibles i adaptatives comparades amb les basades en IPs estàtiques, oferint millor seguretat en entorns de micro-serveis.
- **Simplicitat davant Calico**: Calico és una solució especialment útil en entorns multi-núvol o híbrids, degut a la seva alta compatibilitat amb aquests, però fent-lo més complex. Cilium no té una compatibilitat tan alta però per al nostre cas tampoc ens fa falta amb únicament un clúster de Kubernetes a gestionar.

3.4. El DNS a Kubernetes

Objectius i requisits tecnològics:

El principal objectiu és utilitzar un sistema de resolució de noms robust i eficient dins del clúster de Kubernetes que sigui capaç de manejar les peticions de manera automàtica i sense interrupcions. Els requisits específics inclouen:

- **Alta disponibilitat:** El sistema DNS ha de garantir una disponibilitat contínua, sense caigudes que afectin l'accés als serveis del clúster.
- **Escalabilitat:** El sistema ha de ser capaç d'adaptar-se a l'augment de càrrega de treball sense degradar el rendiment.
- **Baixa latència:** La resolució de noms ha de ser ràpida per no impactar negativament en el temps de resposta dels serveis.
- **Fàcil gestió:** El sistema ha de ser fàcil de configurar i mantenir, integrant-se bé amb les eines de gestió existents del clúster.

3.4.1. CoreDNS com a estàndard de mercat

La resolució de noms a Kubernetes tradicionalment s'ha realitzat utilitzant KubeDNS, però ha estat deprecat a partir de la versió 1.13 de Kubernetes a favor de CoreDNS, especialment per ser mono-procés, més flexible, i escrit en Golang, fent-lo així segur amb l'ús de memòria (KubeDNS és escrit en C). [23]

CoreDNS destaca per la seva flexibilitat i extensibilitat, gràcies a la seva arquitectura de plugins. Això permet als usuaris afegir o modificar funcionalitats segons les seves necessitats específiques. Un exemple d'això és la possibilitat d'integrar-se amb altres serveis de descoberta de serveis o bases de dades per a la resolució de noms. [24]

A més, aquest està dissenyat per operar eficientment en entorns distribuïts i de contenidors, com Kubernetes, oferint una resolució de noms ràpida i fiable per a serveis i pòds dins del clúster. És capaç de gestionar les consultes DNS tant internes com externes del clúster, fent-lo una solució integral per a les necessitats de noms de domini dins de Kubernetes.

La configuració de CoreDNS a Kubernetes es realitza a través del ConfigMap de CoreDNS, que permet als administradors del clúster gestionar fàcilment el comportament del servidor DNS. Aquesta configuració pot incloure l'addició de plugins de CoreDNS, la definició de regles de reescriptura de consultes DNS, i la configuració de polítiques de caché, entre altres. [25]

Per aquestes raons seguirem l'actual estàndard de mercat i utilitzarem CoreDNS.

3.5. L'escalabilitat de pods

Objectius i requisits tecnològics:

L'objectiu principal d'aquesta secció és seleccionar un sistema d'escalabilitat per a pods que sigui eficient, automàtic i que s'ajusti a les fluctuacions de la demanda dins del clúster de Kubernetes. Els requisits específics són:

- **Automatització:** L'escalabilitat dels pods ha de ser completament automàtica, sense necessitat d'intervenció manual en el procés d'escalat.
- **Eficiència de recursos:** Optimitzar l'ús de recursos, assegurant que els pods utilitzin només els recursos necessaris en cada moment.
- **Temps de resposta ràpid:** L'escalat ha de ser prou ràpid per gestionar augments sobtats de càrrega sense afectar el rendiment de les aplicacions.
- **Integració amb serveis AWS:** Utilització de serveis AWS per millorar l'escalabilitat i la gestió de càrregues de treball.

3.5.1. HPA

L'Horizontal Pod Autoscaler (HPA) és una característica de Kubernetes que permet ajustar automàticament el nombre de pods en un desplegament, un conjunt de rèpliques, o un altre objecte que suporti l'escalat, basant-se en l'observació de mètriques específiques internes del clúster com l'ús de la CPU o la memòria. [27] No és per tant adequat al nostre cas d'ús degut a la falta d'integració amb serveis AWS.

3.5.2. VPA

El Vertical Pod Autoscaler (VPA) és una funcionalitat dins de Kubernetes que automatitza el procés d'ajustar els recursos de CPU i memòria assignats als pods. A diferència de l'Horizontal Pod Autoscaler (HPA), que incrementa o disminueix el nombre de pods per adaptar-se a les càrregues de treball, el VPA modifica els límits i les sol·licituds de recursos dels pods existents per optimitzar el seu ús de recursos i rendiment. Aquesta adaptació assegura que els pods tinguin els recursos necessaris per funcionar de manera eficaç, evitant al mateix temps el sobre-provisionament i la infrautilització de recursos. [28] Igual que l'HPA, aquesta funcionalitat no s'adapta a serveis externs a AWS.

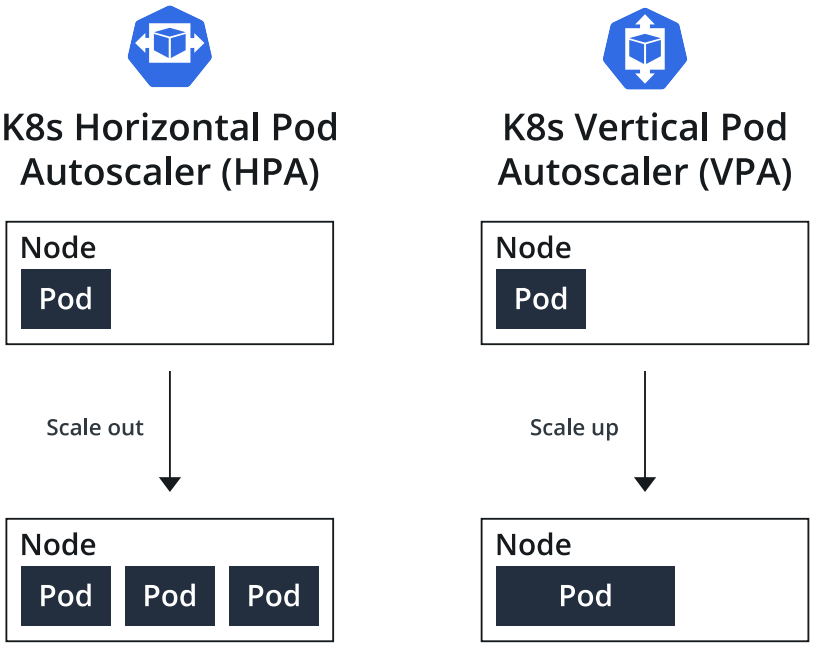


Figura 5: Diferències entre HPA i VPA

3.5.3. Keda

Keda, a diferència de l'HPA i el VPA, no es basa en l'ús de CPU o memòria dels pods, si no en esdeveniments que seran definits per nosaltres, com per exemple trucades a una cua, o en l'ús d'una base de dades, permetent escalar el nombre de pods en base a esdeveniments. [29]

3.5.4. Conclusions i decisions

Com hem definit en l'objectiu tecnològic, la nostra meta és automatitzar l'escalat de pods basant-nos en recursos d'AWS. En base a aquest requeriment, Keda és l'eina que ens proporciona aquesta funcionalitat. Les proves posteriors les realitzarem utilitzant Keda. Per als altres requisits específics, valorarem si Keda els compleix en les proves pràctiques.

3.6. L'escalabilitat de nodes

Objectius i requisits tecnològics:

L'objectiu d'aquesta secció és seleccionar un sistema d'escalabilitat automàtic per als nodes del clúster de Kubernetes que sigui eficaç, dinàmic i econòmic. Els requisits específics són els següents:

- **Automatització completa:** L'escalabilitat dels nodes ha de ser completament automàtica, adaptant-se dinàmicament als canvis en les necessitats de recursos sense intervenció manual.
- **Eficiència de costos:** Minimitzar els costos operacionals assegurant que els nodes s'afegeixin només quan sigui estrictament necessari i es retirin quan no es requereixin.
- **Ràpida resposta a la demanda:** El sistema ha de poder escalar ràpidament cap amunt o cap avall en resposta a les fluctuacions de la demanda.

3.6.1. Cluster autoscaler

El Cluster autoscaler és l'eina per defecte que gestiona automàticament l'escalat de nodes dins d'un clúster de Kubernetes. Aquesta eina assegura que hi hagi prou nodes per executar tots els pods eficientment sense malbaratar recursos. Funciona avaluant contínuament l'estat del clúster, incloent la càrrega de treball i la utilització dels recursos, per determinar quan és necessari afegir o eliminar nodes. [30]

Funcionalitat principal:

- **Avaluació de necessitats**: Determina les necessitats de recursos dels pods que no poden ser programats per falta d'espai i incrementa el nombre de nodes adequadament.
- **Optimització de costos**: Redueix el nombre de nodes quan són infrautilitzats, optimitzant així els costos operacionals.

Beneficis:

- **Gestió dinàmica de recursos**: Manté l'eficiència operativa, adaptant el nombre de nodes a les necessitats reals del clúster.

- **Reducció de despeses**: Escala els nodes d'acord amb les necessitats reals, evitant el sobre-provisionament i el malbaratament de recursos.

3.6.2. Karpenter

Desenvolupat per AWS, Karpenter està dissenyat per optimitzar l'ús dels recursos mitjançant una lògica d'escalat més proactiva i un temps de resposta més curt. A diferència del cluster autoscaler, Karpenter utilitza tant l'estat actual del clúster com prediccions sobre les necessitats futures de recursos per fer decisions més informades i proactives sobre l'escalat. [31]

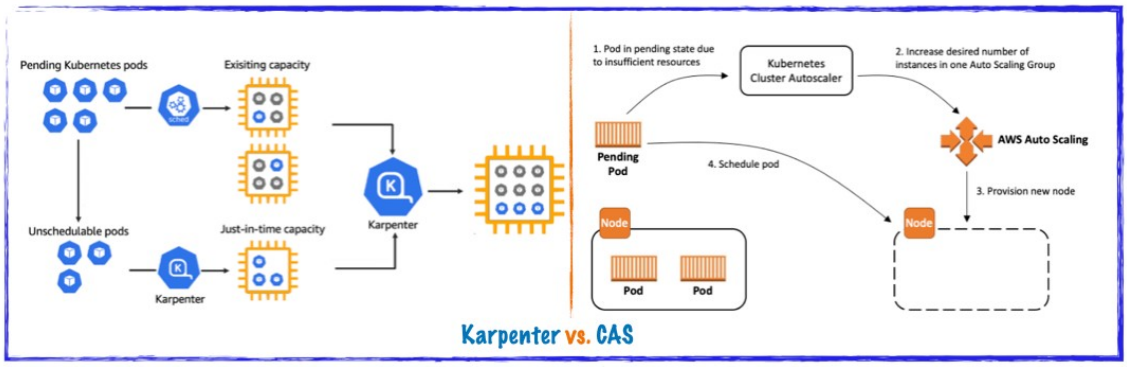


Figura 6: Diferències funcionals entre Karpenter i CAS

Funcionalitat principal:

- **Aprovisionament ràpid**: Aprovisiona nodes en qüestions de segons, permetent una adaptació ràpida a les fluctuacions en la demanda de recursos.
- **Selecció intel·ligent d'instàncies**: Utilitza un algoritme que considera la disponibilitat, cost, eficiència energètica, i compatibilitat amb les càrregues de treball per trobar la millor correspondència per als pods.

Beneficis:

- **Adaptabilitat ràpida**: Permet als clústers adaptar-se ràpidament a canvis en la demanda amb mínim temps de resposta.
- **Eficiència de costos**: Optimitza els costos operatius seleccionant tipus d'instància que millor s'ajusten a les necessitats reals dels pods, reduint així el malbaratament de recursos.

3.6.3. Conclusions i decisions

En un entorn d'AWS per al clúster de Kubernetes, s'ha decidit utilitzar Karpenter com a eina d'escalat de nodes ja que hauria de complir els nostres

objectius i requisits d'automatització, eficiència de costos i resposta ràpida a la demanda, juntament amb una integració nativa amb AWS.

3.7. L'IaC en AWS

Objectius i requisits tecnològic:

L'objectiu tecnològic és implementar l'ús de la infraestructura com a codi a AWS per automatitzar de manera eficient el desplegament i la gestió dels recursos de la infraestructura mitjançant scripts o definicions de codi. L'objectiu és millorar la consistència, la repetibilitat i la seguretat dels desplegaments, alhora que es redueixen els errors humans i es facilita la gestió d'infraestructures complexes a AWS.

Dins l'ecosistema d'IaC en AWS, hi ha diverses eines i serveis dissenyats per implementar IaC, sent *Sceptre* i *Terraform* dos dels més prominents.

3.7.1. Terraform

Terraform és una eina oberta multi-plataforma que permet als usuaris definir i provisionar infraestructura a través de codi de configuració declaratiu. Utilitza els seus propis fitxers de configuració per descriure els recursos necessaris, permetent així crear, modificar i gestionar recursos d'infraestructura en diverses plataformes cloud, incloent AWS. Terraform segueix un enfocament declaratiu, on els usuaris especifiquen "què" volen aconseguir, sense necessàriament definir "com" aconseguir-ho. [32]

3.7.2. Sceptre

Sceptre, per la seva part, és una eina per gestionar i desplegar plantilles amb CloudFormation. CloudFormation és l'eina d'IaC oficial d'AWS i permet als usuaris definir la majoria dels recursos d'AWS i les seves relacions en fitxers YAML, facilitant la creació, actualització i destrucció d'entorns relacionats. Sceptre simplifica el treball amb CloudFormation, gestionant l'ordre de creació, actualització i eliminació de recursos, i permetent la inserció de dades dinàmiques en les plantilles. Aquesta eina s'integra bé amb altres eines i serveis, fent-la molt versàtil per a equips que utilitzen AWS. [33]

3.7.3. Conclusions i decisions

En aquest projecte específic, s'ha decidit optar per **Sceptre** com a la nostra eina de IaC per diverses raons:

1. **Familiaritat i expertesa:** Sceptre és l'eina que utilitzem a l'empresa, el que significa que podem aprofitar el coneixement existent sense necessitat d'un període d'adaptació per aprendre una nova eina.
2. **Costos de migració:** Migrar a una nova eina com Terraform suposa no només un cost temporal significatiu per a l'aprenentatge i la transició sinó també el risc de possibles errors o interrupcions durant el procés de migració, mentre que mantindre diferents codis en llenguatges diferents és molt negatiu per la pèrdua de visibilitat.
3. **Integració amb CloudFormation:** Sceptre està dissenyat per treballar directament amb AWS CloudFormation, el que ens proporciona una integració nativa i eficient amb els serveis AWS, aprofitant al màxim les característiques i les funcionalitats específiques de l'AWS.

En l'Annex 2 s'explica el procés complet de com configurar i integrar Sceptre amb el nostre compte d'AWS.

4. Creació de la infraestructura per al clúster

La implementació d'un clúster de Kubernetes en AWS requereix una infraestructura robusta i segura que pugui suportar l'escalabilitat i la disponibilitat necessària per a aplicacions en producció. En aquesta secció, descriurem l'estratègia general i els components clau dissenyats per a la creació d'aquesta infraestructura, reservant els detalls específics i configuracions per al seu annex corresponent.

Objectius i requisits tecnològic:

L'objectiu principal d'aquesta secció és crear tota la infraestructura necessària per al clúster de Kubernetes en AWS, assegurant que aquesta suporti eficaçment el clúster en termes de rendiment, seguretat, i escalabilitat. Els requisits específics són:

- **Escalabilitat:** La infraestructura ha de poder escalar de manera efectiva per adaptar-se a les necessitats canviants del clúster.
- **Seguretat:** Implementar mesures de seguretat robustes per protegir el clúster contra amenaces externes i internes.
- **Alta Disponibilitat:** Assegurar que la infraestructura suporti una alta disponibilitat per a les aplicacions crítiques.
- **Optimització de Costos:** Dissenyar la infraestructura de manera que maximitzi l'eficiència del cost.

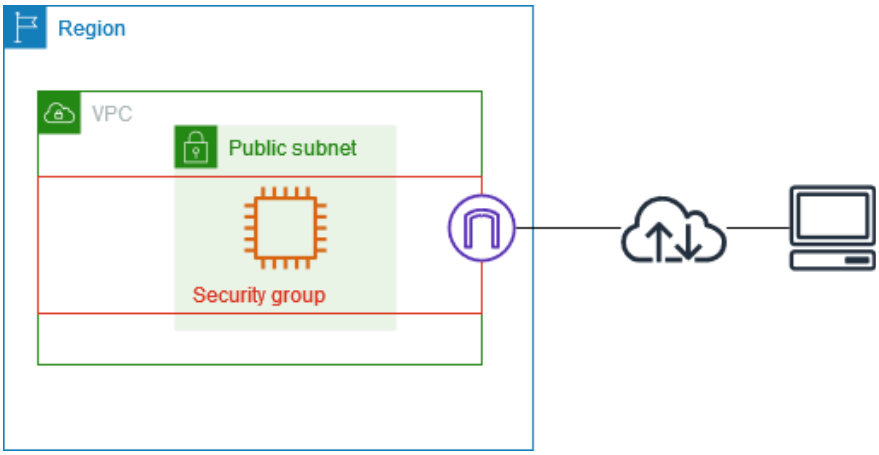


Figura 7: Disseny de la infraestructura de xarxa que utilitzarem, amb múltiples subnets

4.1. VPC i subnets

Per garantir una alta disponibilitat, el clúster residirà dins d'un VPC específicament creat per aquest propòsit a AWS. El disseny realitzat inclou múltiples subnets repartides en tres zones de disponibilitat (AZs) per a prevenir interrupcions de servei en cas que es produeixi una fallada en alguna zona.

- **VPC**: Utilitzant un rang CIDR de `172.16.0.0/20`, el qual ofereix fins a 4096 adreces IP, acomodant així les necessitats futures d'escalabilitat del clúster.
- **Subnets**:
    - **Subnets per nodes mestres**: Petites subnets dedicades que allotjaran els nodes mestres de EKS. Els nodes mestres d'EKS disposarien de 42 IPs (EKS recomana com a mínim 16 IPs [34]).
    - **Subnets per nodes treballadors**: Majoritàriament més gran, aquestes subnets albergaran els nodes treballadors que contindran els pods. Els nodes treballadors disposarien de 3066 IPs en total.

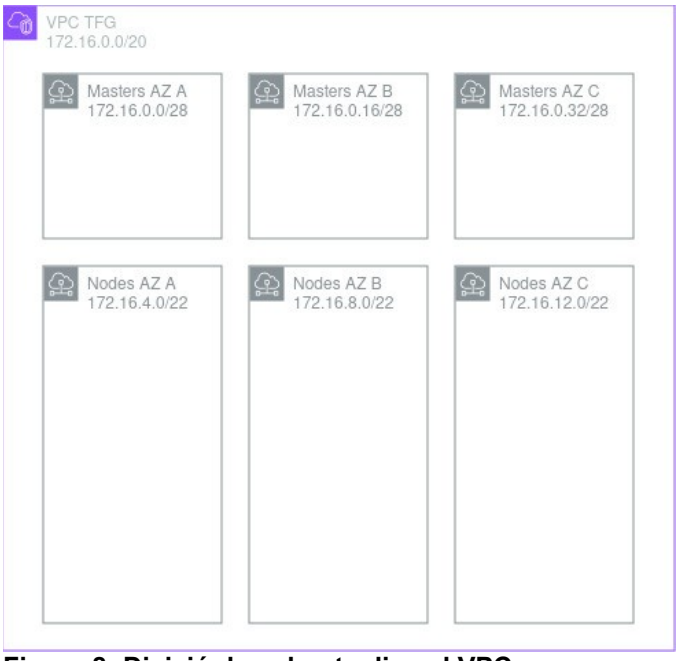


Figura 8: Divisió de subnets dins el VPC

En l'Annex 3 podrem veure el codi utilitzat i el procés de creació del VPC i les subnets especificades.

4.2. Seguretat i compliance

- **Security groups**: Creació de security groups específics que permetran el tràfic necessari per al funcionament del clúster mentre bloquegen l'accés no autoritzat.
- **KMS per a l'encriptació**: Utilització d'AWS Key Management Service (KMS) per a encriptar els secrets del clúster, assegurant la confidencialitat de les dades sensibles. [35]

En l'Annex 4 podrem veure el codi utilitzat i el procés de creació de la clau KMS.

4.3. Resolució de problemàtiques complexes

Durant la fase de disseny, es van identificar i resoldre diverses problemàtiques complexes, incloent la gestió eficient d'IPs pels nodes i pods i la configuració de xarxes dins de múltiples AZs per màxim rendiment i disponibilitat.

- **Gestió d'IPs**: El desplegament de Cilium com a CNI proporciona una gestió eficient de les adreces IP dins del clúster, alliberant així més IPs per als nodes.
- **Configuració Multi-AZ**: La planificació detallada de la ubicació de subnets i la seva assignació a zones específiques ajuden a minimitzar el risc de caiguda i optimitzar la latència de la xarxa.

Una vegada creats aquests recursos, estarem llestos per aixecar el nou clúster de Kubernetes en AWS.

5. Creació del clúster de Kubernetes

5.1. L'IaC a Kubernetes

Objectius i requisits tecnològics:

L'objectiu tecnològic és implementar l'ús de la infraestructura com a codi a Kubernetes per automatitzar el desplegament i la gestió dels recursos de la infraestructura de manera eficient i consistent mitjançant l'ús d'eines com eksctl o Terraform. L'objectiu és millorar la gestió dels recursos, reduir els errors humans i garantir la coherència entre els diferents entorns d'implementació de Kubernetes.

Terraform per a Kubernetes

Terraform és una eina que suporta la gestió de múltiples serveis i plataformes, incloent Kubernetes. A través del seu llenguatge declaratiu, els usuaris poden definir la infraestructura necessària en fitxers de configuració, que Terraform utilitza per crear, modificar i gestionar els recursos de manera eficient, molt semblant a com s'utilitza per la infraestructura d'AWS. Terraform és ideal per a administrar recursos a gran escala i suporta la integració amb diverses plataformes, incloent AWS, on pot gestionar EKS. [32]

Eksctl

Eksctl és l'eina oficial d'AWS per a la gestió de clústers EKS. Eksctl simplifica les operacions d'EKS, permetent als usuaris centrar-se més en l'ús de Kubernetes que en la seva gestió. Amb eksctl, els usuaris poden crear, actualitzar, i eliminar clústers EKS amb comandes senzilles, facilitant un entorn controlat i consistent. [36]

Les característiques clau d'eksctl inclouen:

- **Simplicitat**: Comandes simplificades per a la gestió eficient del clúster, com `eksctl create cluster`.
- **Integració profunda amb AWS**: Optimització per a EKS, aprofitant funcionalitats AWS natives com IAM per a seguretat. Per sota, utilitza CloudFormation.
- **Ràpida configuració i desplegament**: Capacitat per establir clústers en minuts, inclòs el desplegament automàtic de nodes i configuracions.

En aquest projecte específic, s'ha decidit optar per eksctl com a la nostra eina de IaC per a la gestió del clúster de Kubernetes per diverses raons:

1. **Eficiència i eficàcia**: Eksctl redueix significativament el temps necessari per configurar clústers EKS comparat amb altres eines, facilitant una integració ràpida i fiable.
2. **Especialització per EKS**: Al ser una eina dissenyada específicament per EKS, eksctl proporciona una compatibilitat i eficàcia millorades en aquest entorn.
3. **Integració amb CloudFormation**: Com que per a la mateixa infraestructura d'AWS utilitzem CloudFormation, és lògic utilitzar eines que facin ús de la mateixa tecnologia.

5.2 Creació del clúster

Objectius i requisits tecnològics:

L'objectiu d'aquesta secció és crear un clúster de Kubernetes dins de l'entorn AWS que hem creat prèviament, utilitzant Amazon EKS, per assegurar un entorn de gestió escalable, segur i eficient per a aplicacions contenidoritzades.

Els requisits específics com ara l'escalabilitat, l'alta disponibilitat i la eficiència operativa i la seguretat els contemplarem en altres seccions, no en la creació del clúster, on no realitzarem cap tipus de configuració addicional.

Per a la creació del clúster hem utilitzat l'eina d'eksctl. Amb eksctl podem crear un fitxer YAML amb les dades que volem que tingui el nostre clúster. Aquest fitxer YAML segueix una plantilla específica que podem consultar al web oficial d'eksctl. [37]

Per a omplir aquest fitxer hem tingut les següents consideracions:

- Utilitzant el VPC que hem creat abans, hem especificat les 3 diferents subnets en diferents AZs per garantir l'alta disponibilitat.
- Li hem especificat la clau KMS que utilitzarà per a encriptar els secrets i les dades confidencials.
- Utilitzarem la versió 1.29 de Kubernetes, la més recent a dia de la realització del treball.
- Farem ús del sistema operatiu Amazon Linux 2, el recomanat per AWS.

- Hem seleccionat diferents famílies d'instàncies EC2 amb 16GB de memòria RAM per a que AWS tingui diferents opcions d'instàncies d'omplir la demanda.
- S'ha optat per crear el clúster a la regió d'«us-east-1» (Virgínia del nord) degut a que la infraestructura de l'empresa és a aquesta regió.

En l'Annex 5 podrem veure el procés complet pas a pas i el codi utilitzat per a crear i accedir al clúster d'AWS.

Una vegada creat el clúster hem pogut comprovar que podem comunicar-nos correctament amb l'API de Kubernetes mitjançant kubectl.

```
❯ export KUBECONFIG="/home/[redacted]/.kubeuoc/config"
❯ kubectl get node
NAME                           STATUS   ROLES    AGE   VERSION
ip-172-16-13-249.ec2.internal   Ready    <none>   22m   v1.29.0-eks-5e0fdde
ip-172-16-5-154.ec2.internal    Ready    <none>   22m   v1.29.0-eks-5e0fdde
ip-172-16-9-240.ec2.internal    Ready    <none>   22m   v1.29.0-eks-5e0fdde
```

Figura 9: Obtenció dels nodes mitjançant kubectl

Dins de la consola d'AWS podem veure també que el clúster s'ha creat correctament, amb cada un dels nodes dins un *nodegroup* diferent, on cada un és a una AZ diferent.

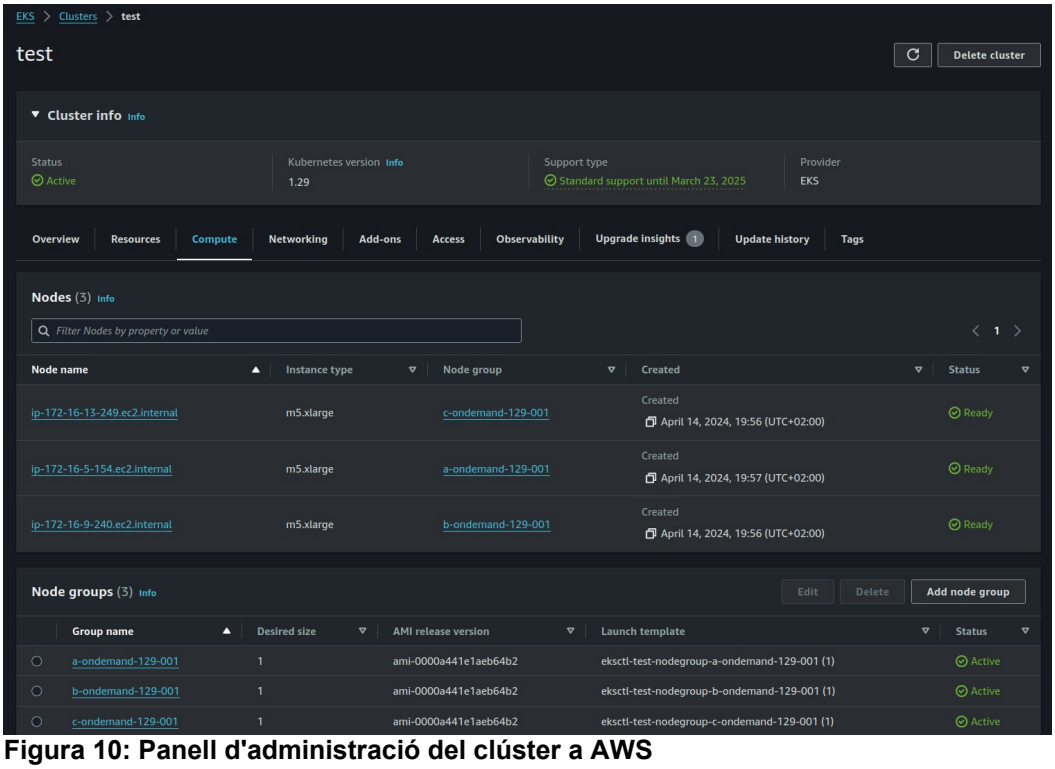


Figura 10: Panell d'administració del clúster a AWS

Amb el clúster ja creat correctament, entrarem a la fase de configuració i personalització del mateix.

6. Configuració i personalització del clúster

6.1. L'IaC de les aplicacions de Kubernetes

Objectius i requisits tecnològics

L'objectiu tecnològic és implementar l'ús de la infraestructura com a codi per a les aplicacions de Kubernetes. Això implica definir i gestionar la configuració de les aplicacions desplegades a Kubernetes de manera codificada, permetent la replicació, el desplegament i l'actualització automatitzats d'aplicacions de manera controlada i repetible.

6.1.1. Helm

Helm és conegut com el "gestor de paquets de Kubernetes". Això significa que Helm permet als usuaris empaquetar, configurar i desplegar aplicacions a Kubernetes de manera fàcil i ràpida. Aquesta empaquetació s'anomena "chart", i conté totes les dades necessàries per a desplegar una aplicació en un clúster de Kubernetes. [38]

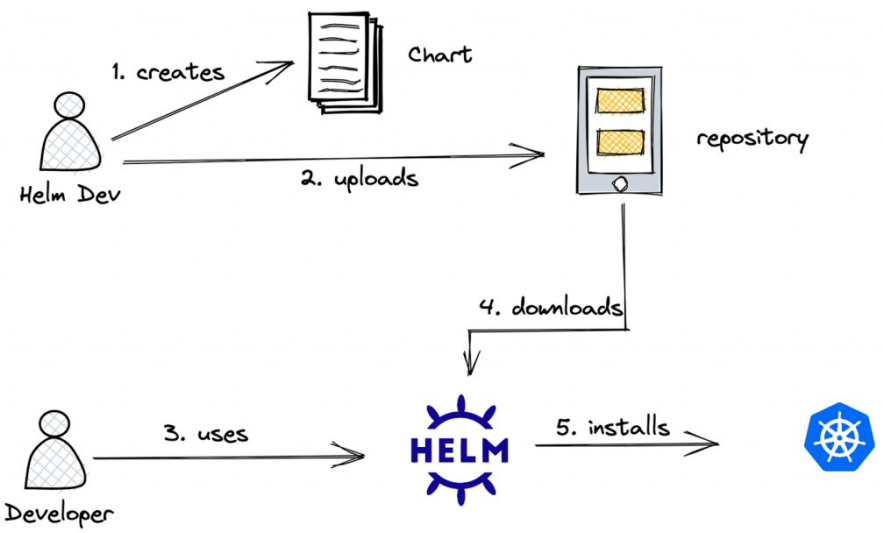


Figura 11: Diagrama de flux de l'ús de Helm

Helm fa possible instal·lar i actualitzar el software desplegat amb una simple comanda, així com també revertir a versions anteriors si es detecta algun problema amb la nova versió. Aquesta capacitat de gestionar versions facilita enormement el manteniment i la gestió de riscos.

6.2. El gestor de xarxa escollit: Cilium

Ara mateix tenim un clúster de Kubernetes operatiu, sense cap tipus de configuració extra. Si, per exemple, aixequem un nou pod amb una imatge de nginx, podrem veure que s'assigna correctament a un node i aixeca correctament.

```
❯ k get pod
No resources found in default namespace.
❯ k run nginx --image=nginx --restart=Never
pod/nginx created
❯ k get pod -owide
NAME    READY   STATUS              RESTARTS   AGE   IP       NODE                            NOMINATED NODE   READINESS GATES
nginx   0/1     ContainerCreating   0          3s    <none>   ip-172-16-9-240.ec2.internal    <none>           <none>
❯ k get pod -owide
NAME    READY   STATUS    RESTARTS   AGE   IP           NODE                            NOMINATED NODE   READINESS GATES
nginx   1/1     Running   0          10s   172.16.8.67  ip-172-16-9-240.ec2.internal    <none>           <none>
```

Figura 12: Creació d'un pod de nginx sense Cilium en ús

El que podem observar també és que està utilitzant IPs de la mateixa subnet que hem creat prèviament a AWS, fet que no ens interessa. El que farem serà instal·lar i configurar Cilium per a que ell mateix assigni les IPs dels pods, i s'encarregui de la comunicació dels pods mitjançant eBPF.

L'assignació d'IPs actual i la comunicació entre pods funciona ja que per defecte, els clústers d'EKS venen amb AWS VPC CNI, kube-proxy i coredns instal·lats per defecte. Fent una ullada als pods que hi ha ara mateix al namespace de kube-system podem veure-ho.

```
❯ k get pod -n kube-system
NAME                       READY   STATUS    RESTARTS   AGE
aws-node-bv84c             2/2     Running   0          40h
aws-node-cmvtb             2/2     Running   0          40h
aws-node-k54mg             2/2     Running   0          40h
coredns-54d6f577c6-rctdg   1/1     Running   0          40h
coredns-54d6f577c6-ttbs6   1/1     Running   0          40h
kube-proxy-dfqb6           1/1     Running   0          40h
kube-proxy-nkpmf           1/1     Running   0          40h
kube-proxy-xvj4g           1/1     Running   0          40h
```

Figura 13: Pods executant-se al namespace de Kube-system

On aws-node és el AWS VPC CNI, i coredns i kube-proxy els mateixos respectivament. El que volem és que Cilium assumeixi les responsabilitats del VPC CNI, i del kube-proxy. Els eliminarem per tant. Ens quedarà un disseny semblant al següent:

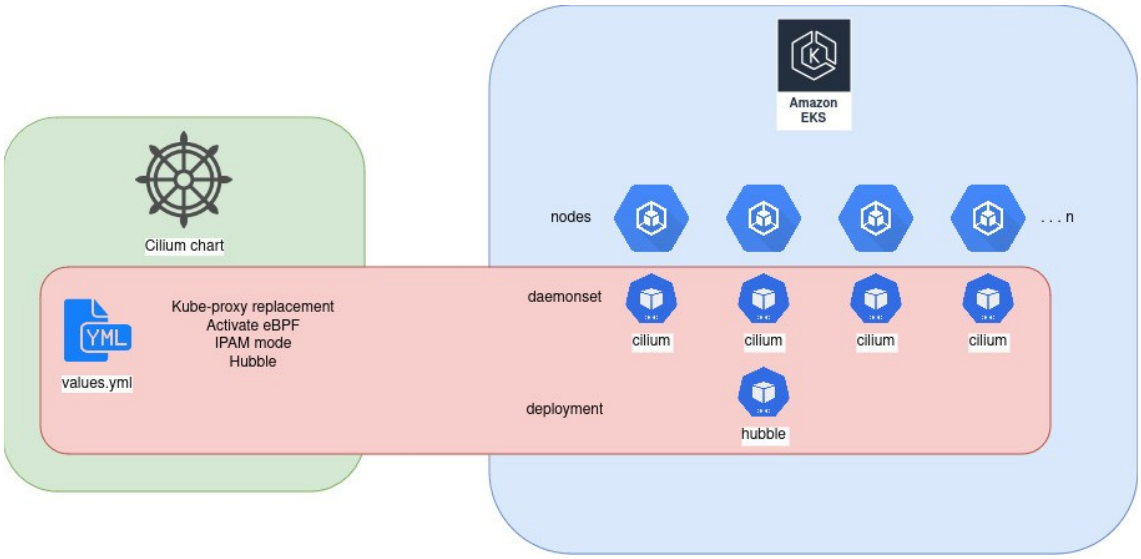


Figura 14: Disseny de la implementació de Cilium que utilitzarem

Al instal·lar Cilium hem tingut algunes consideracions: [39]

- Activarem el mode kubeProxyReplacement i localRedirectPolicy per poder reemplaçar el kube-proxy.
- Especificarem l'endpoint del clúster a la variable «k8sServiceHost»
- Deshabilitarem «hostLegacyRouting» per utilitzar la tecnologia eBPF, la recomanada de Cilium.
- Activarem el mode IPAM i especificarem les IPs internes amb les que treballarà Cilium.
- Activarem el Hubble per tindre visibilitat dels paquets que Cilium reenviarà.

Respecte les IPs internes, utilitzarem el CIDR de 100.64.0.0/10 per a les IPs dels pods, on cada node disposarà d'un rang /24 per als pods que contindrà (254 IPs per cada node).

En l'Annex 6 podrem veure el procés complet d'instal·lació i configuració de Cilium, juntament amb el codi utilitzat.

Si aixequem un pod de nginx, podem veure que cilium li ha assignat una IP interna controlada per ell mateix.

```
❯ k run nginx --image=nginx --restart=Never
pod/nginx created
❯ k get pod
NAME    READY   STATUS              RESTARTS   AGE
nginx   0/1     ContainerCreating   0          2s
❯ k get pod -owide
NAME    READY   STATUS    RESTARTS   AGE   IP             NODE                          NOMINATED NODE   READINESS GATES
nginx   1/1     Running   0          8s    100.64.1.254   ip-172-16-5-154.ec2.internal   <none>           <none>
```

Figura 15: Creació d'un pod de nginx amb Cilium ja configurat

Si mirem el recurs de «CiliumNode» podem veure la IP interna del cada node, I la seva IP interna de Cilium, en concordança als paràmetres que li hem indicat prèviament.

```
❯ k get ciliumnode
NAME                           CILIUMINTERNALIP   INTERNALIP      AGE
ip-172-16-13-249.ec2.internal  100.64.0.183       172.16.13.249   6m27s
ip-172-16-5-154.ec2.internal   100.64.1.121       172.16.5.154    6m26s
ip-172-16-9-240.ec2.internal   100.64.2.72        172.16.9.240    6m26s
```

Figura 16: Obtenció dels recursos de "CiliumNode"

Mitjançant un proxy invers podríem accedir al Hubble de Cilium directament. Ara per sortir del pas sense muntar-ne un, si volem accedir al hubble, podem fer un *port-forward* i accedir des del nostre navegador.

```
❯ kubectl port-forward -n kube-system deployment/hubble-ui 8081
Forwarding from 127.0.0.1:8081 -> 8081
Forwarding from [::1]:8081 -> 8081
```

Figura 17: Redireccionament de ports per a poder accedir a Hubble sense un proxy invers

Ara tenim molt poca cosa, però podem veure els paquets que s'envien els pods entre ells, o cap a fora, amb el permís de cilium.

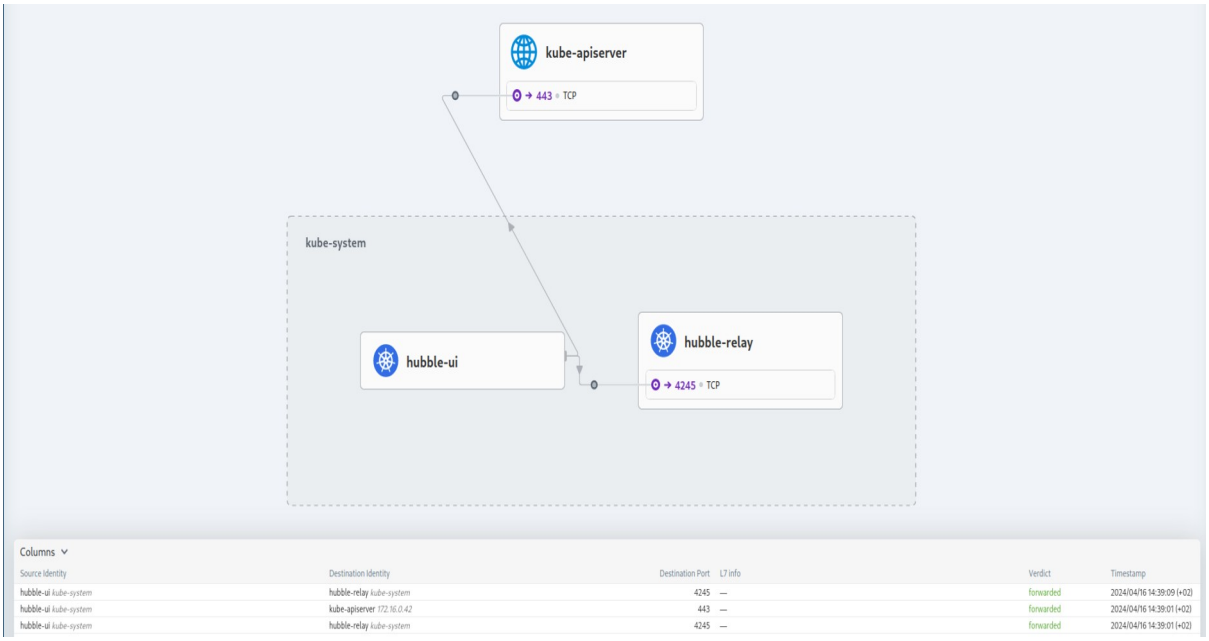


Figura 18: Panell de monitorització de Hubble

6.3. Implementació i proves amb CoreDNS

En versions prèvies de Kubernetes, Kube-DNS era l'eina que venia instal·lada per defecte al clúster. Si fem una ullada als pods que tenim a kube-system, podem veure que això ja no és així.

```
❯ k get pod -n kube-system | grep coredns
coredns-54d6f577c6-gtjdq           1/1     Running   0          27h
coredns-54d6f577c6-vkl2c           1/1     Running   0          27h
```

Figura 19: Obtenció dels pods de CoreDNS

Podem veure que CoreDNS ja és l'eina instal·lada per defecte, comptant amb 2 pods per a garantir una alta disponibilitat i podent escalar horitzontalment mitjançant un deployment.

```
❯ k get deployment coredns -n kube-system
NAME      READY   UP-TO-DATE   AVAILABLE
coredns   2/2     2            2
```

Figura 20: Obtenció del deployment de CoreDNS

També podem modificar fàcilment el seu comportament mitjançant un ConfigMap (temps de catxé, *time-to-lives*, servidors DNS a utilitzar...) però que per l'objectiu del nostre projecte la configuració per defecte ens és ja suficient. No realitzarem cap modificació per aquesta raó. [25]

Des d'un pod, podem realitzar una prova per validar que CoreDNS està funcionant correctament, mitjançant una resolució de noms a un component del mateix clúster. En aquest cas provarem el servei de «Kubernetes» que resideix en el namespace default.

```
❯ k get svc kubernetes
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
kubernetes   ClusterIP   172.31.0.1   <none>        443/TCP
❯ k exec -ti nginx -- bash
root@nginx:/# nslookup kubernetes.default
Server:         172.31.0.10
Address:        172.31.0.10#53

Name:   kubernetes.default.svc.cluster.local
Address: 172.31.0.1

root@nginx:/# _
```

Figura 21: Prova de resolució de noms a Kubernetes

Veiem que ens retorna la mateixa IP del servei de Kubernetes correctament.

Amb la comanda dig realitzarem una prova per veure el temps de resposta del servidor CoreDns.

```
root@nginx:/# dig kubernetes.default

; <<>> DiG 9.18.24-1-Debian <<>> kubernetes.default
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 47044
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: e04a81cf3880d0f1 (echoed)
;; QUESTION SECTION:
;kubernetes.default.            IN      A

;; AUTHORITY SECTION:
.                       30      IN      SOA     a.root-servers.net. nstld.verisign-grs.com. 2024051001 1800 900 604800 86400

;; Query time: 0 msec
;; SERVER: 172.31.0.10#53(172.31.0.10) (UDP)
;; WHEN: Fri May 10 19:41:48 UTC 2024
;; MSG SIZE  rcvd: 134
```

Figura 22: Prova per mesurar el temps de resposta d'una petició

Podem veure que el temps de resposta és menor a 1ms, és a dir, un temps de resposta molt acceptable.

Hem pogut per tant, validar els objectius que hem definit en el punt 3.4 (alta disponibilitat, escalabilitat, baixa latència i fàcil gestió).

6.4. L'escalat de pods basant-se en events: Keda

Si bé en l'apartat de l'anàlisi de l'escalat de pods hem revisat el HPA i el VPA, aquests són components extensament utilitzats i completament assentats en l'ecosistema de Kubernetes. La tècnica d'escalar pods basant-se en events que realitza Keda és una eina molt potent que ens permetrà integrar l'escala i desescala dels nostres pods amb serveis d'AWS, com per exemple el del nostre cas d'ús: una cua SQS.

El nostre disseny per al nostre cas d'ús serà el següent:

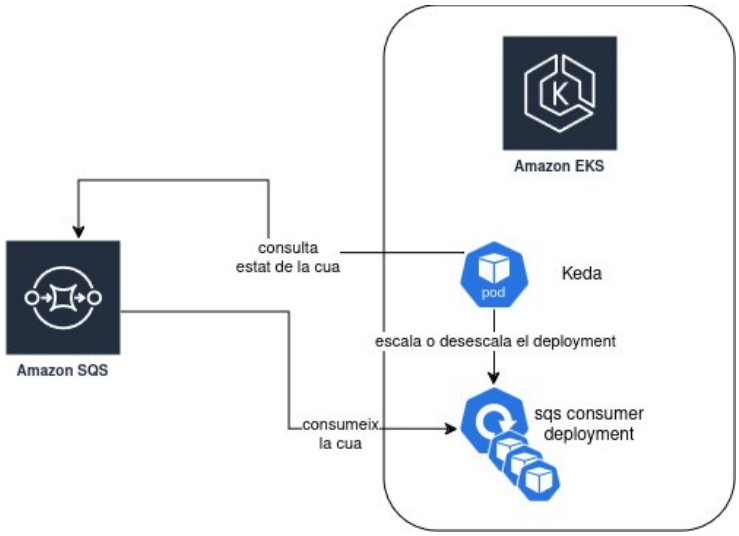


Figura 23: Funcionament de Keda en el nostre cas d'ús

Al instal·lar Keda hem tingut les següents consideracions:

- S'han configurat els paràmetres de dnsPolicy i useHostNetwork seguint la documentació per funcionar correctament amb Cilium. [40]
- Hem creat un rol d'AWS amb permisos de lectura d'atributs de la cua SQS per veure el número de missatges encuats.
- Al mateix rol, mitjançant OIDC, se li ha donat permís al service account de keda-operator, al namespace de kube-system per assumir-lo, seguint les indicacions de la documentació oficial. [41]

En l'Annex 7 podrem veure el procés complet d'instal·lació i configuració de Keda, juntament amb el codi utilitzat.

Una vegada configurat Keda, podrem veure l'objecte i el consumidor de la següent manera:

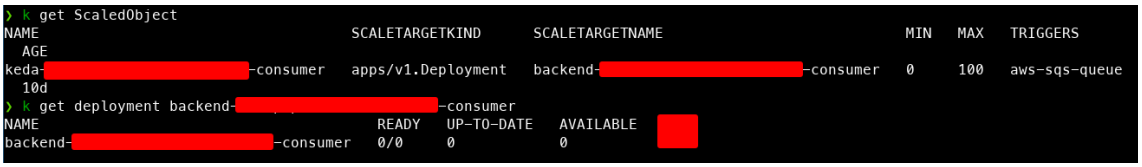


Figura 24: Obtenció de l'objecte a escalar i el deployment, amb 0 missatges a la cua

Podem veure que el Deployment, com que no hi ha res a la cua, és a 0 pods. Estarem llestos per a realitzar la prova.

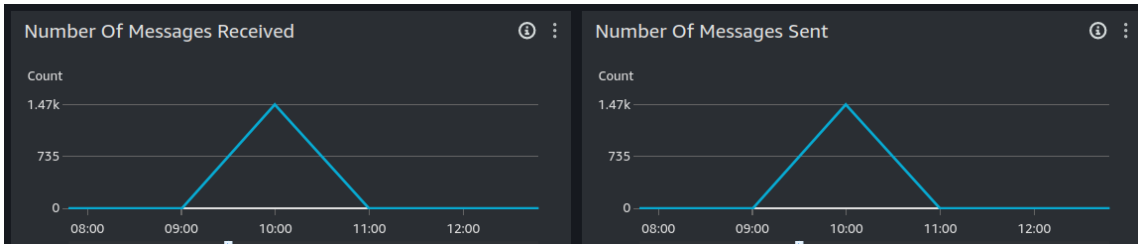


Figura 25: Monitorització del nombre de missatges a la cua

Hem omplert la cua de missatges (les mètriques les obtenim del mateix monitoratge de la cua a AWS). Degut a limitacions del monitoratge d'AWS, aquest s'actualitza en intervals d'una hora.

Si revisem els logs del pod de Keda podem veure com actualitza els consumidors quan detecta missatges a la cua.

2024-04-27T10:03:39Z INFO scaleexecutor Successfully updated ScaleTarget {"scaledobject.Name": "keda-XXXXX-consumer", "scaledObject.Namespace": "default", "scaleTarget.Name": "backend-XXXXX-consumer", "Original Replicas Count": 0, "New Replicas Count": 1}
2024-04-27T10:03:59Z INFO scaleexecutor Successfully updated ScaleTarget {"scaledobject.Name": "keda-XXXXX-consumer", "scaledObject.Namespace": "default", "scaleTarget.Name": "backend-XXXXX-consumer", "Original Replicas Count": 1, "New Replicas Count": 32}
2024-04-27T10:04:59Z INFO scaleexecutor Successfully set ScaleTarget replicas count to ScaledObject minReplicaCount {"scaledobject.Name": "keda-XXXXX-consumer", "scaledObject.Namespace": "default", "scaleTarget.Name": "backend-XXXXX-consumer", "Original Replicas Count": 32, "New Replicas Count": 0}

El deployment ha escalat correctament

```
❯ k get deployment backend-[redacted]-consumer
NAME                                  READY   UP-TO-DATE   AVAILABLE
backend-[redacted]-consumer   32/32   32           32
```

Figura 26: Obtenció del deployment quan hi ha missatges a la cua

I una vegada la cua ha quedat sense missatges, ha desescalat correctament.

```
❯ k get deployment backend-[redacted]-consumer
NAME                                  READY   UP-TO-DATE   AVAILABLE
backend-[redacted]-consumer   0/0     0            0
```

Figura 27: Obtenció del deployment una vegada s'han consumit tots els missatges

Si bé la prova inicialment ha estat un èxit, hem de tindre en compte algunes configuracions per poder aprofitar al màxim l'escalat basat en la mida d'una cua, sense desaprofitar recursos: [42]

1. **Llindars d'escala**: Definirem el valor de `queueLength`, el qual és el nombre de missatges que cada pod hauria de poder processar. Si per exemple tenim molts missatges que requereixen processament ràpid, ajustarem aquest valor cap a baix per a tindre un escalat més gran.
2. **Rèpliques mínimes i màximes**: Defineix els valors de `minReplicaCount` i `maxReplicaCount` per controlar el nombre mínim i màxim de rèpliques que KEDA pot escalar. Això ajuda a controlar els costos i a garantir que sempre hi hagi una capacitat mínima disponible per processar missatges.
3. **Període de refredament**: Després d'escalar cap avall, és útil definir un període de refredament (`cooldownPeriod`) durant el qual no es realitzaran més operacions d'escalat cap avall. Això pot evitar la fluctuació constant del nombre de pods.
4. **Interval de trucada**: El `pollingInterval` és el temps en segons que KEDA espera per consultar l'estat de la cua de SQS i ajustar l'escalat si és necessari. Un valor baix pot fer que l'escalat sigui més reactiu però també pot augmentar els costos d'API i la càrrega sobre el sistema de control de Kubernetes.

6.5. L'escalat de nodes oficial d'AWS: Karpenter

En l'apartat 3.6 hem pres la decisió d'utilitzar Karpenter per sobre de Cluster Autoscaler (CAS) degut a la seva integració amb AWS i optimització de costos. En aquest apartat instal·larem Karpenter, i realitzarem unes proves funcionals per a provar el seu funcionament.

El nostre disseny que implementarem per a realitzar les proves serà com aquest:

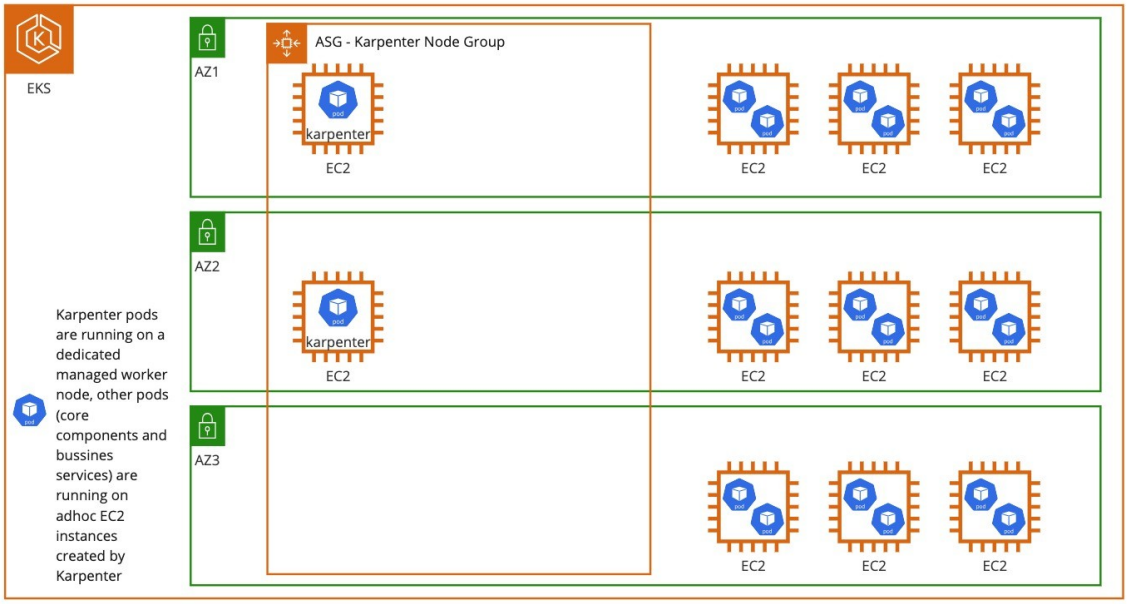


Figura 28: Funcionament de Karpenter en el nostre cas d'ús

Al instal·lar Karpenter hem tingut les següents consideracions:

- Hem seguit la guia d'instal·lació i configuració que parteix que tenim un CAS [45]. Això és degut a que l'altra guia d'instal·lació assumeix que ni tan sols tenim un clúster creat, i aixeca el clúster conjuntament amb Karpenter.
- Hem creat els rols amb els permisos necessaris que tindran els nodes de Keda, i el mateix controlador de Keda.
- Hem afegit tags de descobriment que Keda utilitzarà per a crear els nous nodes en les subnets correctes del clúster, i amb els grups de seguretat d'accés desitjats.
- Li hem proporcionat permisos a nivell de Kubernetes per a que els nous nodes puguin assumir el rol creat prèviament.

- Hem creat una afinitat de pods per al controlador de Karpenter, per a que s'executi en nodes no gestionats per ell mateix. Això és per a evitar que Karpenter s'executi en un mateix node que ell gestiona, i evitar una dependència cíclica.

En l'Annex 8 podem veure amb més detall el procés d'instal·lació i configuració que hem dut a terme.

Crearem un NodePool i un EC2NodeClass per a definir el comportament de Karpenter.

```yaml
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  template:
    spec:
      requirements:
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
        - key: kubernetes.io/os
          operator: In
          values: ["linux"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["m"]
        - key: karpenter.k8s.aws/instance-family
          operator: In
          values: ["m5","m5a","m5ad","m5d","m5dn","m5n","m6i"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["4"]
        - key: "karpenter.k8s.aws/instance-cpu"
          operator: In
          values: ["4"]
      nodeClassRef:
        apiVersion: karpenter.k8s.aws/v1beta1
        kind: EC2NodeClass
        name: default
  limits:
    cpu: 1000
  disruption:
    consolidationPolicy: WhenUnderutilized
    #expireAfter: 720h # 30 * 24h = 720h
---
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
  role: "k8s-test-karpenter-nodegroup"
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "test"
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "test"
  amiSelectorTerms:
    - id: "ami-057f49c54d950e56c"
```

Figura 29: Codi del NodePool amb els filtres i el NodeClass amb l'AMI a utilitzar

En el NodeClass indicarem la AMI, el rol dels nodes, i amb els tags que hem creat abans, descobrirà els subnets i els SG on afegirà els nous nodes. En l'apartat de «limits», podrem indicar el nombre màxim de CPU o memòria que el total de nodes tindran. És a dir, si els nodes que utilitzem tenen 4 vCPU i posem un límit de 40, podrem tindre com a màxim 10 nodes.

En el NodePool indicarem quin tipus de node voldrem. En el nostre cas arquitectura amd64, Linux, de la família m, generació mínima 4... Aqui podrem restringir els tipus de nodes que volem utilitzar. En el meu cas he utilitzat el mateix que he utilitzat prèviament. També podrem configurar dades com un temps màxim per a cada node per a que es vagin rotant periòdicament.

Per a realitzar una prova, crearem un deployment de pods de nginx, on a cada pod li posarem una request de memòria alta, per a que no tinguem capacitat suficient amb els nodes actuals i haguem d'escalar.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        resources:
          requests:
            memory: "12Gi"
          limits:
            memory: "12Gi"
        ports:
        - containerPort: 80
```

Figura 30: Codi del deployment que utilitzarem per fer proves d'escalats de nodes

Els nostres nodes tenen 16Gb de memòria. Per aquesta raó no podrà haver 2 pods de nginx al mateix node.

```
❯ k get pod
NAME                                          READY   STATUS    RESTARTS   AGE
external-secrets-55f5784c48-6hrt4             1/1     Running   0          6d20h
external-secrets-cert-controller-7747659548-w6rcj   1/1     Running   0          6d20h
external-secrets-webhook-868697858d-9z6pk     1/1     Running   0          6d20h
nginx-deployment-8547b666c7-5tq47             1/1     Running   0          2m38s
nginx-deployment-8547b666c7-dd6km             0/1     Pending   0          2s
nginx-deployment-8547b666c7-q4knq             0/1     Pending   0          2s
nginx-deployment-8547b666c7-ts7h7             1/1     Running   0          71s
nginx-deployment-8547b666c7-wxsvt             1/1     Running   0          2m40s
```

Figura 31: Obtenció dels pods, on no hi ha recursos suficients per aixecar-los tots

Només 3 dels pods han pogut arrencar, el clúster ha d'escalar per a poder executar la resta de pods.

```
❯ k get pod -owide
NAME                                          READY   STATUS    RESTARTS   AGE     IP
external-secrets-55f5784c48-6hrt4             1/1     Running   0          6d20h   100.64.2.14
external-secrets-cert-controller-7747659548-w6rcj   1/1     Running   0          6d20h   100.64.1.18
external-secrets-webhook-868697858d-9z6pk     1/1     Running   0          6d20h   172.16.9.240
nginx-deployment-8547b666c7-5tq47             1/1     Running   0          4m38s   100.64.2.135
nginx-deployment-8547b666c7-dd6km             1/1     Running   0          2m2s    100.64.8.15
nginx-deployment-8547b666c7-q4knq             1/1     Running   0          2m2s    100.64.9.191
nginx-deployment-8547b666c7-ts7h7             1/1     Running   0          3m11s   100.64.0.177
nginx-deployment-8547b666c7-wxsvt             1/1     Running   0          4m40s   100.64.1.103
❯ k get node
NAME                             STATUS   ROLES    AGE   VERSION
ip-172-16-13-249.ec2.internal    Ready    <none>   20d   v1.29.0-eks-5e0fdde
ip-172-16-5-154.ec2.internal     Ready    <none>   20d   v1.29.0-eks-5e0fdde
ip-172-16-5-41.ec2.internal      Ready    <none>   91s   v1.29.3-eks-ae9a62a
ip-172-16-6-245.ec2.internal     Ready    <none>   94s   v1.29.3-eks-ae9a62a
ip-172-16-8-81.ec2.internal      Ready    <none>   54s   v1.29.3-eks-ae9a62a
ip-172-16-9-240.ec2.internal     Ready    <none>   20d   v1.29.0-eks-5e0fdde
```

Figura 32: Obtenció dels pods i nodes, una vegada Karpenter ha escalat

Podem veure que Karpenter ha afegit més nodes per a que els nous pods de nginx puguin executar-se. Podem veure quins són els nodes de Karpenter ja que la versió de Kubernetes difereix, al utilitzar una AMI més recent a la que hem utilitzat quan hem creat el clúster.

En els logs podem veure com Karpenter ha aixecat correctament el node

{"level":"INFO","time":"2024-05-05T16:32:48.741Z","logger":"controller.nodeclaim.lifecycle","message":"launched nodeclaim","commit":"fb4d75f","nodeclaim":"default-vxvnv","provider-id":"aws:///us-east-1b/i-005e1edea594c5f60","instance-type":"m5a.xlarge","zone":"us-east-1b","capacity-type":"on-demand","allocatable":{"cpu":"3920m","ephemeral-storage":"17Gi","memory":"14162Mi","pods":"58","vpc.amazonaws.com/pod-eni":"18"}}

Podem veure que si eliminem el deployment, i el clúster deixa de necessitar els nodes, aquest desescalarà automàticament.

```
deployment.apps "nginx-deployment" deleted
❯ k get node
NAME                             STATUS   ROLES    AGE   VERSION
ip-172-16-13-249.ec2.internal    Ready    <none>   20d   v1.29.0-eks-5e0fdde
ip-172-16-5-154.ec2.internal     Ready    <none>   20d   v1.29.0-eks-5e0fdde
ip-172-16-9-240.ec2.internal     Ready    <none>   20d   v1.29.0-eks-5e0fdde
```

Figura 33: Obtenció dels nodes, una vegada ha desescalat

Revisant els logs podem veure com els ha eliminat correctament també

{"level":"INFO","time":"2024-05-05T16:36:13.397Z","logger":"controller.disruption","message":"disrupting via consolidation delete, terminating 1 nodes (0 pods) ip-172-16-6-245.ec2.internal/m5a.xlarge/on-demand","commit":"fb4d75f","command-id":"0767aca6-175b-471b-bca3-358e866e09d2"}
{"level":"INFO","time":"2024-05-05T16:36:14.096Z","logger":"controller.disruption.queue","message":"command succeeded","commit":"fb4d75f","command-id":"0767aca6-175b-471b-bca3-358e866e09d2"}
{"level":"INFO","time":"2024-05-05T16:36:14.144Z","logger":"controller.node.termination","message":"tainted node","commit":"fb4d75f","node":"ip-172-16-6-245.ec2.internal"}
{"level":"INFO","time":"2024-05-05T16:36:14.176Z","logger":"controller.node.termination","message":"deleted node","commit":"fb4d75f","node":"ip-172-16-6-245.ec2.internal"}

{"level":"INFO","time":"2024-05-05T16:36:14.602Z","logger":"controller.nodeclaim.termination","message":"deleted nodeclaim","commit":"fb4d75f","nodeclaim":"default-f8d8v","node":"ip-172-16-6-245.ec2.internal","provider-id":"aws:///us-east-1a/i-09f655f1060dc1c35"}

Per a evitar problemes de seguretat, Karpenter pot actualitzar automàticament l'AMI dels nodes. Aquesta tècnica en conjunt amb una bona configuració del període de disrupció, podrem fer aquest procés amb la màxima disponibilitat possible. [46]

En el nostre cas, degut a que encara no disposem d'aplicacions per a provar aquest funcionament, es surt de l'abast d'aquest projecte.

6.6. Integració d'ASM amb Kubernetes: external-secrets

La necessitat d'integrar AWS Secrets Manager amb un cluster de Kubernetes gestionat mitjançant Amazon EKS es deriva de l'objectiu de centralitzar la gestió de secrets i garantir la confidencialitat de les dades en el núvol. AWS Secrets Manager ofereix una solució robusta per al maneig segur de credencials i secrets, permetent el seu accés segur i auditat. La utilització de External-Secrets a Kubernetes facilita la sincronització automàtica dels secrets des d'AWS cap al cluster, de manera que els secrets estan disponibles per als components del cluster sense necessitat de gestionar-los manualment.

El nostre disseny amb el que realitzarem les proves serà semblant al següent, orientat a AWS.

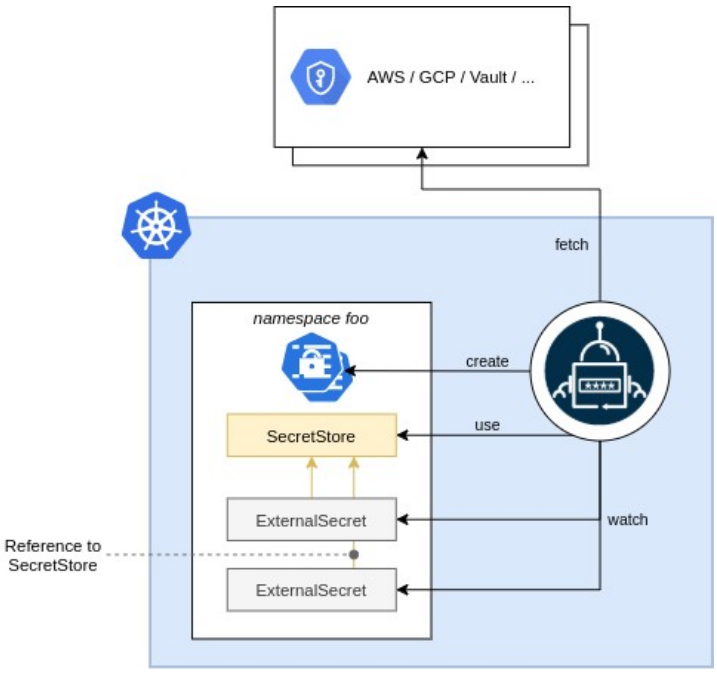


Figura 34: Funcionament d'external-secrets amb diferents eines de guardar secrets

Al instal·lar i configurar external-secrets amb Helm hem tingut les següents consideracions: [44]

- Degut a que utilitzem Cilium, hem habilitat el mode hostNetwork, de la mateixa forma que en paquets anteriors.
- Hem creat un rol amb permisos de lectura del secret que volem obtindre el valor, i que mitjançant OIDC per a garantir la confidencialitat, el Service Account que crearem posteriorment assumirà aquest rol.
- S'ha creat un SecretStore configurat per a utilitzar ASM per a gestionar els secrets, i garantir que els canvis en aquests són propagats de manera segura al clúster,
- Finalment, al recurs ExternalSecret hem definit els detalls del secret a sincronitzar, permetent una gestió fina de quins secrets són necessaris i on han de ser disponibles dins del cluster.

En l'Annex 9 podrem veure el procés complet d'instal·lació i configuració d'external-secrets, juntament amb el codi utilitzat.

Una vegada creat el secret, podem veure que s'està sincronitzant correctament.

```
❯ k get SecretStore
NAME          AGE   STATUS   CAPABILITIES   READY
test-secret   42s   Valid    ReadWrite      True
❯ k get ExternalSecret
NAME          STORE         REFRESH INTERVAL   STATUS         READY
test-secret   test-secret   24h                SecretSynced   True
```

Figura 35: Obtenció del SecretStore i el ExternalSecrets creats, correctament sincronitzats

Per a realitzar una prova, si creem un pod i li afegim aquest secret com a variable d'entorn, podem validar que s'està sincronitzant correctament.

```
❯ cat test.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    env:
      - name: SECRET_ENV_VAR
        valueFrom:
          secretKeyRef:
            name: test-secret
            key: dummysecret
  restartPolicy: Never

❯ k create -f test.yaml
pod/nginx created
❯ k exec -ti nginx -- bash
root@nginx:/# echo $SECRET_ENV_VAR
iamadummysecret
```

Figura 36: Prova funcional on un pod obté el secret creat prèviament

Aquí l'afegeixo com a variable d'entorn i a un sol pod, però podríem afegir-ho a un deployment i com a, per exemple, un fitxer amb dades confidencials.

Per defecte el secret s'actualitza en intervals de 24 hores. Aquest valor pot ser modificat segons els requeriments de cada servei. En cas que volguéssim forçar una sincronització, ho podem fer de forma manual també mitjançant una anotació, amb la data en Epoch. [47] Podem comprovar que els secret s'ha actualitzat correctament de la següent forma.

```
❯ k get es
kNAME          STORE         REFRESH INTERVAL   STATUS         READY
test-secret   test-secret   24h               SecretSynced   True
❯ kubectl annotate es test-secret force-sync=$(date +%s) --overwrite
externalsecret.external-secrets.io/test-secret annotated
❯ k get secret test-secret -oyaml
apiVersion: v1
data:
  dummysecret: aWFtYWR1bW15c2VjcmV0Mg==
immutable: false
kind: Secret
metadata:
  annotations:
    force-sync: "1714928985"
    reconcile.external-secrets.io/data-hash: 506d96f98a5b78ccc8a69f9a3ea34071
  creationTimestamp: "2024-04-28T20:21:43Z"
  labels:
    reconcile.external-secrets.io/created-by: b7cd1ff8ebb944021154c194f5043a44
  name: test-secret
  namespace: default
  ownerReferences:
  - apiVersion: external-secrets.io/v1beta1
    blockOwnerDeletion: true
    controller: true
    kind: ExternalSecret
    name: test-secret
    uid: c3e5f67c-a5cd-45b7-ba27-ff841dad1c00
  resourceVersion: "5606095"
  uid: 408c0344-0bda-4052-9a7b-a823d8d6bc7a
type: Opaque
❯ echo "aWFtYWR1bW15c2VjcmV0Mg==" | base64 -d
iamadummysecret2%
```

Figura 37: Prova funcional on actualitzem el secret

7. Conclusions i treballs futurs

Conclusions del treball

Un cop s'han obtingut els resultats, les principals conclusions extretes són les següents:

- **Investigació de components:** Hem investigat i implementat els principals components de Kubernetes disponibles al mercat, com Cilium per la gestió de la xarxa, CoreDNS per la resolució de noms, Keda per l'escalabilitat de pods, i Karpenter per l'escalabilitat de nodes. Tot i així, hi ha molts altres components que podrien haver estat investigats, però per qüestions de temps no ha estat possible explorar-los tots.
- **Experiència amb AWS:** La meva experiència prèvia amb AWS ha estat molt valuosa, especialment en la preparació de la infraestructura per al clúster de Kubernetes. La familiaritat amb els serveis d'AWS ha facilitat la configuració inicial de la infraestructura.
- **Facilitat de instal·lació del clúster:** Seguint la documentació d'Amazon EKS, el muntatge inicial del clúster de Kubernetes ha estat relativament senzill. Amazon proporciona una documentació detallada i passos clars que han simplificat aquest procés.
- **Desafiaments en la personalització:** Els veritables desafiaments han sorgit en la fase de personalització del clúster. Adaptar i configurar el clúster per satisfer les necessitats específiques de l'empresa ha requerit una comprensió profunda dels components i una gran atenció als detalls per a que els components funcionin correctament els uns amb els altres.

Reflexió crítica sobre l'assoliment dels objectius plantejats inicialment:

- **Assoliment d'objectius:** Hem assolit la majoria dels objectius plantejats inicialment, com el coneixement d'AWS EKS i Kubernetes, el desenvolupament d'habilitats en IaC, l'anàlisi i selecció de components de Kubernetes, i l'avaluació del rendiment i seguretat del clúster. No obstant, la limitació de temps ha impedit explorar més components i eines que podrien haver enriquit encara més el projecte.

Anàlisi crítica del seguiment de la planificació i metodologia:

- **Seguiment de la planificació:** Hem seguit la planificació establerta amb petites desviacions que no han afectat significativament els objectius del projecte.
- **Adequació de la metodologia:** La metodologia agile ha estat adequada per a aquest projecte, permetent una adaptació flexible als canvis.
- **Canvis introduïts:** Hem introduït ajustos menors per optimitzar el rendiment, especialment en la configuració de Keda i Karpenter.

Avaluació dels impactes previstos a 1.3:

- **Sostenibilitat:** S'ha aconseguit una major eficiència energètica gràcies a la millor gestió dels recursos.
- **Ètic-social:** Hem contribuït a la innovació i millora de la infraestructura tecnològica de l'empresa, creant oportunitats laborals qualificades.
- **Diversitat:** La utilització d'eines de codi obert facilita l'accés a infraestructures tecnològiques punteres a més organitzacions.

Impactes no previstos:

- No s'han identificat impactes no previstos significatius durant la implementació del projecte.

Línies de treball futur:

- **Millores en la monitorització i observabilitat:** Implementar eines com Prometheus i Grafana per obtenir millors *insights*.
- **Optimització contínua de l'escalabilitat:** Revisió contínua dels paràmetres d'escalabilitat.
- **Integració de noves tecnologies:** Mantenir-se actualitzat amb les últimes innovacions en tecnologies de contenidors i orquestració.

Aquestes conclusions i propostes de treballs futurs proporcionen una base sòlida per a la continuïtat del projecte, assegurant que l'empresa pugui seguir beneficiant-se de les millores implementades i continuar evolucionant la seva infraestructura tecnològica per satisfer les necessitats canviants del mercat.

8. Glossari

Agile	Metodologia de gestió de projectes que promou el desenvolupament iteratiu, la col·laboració constant amb el client i la capacitat de resposta al canvi.
Amazon Linux 2	Distribució de Linux basada en Red Hat creada per Amazon Web Services que proporciona un entorn segur, estable i d'alt rendiment per executar aplicacions en la infraestructura en núvol.
AMI	Acrònim d'Amazon Machine Image, una imatge de màquina preconfigurada utilitzada per crear una instància EC2.
Amd64	Arquitectura de processador x86 de 64 bits desenvolupada per AMD.
API	Acrònim d'Application Programming Interface, un conjunt de rutines que proveeix accés a funcions d'un determinat programari.
ASG	Acrònim d'Auto Scaling Group, un conjunt d'instàncies EC2 gestionades per l'auto escalat d'AWS per assegurar-se que es manté el nombre correcte d'instàncies per gestionar la càrrega.
AWS	Acrònim d'Amazon Web Services, una plataforma de serveis en núvol que ofereix potència de càlcul, emmagatzematge, bases de dades i altres funcionalitats a través d'Internet.
AZ	Acrònim d'Availability Zone, una ubicació geogràfica dins d'una regió d'AWS que té una infraestructura independent per assegurar una alta disponibilitat.
C	Llenguatge de programació de propòsit general desenvolupat als anys 70.
CI/CD	Acrònim de Continuous Integration/Continuous Deployment, una pràctica de desenvolupament de programari on es fan integracions freqüents i es despleguen automàticament.
CIDR	Acrònim de Classless Inter-Domain Routing, una metodologia per assignar adreces IP que millora l'eficiència en la utilització de l'espai d'adreces IP.
CloudFormation	Servei d'AWS que proporciona una manera fàcil de modelar i configurar recursos d'AWS i de tercers mitjançant l'ús de codi.
Clúster	Conjunt d'ordinadors o aplicacions connectades entre sí que donen un servei com si fossin un de sol.
CNI	Acrònim de Container Network Interface, un estàndard que defineix com configurar xarxes de contenidors.

Contenidor	Unitat de programari que empaqueta codi i totes les seves dependències perquè l'aplicació s'executi de manera ràpida i fiable en diferents entorns informàtics.
Daemonset (Kubernetes)	Ens permet assegurar-nos que una còpia d'un pod s'executa en tots els nodes del clúster de Kubernetes.
Deployment (Kubernetes)	Recurs de Kubernetes que proporciona actualitzacions declaratives per a pods.
DNS	Acrònim de Domain Name System, un sistema que tradueix noms de domini llegibles per humans a adreces IP numèriques.
Docker	Plataforma de codi obert per automatitzar el desplegament d'aplicacions dins de contenidors de programari.
eBPF	Acrònim d'extended Berkeley Packet Filter, una tecnologia que permet executar programes en l'espai del nucli de Linux per millorar la xarxa i la seguretat.
EC2	Acrònim d'Elastic Compute Cloud, un servei d'AWS que proporciona capacitat de càlcul escalable al núvol.
ECR	Acrònim d'Elastic Container Registry, un servei d'AWS per emmagatzemar, gestionar i desplegar imatges de contenidors Docker.
EKS	Acrònim d'Elastic Kubernetes Service, un servei d'AWS per executar Kubernetes al núvol.
ENI	Acrònim d'Elastic Network Interface, una interfície de xarxa virtual que es pot adjuntar a una instància EC2.
Epoch	Data de referència a partir de la qual es calculen els temps en molts sistemes informàtics, normalment el 1 de gener de 1970.
Etcd	Magatzem de claus-valor distribuït altament disponible que utilitza Kubernetes per emmagatzemar totes les dades del seu clúster.
Fargate	Tecnologia d'AWS que permet executar contenidors sense haver de gestionar els servidors o clústers subjacents.
Golang	Llenguatge de programació de codi obert desenvolupat per Google.
HTTP	Acrònim de Hypertext Transfer Protocol, el protocol utilitzat a la web per transferir dades i documents.
IaC	Acrònim d'Infrastructure as Code, la pràctica de gestionar i aprovisionar infraestructura informàtica mitjançant fitxers de configuració llegibles per l'usuari.
IAM	Acrònim d'Identity and Access Management, un servei d'AWS per controlar l'accés als recursos.
Imatge	Fitxer que conté un sistema de fitxers complet i tot el necessari per executar una aplicació, sovint utilitzat amb contenidors.
IPAM	Acrònim d'IP Address Management, eines i processos per planificar, seguir i gestionar les adreces IP en una xarxa.

Iptables	Eina de línia de comandes per configurar les regles del tallafocs al sistema operatiu Linux.
Ipvs	Acrònim d'IP Virtual Server, una tecnologia utilitzada per balancejar la càrrega de la xarxa al nivell IP.
IRSA	Acrònim d'IAM Roles for Service Accounts, una funcionalitat d'AWS EKS que permet associar rols d'IAM amb comptes de servei de Kubernetes mitjançant OIDC.
Kernel	Nucli del sistema operatiu que gestiona les operacions del maquinari i proporciona serveis bàsics als altres programes.
KMS	Acrònim de Key Management Service, un servei d'AWS que facilita la creació i control de claus criptogràfiques.
Kubectl o «k»	Eina de línia de comandes per interactuar amb clústers de Kubernetes.
Kubelet	Agent que s'executa en cada node del clúster de Kubernetes per garantir que els contenidors estan executant-se correctament.
Llibreria	Conjunt de recursos reutilitzables que poden ser utilitzats per desenvolupar programari.
Mono-procés	Aplicació que s'executa com un sol procés, sovint en contraposició a una arquitectura de microserveis.
Nginx	Servidor web d'alt rendiment que també pot ser utilitzat com a servidor intermediari invers (reverse proxy) i equilibrador de càrrega.
OIDC	Acrònim d'OpenID Connect, un protocol d'autenticació que permet la verificació de la identitat dels usuaris basada en l'autenticació a través d'un servidor d'autorització.
Outposts	Solució d'AWS que permet executar serveis d'AWS en infraestructures locals.
PaaS	Acrònim de Platform as a Service, una categoria de serveis de computació en núvol que proporciona una plataforma permetent als clients desenvolupar, executar i gestionar aplicacions sense la complexitat de construir i mantenir la infraestructura subjacent.
Pip	Eina de gestió de paquets per a Python que permet instal·lar i gestionar biblioteques i dependències de programari.
Plugin	Mòdul o extensió que afegeix funcionalitats addicionals a un programari existent.
Port forwarding	Tècnica que redirigeix una comunicació d'una adreça IP i número de port a una altra mentre travessa un tallafoc o router.
Programari "vanilla"	Programari en el seu estat original, sense cap modificació o personalització.
Prometheus	Sistema de monitorització i alerta de codi obert dissenyat per recopilar i agrupar dades de rendiment.

Proxy invers	Servidor que intercepta les peticions del client a un servidor de backend, proporcionant funcions com equilibrament de càrrega, autenticació i xifratge SSL.
REST	Acrònim de Representational State Transfer, un estil d'arquitectura per dissenyar serveis web amb operacions estàndard HTTP.
Runtime	Entorn en el qual un programa o aplicació s'executa, incloent el conjunt de biblioteques, serveis i altres components necessaris per a la seva execució.
SQS	Acrònim de Simple Queue Service, un servei d'AWS que ofereix una cua de missatges per desconnectar la comunicació entre components de l'aplicació.
TTL	Acrònim de Time to Live, un valor que indica el temps o salt màxim que un paquet de dades pot existir abans de ser descartat.
VPC	Acrònim de Virtual Private Cloud, un servei d'AWS que permet llançar recursos d'AWS en una xarxa virtual aïllada de manera lògica.
YAML	Acrònim de YAML Ain't Markup Language, un format de serialització de dades llegible per humans utilitzat sovint per fitxers de configuració.

9. Bibliografia

[1] Guia transversal sobre la CCEG para estudiantado de TFx-EIMT [Documentació en línia]
[Dia de la consulta: 9 de març de 2024]
< https://aula.uoc.edu/courses/31435/files/2553557?wrap=1 >

[2] How to Choose the Right Tool for Kubernetes Deployment [Documentació en línia]
[Dia de la consulta: 10 de març de 2024]
< https://kublr.com/blog/how-to-choose-the-right-tool-for-kubernetes-deployment/ >

[3] 6 Important things you need to run Kubernetes in production [Documentació en línia]
[Dia de la consulta: 10 de març de 2024]
<https://www.pionative.com/nl/post/6-important-things-you-need-to-run-kubernetes-in-production-1 >

[4] Documentació inicial de Kubernetes [Documentació en línia]
[Dia de la consulta: 15 de març de 2024]
< https://kubernetes.io/docs/concepts/overview/ >

[5] L'API de Kubernetes [Documentació en línia]
[Dia de la consulta: 15 de març de 2024]
< https://kubernetes.io/docs/concepts/overview/kubernetes-api/ >

[6] L'arquitectura de Kubernetes [Documentació en línia]
[Dia de la consulta: 15 de març de 2024]
< https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-architecture/ >

[7] Amazon Elastic Kubernetes Service [Documentació en línia]
[Dia de la consulta: 17 de març de 2024]
< https://aws.amazon.com/eks/ >

[8] Amazon EKS features [Documentació en línia]
[Dia de la consulta: 17 de març de 2024]
< https://aws.amazon.com/eks/features/ >

[9] Cluster Networking [Documentació en línia]
[Dia de la consulta: 19 de març de 2024]
< https://kubernetes.io/docs/concepts/cluster-administration/networking/ >

[10] Services in Kubernetes [Documentació en línia]
[Dia de la consulta: 19 de març de 2024]
< https://kubernetes.io/docs/concepts/services-networking/service/ >

[11] Kubernetes network model [Documentació en línia]
[Dia de la consulta: 19 de març de 2024]
< https://kubernetes.io/docs/concepts/services-networking/#the-kubernetes-network-model >

[12] Kubernetes Network Plugins [Documentació en línia]
[Dia de la consulta: 19 de març de 2024]
< https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/ >

[13] Best practices AWS VPC CNI [Documentació en línia]
[Dia de la consulta: 19 de març de 2024]
< https://aws.github.io/aws-eks-best-practices/networking/vpc-cni/ >

[14] Kube-Proxy: What Is It and How It Works [Documentació en línia]
[Dia de la consulta: 20 de març de 2024]
< https://kodekloud.com/blog/kube-proxy/ >

[15] K8s: A Closer Look at Kube-Proxy [Documentació en línia]
[Dia de la consulta: 20 de març de 2024]
< https://betterprogramming.pub/k8s-a-closer-look-at-kube-proxy-372c4e8b090 >

[16] What is Cilium? [Documentació en línia]
[Dia de la consulta: 20 de març de 2024]
< https://docs.cilium.io/en/stable/overview/intro/#what-is-cilium >

[17] Cilium IP Address Management [Documentació en línia]
[Dia de la consulta: 20 de març de 2024]
< https://docs.cilium.io/en/stable/network/concepts/ipam/ >

[18] Cilium kube-proxy replacement [Documentació en línia]
[Dia de la consulta: 20 de març de 2024]
< https://cilium.io/use-cases/kube-proxy/ >

[19] Cilium overview of Network Policy [Documentació en línia]
[Dia de la consulta: 20 de març de 2024]
< https://docs.cilium.io/en/latest/security/policy/ >

[20] Github Hubble [Documentació en línia]
[Dia de la consulta: 20 de març de 2024]
< https://github.com/cilium/hubble >

[21] About Calico [Documentació en línia]
[Dia de la consulta: 20 de març de 2024]
< https://docs.tigera.io/calico/latest/about/ >

[22] Cilium identity-based [Documentació en línia]
[Dia de la consulta: 23 de març de 2024]
< https://docs.cilium.io/en/stable/security/network/identity/ >

[23] Kubernetes 1.13 release notes [Documentació en línia]
[Dia de la consulta: 25 de març de 2024]
< https://discuss.kubernetes.io/t/kubernetes-io-blog-kubernetes-1-13-simplified-cluster-management-with-kubeadm-container-storage-interface-csi-and-coredns-as-default-dns-are-now-generally-available/3713 >

[24] CoreDNS plugins [Documentació en línia]
[Dia de la consulta: 25 de març de 2024]
< https://coredns.io/plugins/ >

[25] Customizing DNS Service [Documentació en línia]
[Dia de la consulta: 25 de març de 2024]
< https://kubernetes.io/docs/tasks/administer-cluster/dns-custom-nameservers/ >

[26] Using NodeLocal DNSCache in Kubernetes Clusters [Documentació en línia]
[Dia de la consulta: 25 de març de 2024]
< https://kubernetes.io/docs/tasks/administer-cluster/nodelocaldns/ >

[27] Horizontal Pod Autoscaling [Documentació en línia]
[Dia de la consulta: 26 de març de 2024]
< https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ >

[28] Vertical Pod Autoscaler [Documentació en línia]
[Dia de la consulta: 26 de març de 2024]
< https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler#intro >

[29] What is KEDA? [Documentació en línia]
[Dia de la consulta: 26 de març de 2024]
< https://keda.sh/ >

[30] What is Cluster Autoscaler? [Documentació en línia]
[Dia de la consulta: 27 de març de 2024]
< https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-is-cluster-autoscaler >

[31] Karpenter documentation [Documentació en línia]
[Dia de la consulta: 27 de març de 2024]
< https://karpenter.sh/docs/ >

[32] Terraform use cases [Documentació en línia]
[Dia de la consulta: 29 de març de 2024]
< https://developer.hashicorp.com/terraform/intro/use-cases >

[33] About Sceptre [Documentació en línia]
[Dia de la consulta: 29 de març de 2024]
< https://docs.sceptre-project.org/latest/#about >

[34] EKS VPC and subnet requirements and considerations [Documentació en línia]
[Dia de la consulta: 2 d'abril de 2024]
< https://docs.aws.amazon.com/eks/latest/userguide/network_reqs.html >

[35] Enabling secret encryption on an existing cluster [Documentació en línia]
[Dia de la consulta: 4 d'abril de 2024]
< https://docs.aws.amazon.com/eks/latest/userguide/enable-kms.html >

[36] Eksctl getting started [Documentació en línia]
[Dia de la consulta: 10 d'abril de 2024]
< https://eksctl.io/getting-started/ >

[37] Eksctl config file schema [Documentació en línia]
[Dia de la consulta: 14 d'abril de 2024]
< https://eksctl.io/usage/schema/ >

[38] Eksctl config file schema [Documentació en línia]
[Dia de la consulta: 15 d'abril de 2024]
< https://www.devopsschool.com/blog/a-quick-tutorial-of-helm/ >

[39] Cilium installation using Helm [Documentació en línia]
[Dia de la consulta: 16 d'abril de 2024]
< https://docs.cilium.io/en/stable/installation/k8s-install-helm/ >

[40] Kubernetes control plane unable to comunicate to metric server [Documentació en línia]
[Dia de la consulta: 25 d'abril de 2024]
< https://keda.sh/troubleshooting/keda-external-metrics-issue/ >

[41] Configure a Kubernetes service account to assume an IAM role [Documentació en línia]
[Dia de la consulta: 25 d'abril de 2024]
< https://docs.aws.amazon.com/eks/latest/userguide/associate-service-account-role.html >

[42] Keda – AWS SQS Queue [Documentació en línia]
[Dia de la consulta: 26 d'abril de 2024]
< https://keda.sh/docs/2.11/scalers/aws-sqs/ >

[43] Cilium - Masquerading [Documentació en línia]
[Dia de la consulta: 28 d'abril de 2024]
< https://docs.cilium.io/en/stable/network/concepts/masquerading/ >

[44] External Secrets – AWS SM [Documentació en línia]
[Dia de la consulta: 30 d'abril de 2024]
< https://external-secrets.io/latest/provider/aws-secrets-manager/ >

[45] Karpenter migrating from CAS [Documentació en línia]

[Dia de la consulta: 3 de maig de 2024]
< https://karpenter.sh/docs/getting-started/migrating-from-cas/ >

[46] Karpenter managing AMIs [Documentació en línia]
[Dia de la consulta: 5 de maig de 2024]
< https://karpenter.sh/docs/tasks/managing-amis/ >

[47] External-secrets FAQ [Documentació en línia]
[Dia de la consulta: 5 de maig de 2024]
< https://external-secrets.io/v0.9.17/introduction/faq/ >

10. Annexos

10.1: Anàlisi detallat de les solucions de xarxa a Kubernetes

10.1.1. AWS VPC CNI

L'Amazon VPC CNI és el component de xarxa per defecte que s'utilitza en els clústers d'EKS. Proporciona una integració profunda entre els pods del clúster i la infraestructura de xarxa d'Amazon (VPC), permetent als usuaris aprofitar els avantatges de la xarxa nativa d'AWS dins dels seus clústers de Kubernetes. A continuació detallaré el funcionament i algunes de les característiques que té aquest component. [13]

Assignació d'IPs natives d'AWS

Una de les principals característiques del VPC CNI és la seva capacitat per assignar adreces IP de VPC directament als pods d'EKS. Això significa que cada pod rep una adreça IP del mateix VPC, el que permet que els pods es puguin comunicar amb altres serveis d'AWS i recursos d'internet com si fossin instàncies EC2 independents. Aquest enfocament simplifica significativament la gestió de xarxes i la seguretat, ja que es poden aplicar les polítiques de seguretat d'AWS (com els grups de seguretat) directament als mateixos pods.

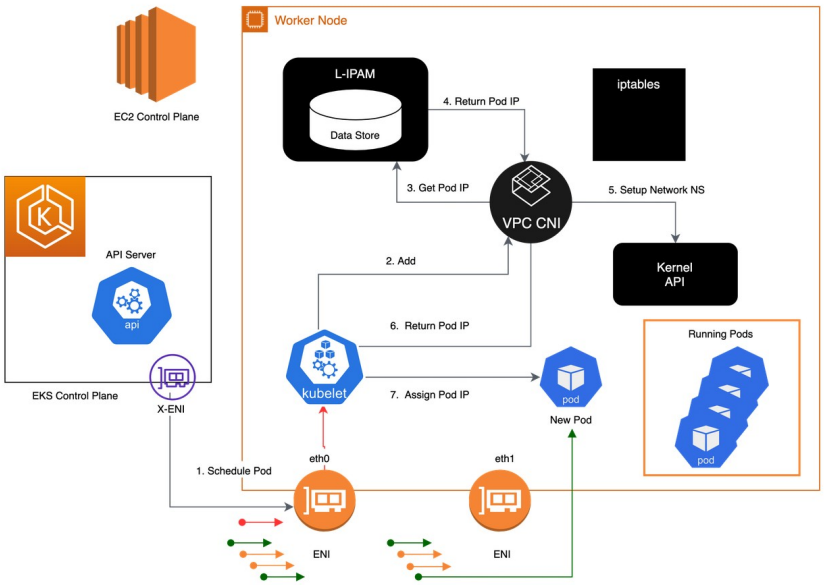


Figura 38: Procés d'assignació d'una IP a un pod, amb AWS VPC CNI

Gestió eficient de l'adreçament IP

El CNI gestiona un «pool» d'adreces IP per a cada node del clúster, assignant-les als pods a mesura que s'inicien i alliberant-les quan els pods es tanquen. Això s'aconsegueix mitjançant la creació d'interfícies de xarxa elàstiques (ENIs) addicionals i assignant múltiples adreces IP a aquestes ENIs en cada node. Aquest mètode permet una alta densitat de pods per node, això si subjecte als límits imposats per les tipologies d'instàncies EC2 que s'utilitzin com a nodes.

Integració amb AWS

El CNI d'AWS VPC està profundament integrat amb altres serveis d'AWS, com Amazon Route 53 per a la resolució de noms, AWS Identity and Access Management (IAM) per a la gestió d'identitats i permisos, i Amazon CloudWatch per a la monitorització i el registre. Això permet una gestió còmoda i segura de l'accés als recursos d'AWS i facilita la implementació de pràctiques de seguretat avançades.

Desavantatges

Això si, l'ús d'aquest component té els següents desavantatges:

1. **Consum d'adreces IP**: Cada pod necessita una adreça IP del VPC, el que pot esgotar ràpidament l'espai d'adreçament disponible en xarxes grans, especialment en clústers de gran escala.
2. **Gestió de recursos**: La gestió de les ENIs i adreces IP pot ser complexa, especialment en clústers amb una gran quantitat de nodes i pods, requerint una planificació atenta de la xarxa.
3. **Costos**: L'ús intensiu de recursos i la necessitat de xarxes més grans per acomodar una adreça IP per pod poden incrementar els costos operatius en AWS.
4. **Limitacions de les instàncies EC2**: El nombre màxim d'ENIs i adreces IP per instància està limitat per el tipus d'instància EC2, el que pot limitar el nombre de pods per node i requerir una planificació detallada per evitar colls d'ampolla.
5. **Complexitat operativa**: La configuració i el manteniment del CNI d'AWS VPC poden arribar a ser més complexos en comparació amb altres solucions.

L'Amazon VPC CNI es centra principalment en l'assignació de Ips per als pods, i que aquests es puguin comunicar entre ells i amb l'exterior. Però hi ha una característica que ens manca, els serveis. Un servei a Kubernetes és el que permet als pods comunicar-se a partir d'un nom abstracte, i no mitjançant la seva IP, que en un entorn distribuït com Kubernetes pot variar molt. Aquesta abstracció la duu a terme kube-proxy.

10.1.2. Kube-proxy

Kube-proxy és un component clau dins de l'arquitectura de Kubernetes, encarregat de gestionar els serveis de Kubernetes, facilitant la comunicació dins del clúster i l'exposició dels mateixos a l'exterior. Com el seu nom indica, fa una funció de *intermediari*. Algunes de les característiques principals de kube-proxy són les següents: [14]

Gestió de l'accés a serveis

Kube-proxy funciona com un controlador de trànsit de xarxa dins del clúster de Kubernetes, encarregat de redirigir les peticions cap als pods que ofereixen el servei sol·licitat. Utilitza IPTABLES, IPVS, o mecanismes de redireccionament de trànsit específics del nucli de Linux per garantir que les peticions als serveis arribin als pods correctes, independentment de la seva ubicació dins del clúster.

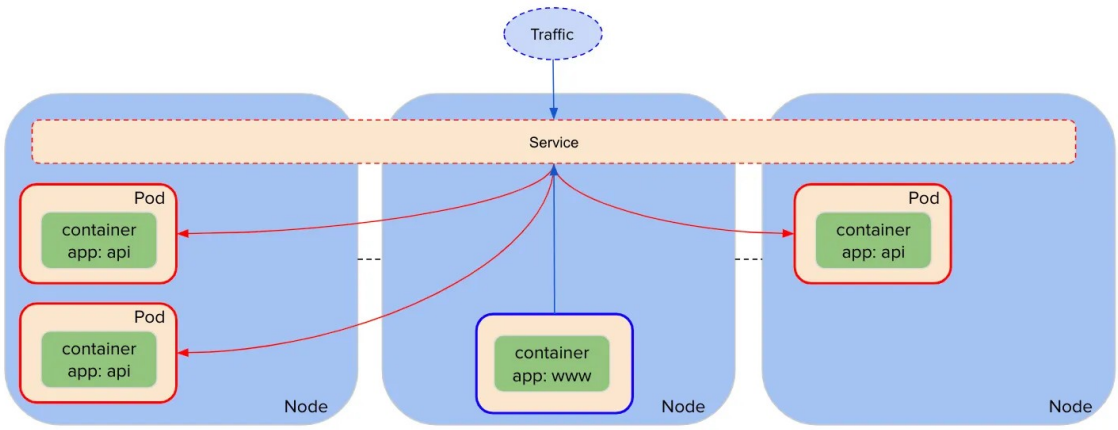


Figura 39: Funcionament d'un servei a K8s
[15]

Equilibri de Càrrega

Una de les funcions principals de kube-proxy és l'equilibri de càrrega del trànsit entrant cap als pods d'un servei. Distribueix les peticions entrants de manera

equitativa entre tots els pods que formen un servei, millorant així la disponibilitat i la fiabilitat de les aplicacions desplegades en Kubernetes.

Suport per a Múltiples Modes de Funcionament

kube-proxy pot funcionar en diversos modes, incloent el mode user space, el mode iptables, i el mode IPVS. Cada mode té els seus propis avantatges i desavantatges en termes de rendiment, escalabilitat, i complexitat de configuració. El mode més adequat pot variar segons les necessitats específiques de l'aplicació i de la infraestructura.

Seguretat i Aïllament

Tot i que no és la seva funció principal, kube-proxy juga un paper important en la seguretat del clúster, ja que les regles de redireccionament que estableix poden ser utilitzades per aplicar polítiques d'accés i garantir que només el trànsit autoritzat pugui arribar als serveis.

Inconvenients

1. **Rendiment a gran escala**: En modes com user space o iptables, kube-proxy pot convertir-se en un coll d'ampolla a mesura que la mida del clúster i el volum de trànsit augmenten, afectant el rendiment i la latència.
2. **Complexitat de gestió**: La configuració i optimització de kube-proxy, especialment en entorns de producció a gran escala, pot ser complexa i requereix un coneixement profund de la xarxa de Kubernetes.
3. **Limitacions del mode IPVS**: Tot i que el mode IPVS ofereix un millor rendiment per a grans quantitats de serveis, la seva configuració i depuració pot ser molt més complexa en comparació amb els altres modes.
4. **Dependència de característiques del nucli Linux**: Com que kube-proxy fa ús intensiu de les característiques del nucli de Linux com IPTABLES o IPVS, qualsevol limitació o bug en aquestes tecnologies pot afectar el seu funcionament.

10.1.3. Cilium

Cilium és una solució de codi obert avançada de xarxa, observabilitat i seguretat per a aplicacions en contenidors, com ara les desplegades en Kubernetes. Utilitza la tecnologia eBPF (Extended Berkeley Packet Filter) del kernel de Linux per injectar dinàmicament lògica de seguretat, visibilitat i control de xarxa directament dins del kernel, sense necessitat de canvis en les aplicacions o la configuració dels contenidors. [16] Algunes de les seves característiques són les següents:

Funcions de CNI i d'intermediari

Cilium pot realitzar tant les funcions de CNI (que duia a terme AWS VPC CNI en l'exemple anterior), com de intermediari (que duia a terme kube-proxy). Això es pot aconseguir utilitzant tant l'IPAM [17] com activant el mode de reemplaçament de kube-proxy [18] que, amb BPF, té una molt millor escalabilitat. També suporta models de xarxa d'encapsulació i rutes natives, oferint als usuaris flexibilitat per adaptar-se a diferents necessitats d'infraestructura.

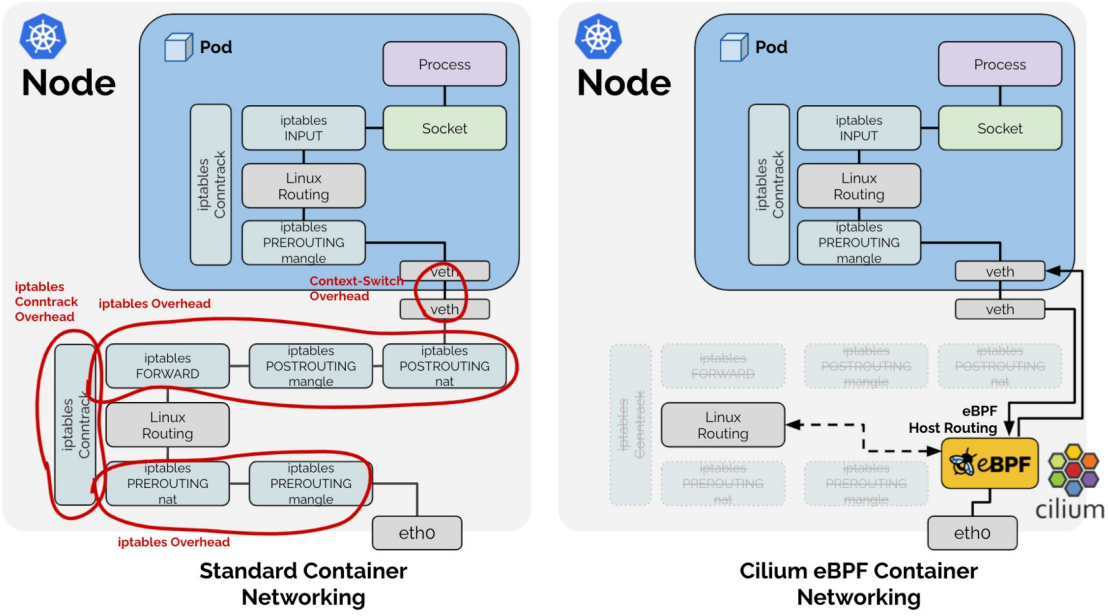


Figura 40: Diferències entre un mode d'iptables amb un de bpfilter

Seguretat transparent

Cilium fa ús d'identitats per aplicar polítiques de seguretat entre serveis. A diferència dels mètodes tradicionals que utilitzen adreces IP, les identitats permeten una gestió de polítiques més flexible i dinàmica, ideal per a entorns de microserveis on les IPs poden canviar freqüentment. [19]

Observabilitat detallada

Integrat amb *Hubble*, Cilium proporciona visibilitat detallada sobre la comunicació entre serveis, incloent el mapeig de dependències de serveis, el monitoratge de la xarxa, i la monitorització d'aplicacions. Això permet als administradors de sistemes i desenvolupadors entendre millor com interactuen els serveis i identificar problemes de rendiment o seguretat. [20]

eBPF com a base de tot

eBPF permet a Cilium injectar codi a nivell de kernel per realitzar funcions de xarxa, seguretat i observabilitat sense canviar els binaris del kernel ni reiniciar el sistema. Aquesta capacitat fa que Cilium sigui extremadament potent, eficient i flexible, capaç de respondre dinàmicament als canvis en l'entorn de xarxa.

Desavantatges

Alguns dels desafiaments a tindre en compte de Cilium són els següents:

1. **Complexitat**: Encara que Cilium ofereix una potència i flexibilitat molt considerables, la seva complexitat pot ser un repte per a equips sense experiència prèvia amb tecnologies com eBPF o amb els conceptes avançats de xarxa que utilitza.
2. **Requisits del Sistema**: Com que Cilium s'aprofita de les característiques avançades del kernel de Linux proporcionades per eBPF, pot haver-hi requisits específics de versió del kernel o capacitats de sistema que cal complir per poder utilitzar totes les seves funcionalitats.
3. **Dependència de la Tecnologia eBPF**: La forta dependència de Cilium en eBPF significa que el seu rendiment i capacitat estan lligats a les característiques i limitacions de eBPF. Això pot afectar la seva aplicabilitat en entorns on eBPF no estigui completament suportat o optimitzat.

10.1.4. Calico

Calico és una solució de xarxa i seguretat de codi obert per a contenidors, màquines virtuals i càrregues de treball natives del núvol. Com a projecte integrat dins de l'ecosistema de Kubernetes, Calico ofereix suport de xarxa de gran rendiment i funcions de seguretat de xarxa fines per a aplicacions en diversos entorns, incloent núvols privats, públics i híbrids. Utilitza un model de dades simple i eficient per aplicar polítiques de xarxa i seguretat, i és compatible amb una àmplia gamma de plataformes i núvols.

Les característiques principals de Calico inclouen: [21]

Funcionalitats de Xarxa Avançades

Calico proporciona funcionalitats de xarxa de nivell 3, utilitzant IP per IP sobre l'encaminament del protocol, que elimina la necessitat d'encapsulació i potencialment millora el rendiment. Suporta una àmplia varietat de configuracions de xarxa, incloent la possibilitat de funcionar en modes tant encapçalats com no encapçalats, així com la implementació de polítiques de xarxa fines.

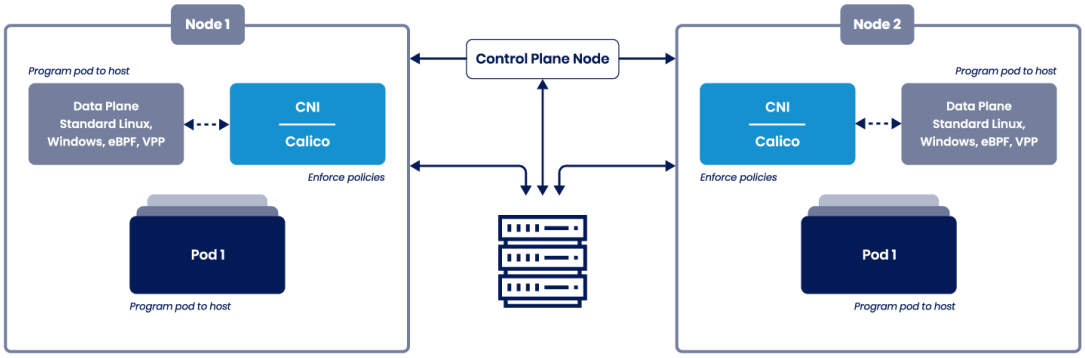


Figura 41: Arquitectura de Calico

Seguretat a nivell de xarxa i aplicació

Calico fa ús de polítiques de xarxa definides per l'usuari per controlar l'accés entre càrregues de treball dins de Kubernetes. Aquestes polítiques permeten als administradors de sistemes i desenvolupadors definir regles explícites per qui pot comunicar-se amb qui, millorant així la seguretat global de l'entorn.

Observabilitat i Monitoratge

Integració amb eines com Prometheus i Grafana per proporcionar monitoratge detallat i visibilitat de la xarxa. Calico també ofereix registres detallats i estadístiques que faciliten la identificació de problemes de rendiment o seguretat.

Escalabilitat i rendiment

Calico està dissenyat per escalar de manera eficient en entorns grans i complexos. La seva arquitectura sense controlador central elimina punts únics de fallida i garanteix que el rendiment de la xarxa escali linealment amb el nombre de càrregues de treball.

Desavantatges

Encara que Calico és una solució potent i flexible per a la gestió de la xarxa i la seguretat en Kubernetes, presenta alguns desafiaments:

1. **Curva d'aprenentatge:** Les nombroses funcionalitats i opcions de configuració de Calico poden ser aclaparadores per a nous usuaris o equips sense experiència en xarxes de nivell 3.
2. **Complexitat en entorns híbrids o multinúvol**: Gestionar la xarxa i les polítiques de seguretat en entorns híbrids o multinúvol pot ser més complex, requerint una planificació i coordinació detallades.
3. **Dependència dels recursos de la xarxa subjacent**: Semblant a Cilium, el rendiment i l'eficàcia de Calico poden variar depenent de la infraestructura de xarxa subjacent i de com està configurada, el que pot requerir ajustaments específics per optimitzar el rendiment.

10.2: Instal·lació i integració d'Sceptre amb el compte d'AWS

Per instal·lar Sceptre en la nostra màquina local podrem utilitzar pip.

```
❯ pip install sceptre -q

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python3.11 -m pip install --upgrade pip
❯
❯ sceptre --version
Sceptre, version 4.4.2
```

Figura 42: Comandes per instal·lar sceptre

Per integrar Sceptre amb un compte d'AWS haurem de crear a ma un bucket S3, a on es guardaran els stacks de Sceptre.

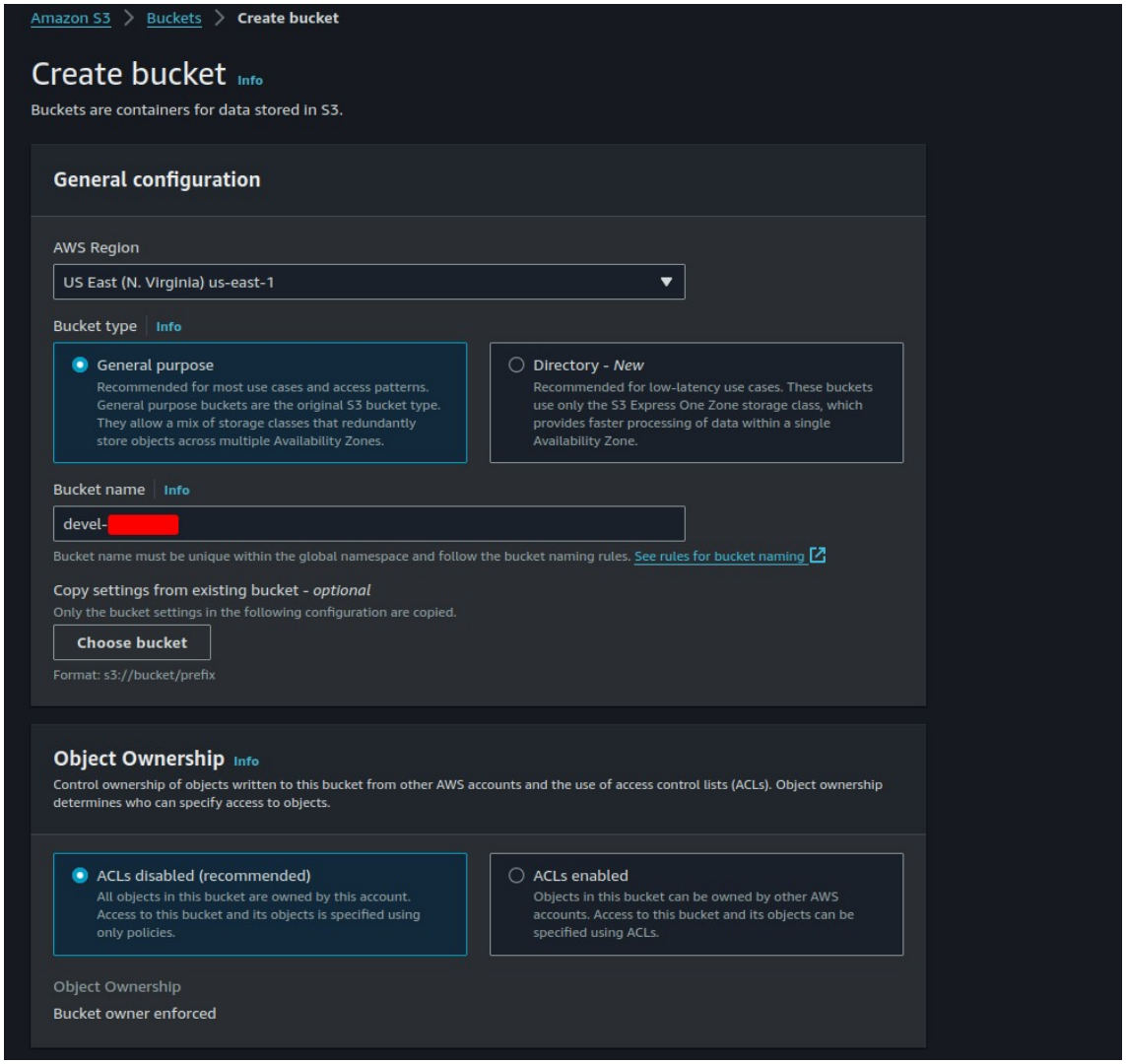


Figura 43: Creació del bucket S3 per a sceptre

No caldrà que modifiquem cap paràmetre dels que venen per defecte, únicament el nom del bucket (en el meu cas censuraré el sufix ja que els noms dels buckets són globals). Només haurem de tindre en compte que des de la nostra màquina local d'on executarem sceptre hem de tindre accés a aquest bucket (ja sigui fent el bucket públic encara que no sigui recomanable, o afegint polítiques d'accés al bucket).

Una vegada tinguem el bucket creat, a cada carpeta on tinguem un stack de sceptre, haurem d'afegir un config.yaml com aquest:

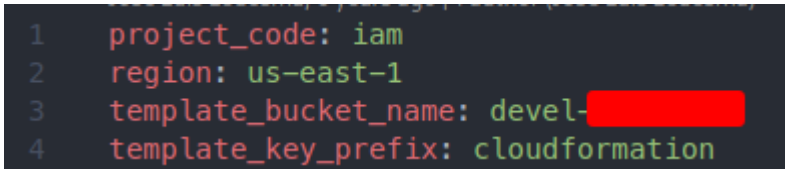


Figura 44: Exemple de YAML de configuració de sceptre

On project_code serà el codi de projecte de tots els stacks dins aquesta carpeta, la regió la zona d'AWS on es crearan en stacks en cas que no siguin serveis globals, el template_bucket_name el nom del bucket on es guardarà el codi dels stacks, i template_key_prefix un prefixe que podrem posar al nom dels stacks.

D'aquesta forma, cada vegada que creem, actualitzem o esborrem un stack, sceptre agafarà aquesta informació per trobar el bucket on es guardaran els stacks. De manera prèvia a qualsevol acció, necessitarem haver assumit un rol amb permisos suficients per a realitzar les accions que li demanem.

10.3: Creació del VPC i les subnets per al clúster

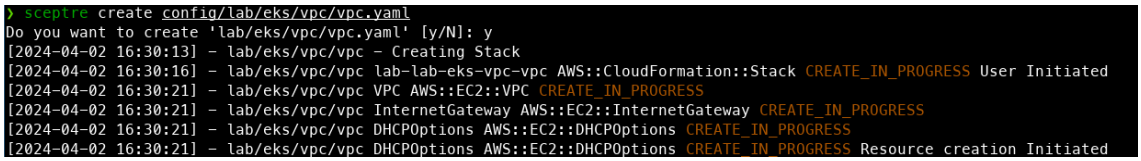
El codi del VPC utilitzat és el següent:

```
sceptre_user_data:
  Name: eks-test
  CidrBlock: 172.16.0.0/20
  Subnets:
    EksMastersA:
      az: us-east-1a
      private:
        cidr: 172.16.0.0/28
        extraTags:
          kubernetes.io/role/master: 1
    EksMastersB:
      az: us-east-1b
      private:
        cidr: 172.16.0.16/28
        extraTags:
          kubernetes.io/role/master: 1
    EksMastersC:
      az: us-east-1c
      private:
        cidr: 172.16.0.32/28
        extraTags:
          kubernetes.io/role/master: 1
    EksA:
      az: us-east-1a
      private:
        cidr: 172.16.4.0/22
        extraTags:
          kubernetes.io/role/internal-elb: 1
      nat: true
    EksB:
      az: us-east-1b
      private:
        cidr: 172.16.8.0/22
        extraTags:
          kubernetes.io/role/internal-elb: 1
      nat: true
    EksC:
      az: us-east-1c
      private:
        cidr: 172.16.12.0/22
        extraTags:
          kubernetes.io/role/internal-elb: 1
      nat: true
```

Figura 45: Codi utilitzat per a crear el VPC

Les tags s'han posat degut als requeriments especificats a la documentació [32].

Aquí la creació del VPC



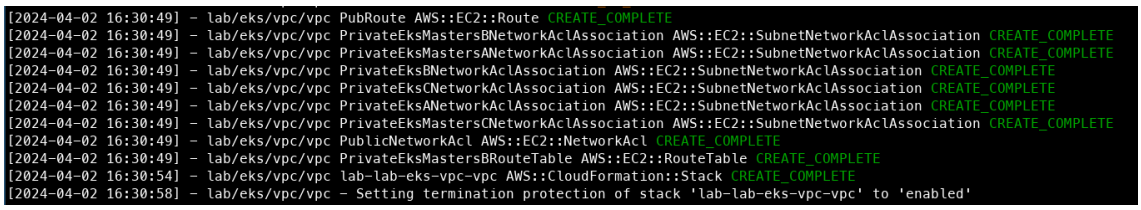


Figura 46: Creació del VPC amb sceptre

I aquí el codi CloudFormation del VPC generat per la template de sceptre:

```
❯ sceptre generate config/lab/eks/vpc/vpc.yaml
---
Parameters: {}
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: "172.16.0.0/20"
      EnableDnsHostnames: "true"
      EnableDnsSupport: "true"
      Tags:
        - Key: Application
          Value: {Ref: "AWS::StackName"}
        - Key: Name
          Value: "eks-test"

  DHCPOptions:
    Type: AWS::EC2::DHCPOptions
    Properties:
      DomainName: ec2.internal
      DomainNameServers: [AmazonProvidedDNS]
      Tags:
        - {Key: Name, Value: "eks-test-dhcp"}

  DHCPOptionsAssociation:
    Type: AWS::EC2::VPCDHCPOptionsAssociation
    Properties:
      DhcpOptionsId: {Ref: DHCPOptions}
      VpcId: {Ref: VPC}

  GatewayToInternet:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: {Ref: InternetGateway}
      VpcId: {Ref: VPC}

  InboundPublicNetworkAclEntry:
    Type: AWS::EC2::NetworkAclEntry
    Properties:
      CidrBlock: 0.0.0.0/0
      Egress: "false"
      NetworkAclId: {Ref: PublicNetworkAcl}
      PortRange: {From: "0", To: "65535"}
      Protocol: "-1"
      RuleAction: allow
      RuleNumber: "100"
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Application
```

```
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-gw"

OutboundPublicNetworkAclEntry:
  Type: AWS::EC2::NetworkAclEntry
  Properties:
    CidrBlock: 0.0.0.0/0
    Egress: "true"
    NetworkAclId: {Ref: PublicNetworkAcl}
    PortRange: {From: "0", To: "65535"}
    Protocol: "-1"
    RuleAction: allow
    RuleNumber: "100"

PublicNetworkAcl:
  Type: AWS::EC2::NetworkAcl
  Properties:
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-acl"
    VpcId: {Ref: VPC}
PubRoute:
  Type: AWS::EC2::Route
  DependsOn: GatewayToInternet
  Properties:
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: {Ref: InternetGateway}
    RouteTableId: {Ref: PubRouteTable}

PubRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-routes"
    VpcId: {Ref: VPC}
PrivateEksMastersA:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone: "us-east-1a"
    CidrBlock: "172.16.0.0/28"
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksMastersA"
      - Key: Network
        Value: Private
      - Key: region
        Value: "us-east-1a"
      - Key: vpc
        Value: "eks-test"
      - Key: kubernetes.io/role/master
        Value: "1"
    VpcId: {Ref: VPC}

PrivateEksMastersANetworkAclAssociation:
  Type: AWS::EC2::SubnetNetworkAclAssociation
  Properties:
    NetworkAclId: {Ref: PublicNetworkAcl}
    SubnetId: {Ref: "PrivateEksMastersA"}

PrivateEksMastersARouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: {Ref: PrivateEksMastersARouteTable}
```

```
    SubnetId: {Ref: "PrivateEksMastersA"}

PrivateEksMastersARouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksMastersA-routes"
    VpcId: {Ref: VPC}

PrivateEksMastersB:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone: "us-east-1b"
    CidrBlock: "172.16.0.16/28"
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksMastersB"
      - Key: Network
        Value: Private
      - Key: region
        Value: "us-east-1b"
      - Key: vpc
        Value: "eks-test"
      - Key: kubernetes.io/role/master
        Value: "1"
    VpcId: {Ref: VPC}

PrivateEksMastersBNetworkAclAssociation:
  Type: AWS::EC2::SubnetNetworkAclAssociation
  Properties:
    NetworkAclId: {Ref: PublicNetworkAcl}
    SubnetId: {Ref: "PrivateEksMastersB"}

PrivateEksMastersBRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: {Ref: PrivateEksMastersBRouteTable}
    SubnetId: {Ref: "PrivateEksMastersB"}

PrivateEksMastersBRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksMastersB-routes"
    VpcId: {Ref: VPC}

PrivateEksMastersC:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone: "us-east-1c"
    CidrBlock: "172.16.0.32/28"
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksMastersC"
      - Key: Network
        Value: Private
      - Key: region
        Value: "us-east-1c"
      - Key: vpc
        Value: "eks-test"
```

```
      - Key: kubernetes.io/role/master
        Value: "1"
    VpcId: {Ref: VPC}

PrivateEksMastersCNetworkAclAssociation:
  Type: AWS::EC2::SubnetNetworkAclAssociation
  Properties:
    NetworkAclId: {Ref: PublicNetworkAcl}
    SubnetId: {Ref: "PrivateEksMastersC"}

PrivateEksMastersCRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: {Ref: PrivateEksMastersCRouteTable}
    SubnetId: {Ref: "PrivateEksMastersC"}

PrivateEksMastersCRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksMastersC-routes"
    VpcId: {Ref: VPC}

PrivateEksA:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone: "us-east-1a"
    CidrBlock: "172.16.4.0/22"
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksA"
      - Key: Network
        Value: Private
      - Key: region
        Value: "us-east-1a"
      - Key: vpc
        Value: "eks-test"
      - Key: kubernetes.io/role/internal-elb
        Value: "1"
    VpcId: {Ref: VPC}

PrivateEksANetworkAclAssociation:
  Type: AWS::EC2::SubnetNetworkAclAssociation
  Properties:
    NetworkAclId: {Ref: PublicNetworkAcl}
    SubnetId: {Ref: "PrivateEksA"}

PrivateEksARouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: {Ref: PrivateEksARouteTable}
    SubnetId: {Ref: "PrivateEksA"}

PrivateEksARouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksA-routes"
    VpcId: {Ref: VPC}

PrivateEksB:
  Type: AWS::EC2::Subnet
  Properties:
```

```
    AvailabilityZone: "us-east-1b"
    CidrBlock: "172.16.8.0/22"
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksB"
      - Key: Network
        Value: Private
      - Key: region
        Value: "us-east-1b"
      - Key: vpc
        Value: "eks-test"
      - Key: kubernetes.io/role/internal-elb
        Value: "1"
    VpcId: {Ref: VPC}

PrivateEksBNetworkAclAssociation:
  Type: AWS::EC2::SubnetNetworkAclAssociation
  Properties:
    NetworkAclId: {Ref: PublicNetworkAcl}
    SubnetId: {Ref: "PrivateEksB"}

PrivateEksBRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: {Ref: PrivateEksBRouteTable}
    SubnetId: {Ref: "PrivateEksB"}

PrivateEksBRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksB-routes"
    VpcId: {Ref: VPC}

PrivateEksC:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone: "us-east-1c"
    CidrBlock: "172.16.12.0/22"
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Application
        Value: {Ref: "AWS::StackName"}
      - Key: Name
        Value: "eks-test-PrivateEksC"
      - Key: Network
        Value: Private
      - Key: region
        Value: "us-east-1c"
      - Key: vpc
        Value: "eks-test"
      - Key: kubernetes.io/role/internal-elb
        Value: "1"
    VpcId: {Ref: VPC}

PrivateEksCNetworkAclAssociation:
  Type: AWS::EC2::SubnetNetworkAclAssociation
  Properties:
    NetworkAclId: {Ref: PublicNetworkAcl}
    SubnetId: {Ref: "PrivateEksC"}

PrivateEksCRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: {Ref: PrivateEksCRouteTable}
    SubnetId: {Ref: "PrivateEksC"}
```

```
  PrivateEksCRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      Tags:
        - Key: Application
          Value: {Ref: "AWS::StackName"}
        - Key: Name
          Value: "eks-test-PrivateEksC-routes"
      VpcId: {Ref: VPC}


Outputs:
  VpcId:
    Value: !Ref VPC
    Export:
      Name: "eks-testId"
  PrivateEksMastersAId:
    Value: !Ref PrivateEksMastersA
    Export:
      Name: eks-test-PrivateEksMastersAId
  PrivateEksMastersAAz:
    Value:
      Fn::GetAtt : [ "PrivateEksMastersA", "AvailabilityZone" ]
    Export:
      Name: eks-test-PrivateEksMastersAAz
  PrivateEksMastersARouteTable:
    Value: !Ref PrivateEksMastersARouteTable
    Export:
      Name: "eks-test-PrivateEksMastersA-RouteTable"

  PrivateEksMastersBId:
    Value: !Ref PrivateEksMastersB
    Export:
      Name: eks-test-PrivateEksMastersBId
  PrivateEksMastersBAz:
    Value:
      Fn::GetAtt : [ "PrivateEksMastersB", "AvailabilityZone" ]
    Export:
      Name: eks-test-PrivateEksMastersBAz
  PrivateEksMastersBRouteTable:
    Value: !Ref PrivateEksMastersBRouteTable
    Export:
      Name: "eks-test-PrivateEksMastersB-RouteTable"

  PrivateEksMastersCId:
    Value: !Ref PrivateEksMastersC
    Export:
      Name: eks-test-PrivateEksMastersCId
  PrivateEksMastersCAz:
    Value:
      Fn::GetAtt : [ "PrivateEksMastersC", "AvailabilityZone" ]
    Export:
      Name: eks-test-PrivateEksMastersCAz
  PrivateEksMastersCRouteTable:
    Value: !Ref PrivateEksMastersCRouteTable
    Export:
      Name: "eks-test-PrivateEksMastersC-RouteTable"

  PrivateEksAId:
    Value: !Ref PrivateEksA
    Export:
      Name: eks-test-PrivateEksAId
  PrivateEksAAz:
    Value:
      Fn::GetAtt : [ "PrivateEksA", "AvailabilityZone" ]
    Export:
      Name: eks-test-PrivateEksAAz
  PrivateEksARouteTable:
    Value: !Ref PrivateEksARouteTable
    Export:
      Name: "eks-test-PrivateEksA-RouteTable"
```

```
PrivateEksBId:
  Value: !Ref PrivateEksB
  Export:
    Name: eks-test-PrivateEksBId
PrivateEksBAz:
  Value:
    Fn::GetAtt : [ "PrivateEksB", "AvailabilityZone" ]
  Export:
    Name: eks-test-PrivateEksBAz
PrivateEksBRouteTable:
  Value: !Ref PrivateEksBRouteTable
  Export:
    Name: "eks-test-PrivateEksB-RouteTable"

PrivateEksCId:
  Value: !Ref PrivateEksC
  Export:
    Name: eks-test-PrivateEksCId
PrivateEksCAz:
  Value:
    Fn::GetAtt : [ "PrivateEksC", "AvailabilityZone" ]
  Export:
    Name: eks-test-PrivateEksCAz
PrivateEksCRouteTable:
  Value: !Ref PrivateEksCRouteTable
  Export:
    Name: "eks-test-PrivateEksC-RouteTable"
```

Una vegada creat el VPC i les subnets, quedarien de la següent forma

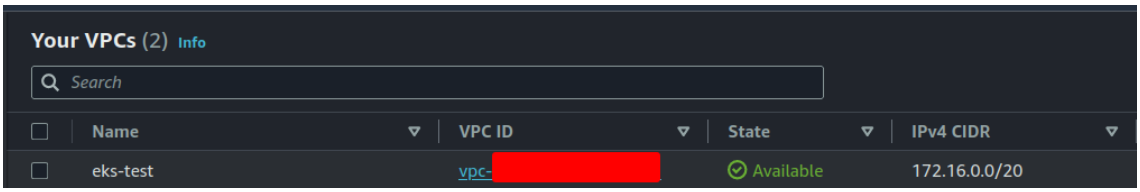


Figura 47: VPC resultant a la consola d'AWS

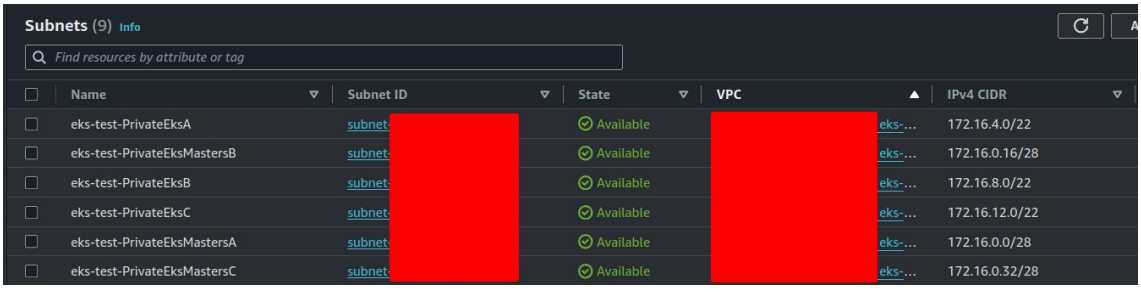


Figura 48: Subnets resultants a la consola d'AWS

I el security group per poder accedir al clúster (amb les IPs permeses censurades per raons de seguretat)

```yaml
sceptre_user_data:
  Name: SGExtPrivOffice
  FriendlyName: ext-privoffice-eks-test
  VPC: eks-test
  BuildRules:
    - CidrIps:
        -
        -
        -
      Rules:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
        - IpProtocol: icmp
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
        - IpProtocol: tcp
          FromPort: 443
          ToPort: 443
```

Figura 49: Codi YAML del SG que anirà al cluster

```
❯ sceptre create config/lab/eks/vpc/sg-ext-privoffice.yaml
Do you want to create 'lab/eks/vpc/sg-ext-privoffice.yaml' [y/N]: y
[2024-04-04 15:53:22] - lab/eks/vpc/sg-ext-privoffice - Creating Stack
[2024-04-04 15:53:25] - lab/eks/vpc/sg-ext-privoffice lab-lab-eks-vpc-sg-ext-privoffice AWS::C
loudFormation::Stack CREATE_IN_PROGRESS User Initiated
```

Figura 50: Creació amb sceptre del SG

I aquí el codi CloudFormation generat per la template de sceptre (amb dades censurades en vermell):

```
❯ sceptre generate config/lab/eks/vpc/sg-ext-privoffice.yaml
---
AWSTemplateFormatVersion: "2010-09-09"
Description: "Security group SGExtPrivOffice"
Resources:
  SGExtPrivOffice:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: ext-privoffice-eks-test
      Tags:
        - Key: "Name"
          Value: ext-privoffice-eks-test
      VpcId: !ImportValue "eks-testId"

  SGExtPrivOfficeIngress0:

      Type: "AWS::EC2::SecurityGroupIngress"
      DependsOn: "SGExtPrivOffice"
      Properties:
        GroupId: !Ref SGExtPrivOffice
        CidrIp:  XXXXXXXXX
        IpProtocol: tcp
        FromPort: 22
```

```
      ToPort: 22

  SGExtPrivOfficeIngress1:

     Type: "AWS::EC2::SecurityGroupIngress"
     DependsOn: "SGExtPrivOffice"
     Properties:
       GroupId: !Ref SGExtPrivOffice
       CidrIp:  XXXXXXXXX
       IpProtocol: icmp
       FromPort: -1
       ToPort: -1

  SGExtPrivOfficeIngress2:

     Type: "AWS::EC2::SecurityGroupIngress"
     DependsOn: "SGExtPrivOffice"
     Properties:
       GroupId: !Ref SGExtPrivOffice
       CidrIp:  XXXXXXXXX
       IpProtocol: tcp
       FromPort: 80
       ToPort: 80

  SGExtPrivOfficeIngress3:

     Type: "AWS::EC2::SecurityGroupIngress"
     DependsOn: "SGExtPrivOffice"
     Properties:
       GroupId: !Ref SGExtPrivOffice
       CidrIp:  XXXXXXXXX
       IpProtocol: tcp
       FromPort: 443
       ToPort: 443

Outputs:
  SGExtPrivOffice:
    Value: { Ref: SGExtPrivOffice}
    Export:
      Name: "eks-test-ext-privoffice-eks-test"
```

I finalment el security group quedaria així, amb les següents regles d'entrada:

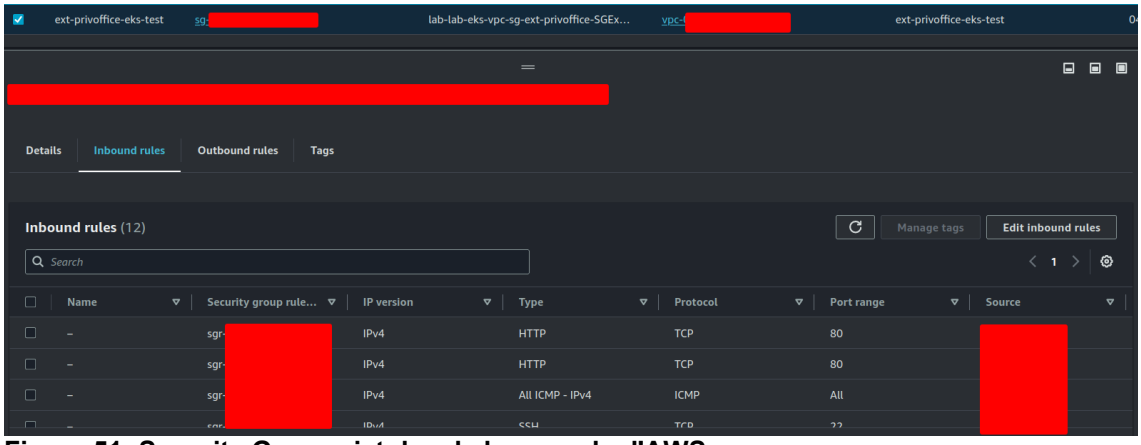


Figura 51: Security Group vist des de la consola d'AWS

10.4: Creació de la clau KMS per a la encriptació de secrets

Aquest és el codi utilitzat per a la creació de la clau KMS que s'utilitza per a l'encriptació de secrets:

```
sceptre_user_data:
  Description: "KMS key for encrypting EKS secrets"
  Name: "eks-test-kms"
  Account: "[redacted]"
  Environment: "lab"
```

Figura 52: Codi YAML de la clau KMS

```
❯ sceptre create config/lab/global-use1/kms/k8s-lab-cluster.yaml
Do you want to create 'lab/global-use1/kms/k8s-lab-cluster.yaml' [y/N]: y
[2024-04-04 17:19:00] - lab/global-use1/kms/k8s-lab-cluster - Creating Stack
[2024-04-04 17:19:03] - lab/global-use1/kms/k8s-lab-cluster kms-lab-global-use1-kms-k8s-lab-cl
uster AWS::CloudFormation::Stack CREATE_IN_PROGRESS User Initiated
sp-[2024-04-04 17:19:07] - lab/global-use1/kms/k8s-lab-cluster KMSKey AWS::KMS::Key CREATE_IN_
PROGRESS
[2024-04-04 17:19:07] - lab/global-use1/kms/k8s-lab-cluster KMSKey AWS::KMS::Key CREATE_IN_PRO
GRESS Resource creation Initiated
[2024-04-04 17:19:11] - lab/global-use1/kms/k8s-lab-cluster KMSKey AWS::KMS::Key CREATE_IN_PRO
GRESS Eventual consistency check initiated
[2024-04-04 17:19:11] - lab/global-use1/kms/k8s-lab-cluster KMSAlias AWS::KMS::Alias CREATE_IN
_PROGRESS
[2024-04-04 17:19:11] - lab/global-use1/kms/k8s-lab-cluster KMSAlias AWS::KMS::Alias CREATE_IN
_PROGRESS Resource creation Initiated
[2024-04-04 17:19:24] - lab/global-use1/kms/k8s-lab-cluster KMSKey AWS::KMS::Key CREATE_COMPLE
TE
[2024-04-04 17:20:12] - lab/global-use1/kms/k8s-lab-cluster KMSAlias AWS::KMS::Alias CREATE_CO
MPLETE
[2024-04-04 17:20:12] - lab/global-use1/kms/k8s-lab-cluster kms-lab-global-use1-kms-k8s-lab-cl
uster AWS::CloudFormation::Stack CREATE_COMPLETE
```

Figura 53: Creació amb sceptre de la clau KMS

I aquí el codi CloudFormation generat per la template de sceptre (amb dades censurades en vermell):

```
❯ sceptre generate config/lab/global-use1/kms/k8s-lab-cluster.yaml
---
AWSTemplateFormatVersion: '2010-09-09'
Description: "KMS key for encrypting EKS secrets"
Resources:
  KMSKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: KMS key for encrypting EKS secrets
      Enabled: true
      EnableKeyRotation: false
      Tags:
        - Key: Name
          Value: eks-test-kms
        - Key: environment
          Value: lab
      KeyPolicy:
        Version: "2012-10-17"
        Id: "key-consolepolicy-3"
        Statement:
          - Sid: "Enable IAM User Permissions"
            Effect: "Allow"
            Principal:
              "AWS": "arn:aws:iam::XXXXXXXXX:root"
            Action: "kms:*"
            Resource: "*"
```

```
        - Sid: "Allow access for Key Administrators to Root"
          Effect: "Allow"
          Principal:
            "AWS": "arn:aws:iam::XXXXXXXXX:root"
          Action:
            - "kms:Create*"
            - "kms:Describe*"
            - "kms:Enable*"
            - "kms:List*"
            - "kms:Put*"
            - "kms:Update*"
            - "kms:Revoke*"
            - "kms:Disable*"
            - "kms:Get*"
            - "kms:Delete*"
            - "kms:ScheduleKeyDeletion"
            - "kms:CancelKeyDeletion"
            - "kms:TagResource"
            - "kms:UntagResource"
          Resource: "*"
        - Sid: "Allow use of the key"
          Effect: "Allow"
          Principal:
            "AWS": "arn:aws:iam::XXXXXXXXX:root"
          Action:
            - "kms:Encrypt"
            - "kms:Decrypt"
            - "kms:ReEncrypt*"
            - "kms:GenerateDataKey*"
            - "kms:DescribeKey"
          Resource: "*"
        - Sid: "Allow attachment of persistent resources"
          Effect: "Allow"
          Principal:
            "AWS": "arn:aws:iam::XXXXXXXXX:root"
          Action:
            - "kms:CreateGrant"
            - "kms:ListGrants"
            - "kms:RevokeGrant"
          Resource: "*"
          Condition:
            Bool:
              "kms:GrantIsForAWSResource": "true"
  KMSAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: alias/eks-test-kms
      TargetKeyId: !Ref KMSKey
Outputs:
  KMSKeyARN:
    Description: "ARN of kms key"
    Value:
      Fn::GetAtt:
        - KMSKey
        - Arn
    Export:
      Name: KMSeks-test-kmsARN
```

I la clau generada a AWS

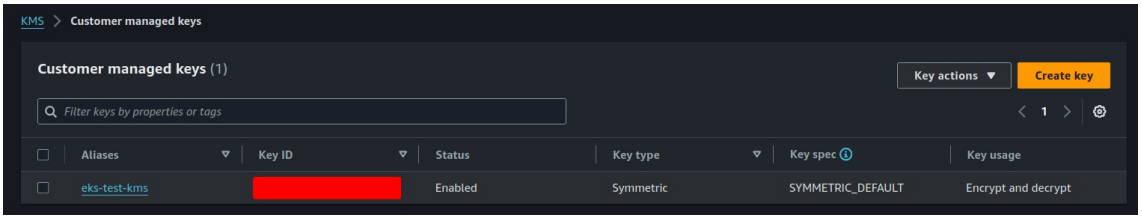


Figura 54: Clau KMS creada vista des de la consola d'AWS

10.5: Creació del clúster de Kubernetes

Per a la creació del clúster de Kubernetes, s'ha creat un fitxer YAML seguint la plantilla oficial d'eksctl per a la configuració els clústers. Aquesta ha quedat de la següent manera (amb dades sensibles censurades en vermell): [36]

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: test
  region: us-east-1
  version: "1.29"
  tags:
    product: eks
    role: test
vpc:
  id: "vpc-XXXXXXXXX"
  subnets:
    private:
      us-east-1a: { id: "subnet-XXXXXXXXX" }
      us-east-1b: { id: "subnet-XXXXXXXXX" }
      us-east-1c: { id: "subnet-XXXXXXXXX" }
  publicAccessCIDRs:
    - XX.XX.XX.XXX/32
  clusterEndpoints:
    privateAccess: true
    publicAccess: true
kubernetesNetworkConfig:
  serviceIPv4CIDR: 172.31.0.0/16
secretsEncryption:
  keyARN: arn:aws:kms:us-east-1:XXXXXXXXX:alias/eks-test-kms
managedNodeGroups:
  - name: a-ondemand-129-001
    ami: ami-0000a441e1aeb64b2
    amiFamily: AmazonLinux2
    instanceTypes:
      - m5.xlarge
      - m5a.xlarge
      - m5ad.xlarge
      - m5d.xlarge
      - m5dn.xlarge
      - m5n.xlarge
      - m6i.xlarge
    subnets:
      - subnet- XXXXXXXXX
    privateNetworking: true
    minSize: 1
    maxSize: 10
    desiredCapacity: 1
    volumeSize: 100
    volumeType: gp3
    volumeEncrypted: true
    securityGroups:
      attachIDs:
        - sg- XXXXXXXXX


  - name: b-ondemand-129-001
    ami: ami-0000a441e1aeb64b2
    amiFamily: AmazonLinux2
    instanceTypes:
      - m5.xlarge
      - m5a.xlarge
      - m5ad.xlarge
      - m5d.xlarge
      - m5dn.xlarge
      - m5n.xlarge
      - m6i.xlarge
```

```
  subnets:
    - subnet- XXXXXXXXX
  privateNetworking: true
  minSize: 1
  maxSize: 10
  desiredCapacity: 1
  volumeSize: 100
  volumeType: gp3
  volumeEncrypted: true
  securityGroups:
    attachIDs:
      - sg- XXXXXXXXX


- name: c-ondemand-129-001
  ami: ami-0000a441e1aeb64b2
  amiFamily: AmazonLinux2
  instanceTypes:
    - m5.xlarge
    - m5a.xlarge
    - m5ad.xlarge
    - m5d.xlarge
    - m5dn.xlarge
    - m5n.xlarge
    - m6i.xlarge
  subnets:
    - subnet- XXXXXXXXX
  privateNetworking: true
  minSize: 1
  maxSize: 10
  desiredCapacity: 1
  volumeSize: 100
  volumeType: gp3
  volumeEncrypted: true
  securityGroups:
    attachIDs:
      - sg- XXXXXXXXX
```

Diferents coses a analitzar en el codi utilitzat:

- S'utilitza la versió 1.29 de Kubernetes, la més recent a dia de la realització del treball.
- S'han utilitzat els IDs tant de VPC, subnets, security groups... creats prèviament.
- Per a obtindre l'identificador de la AMI utilitzada, s'ha fet amb la següent comanda

```
❯ aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.29/amazon-linux-2/recommended/
image_id --region us-east-1 --query "Parameter.Value" --output text

ami-0000a441e1aeb64b2
```

Figura 55: Obtenció de l'última AMI amb la comanda d'AWS

- Els nodes tenen el sistema operatiu «Amazon Linux 2», el recomenat per AWS.
- Sobre els tipus d'instàncies, s'ha seleccionat un grup d'instàncies amb 16Gb de RAM. AWS afegirà nodes d'aquest grup, segons la disponibilitat.
- S'ha creat un grup de nodes per a cada zona de disponibilitat (a, b i c).

En els següent passos utilitzarem dues eines: «eksctl» i «kubectl», aquestes eines es poden instal·lar des dels respectius webs, o utilitzant el gestor de paquets del sistema operatiu que s'estigui utilitzant. En el meu cas utilitzaré «pacman», el gestor de paquets d'Arch Linux.

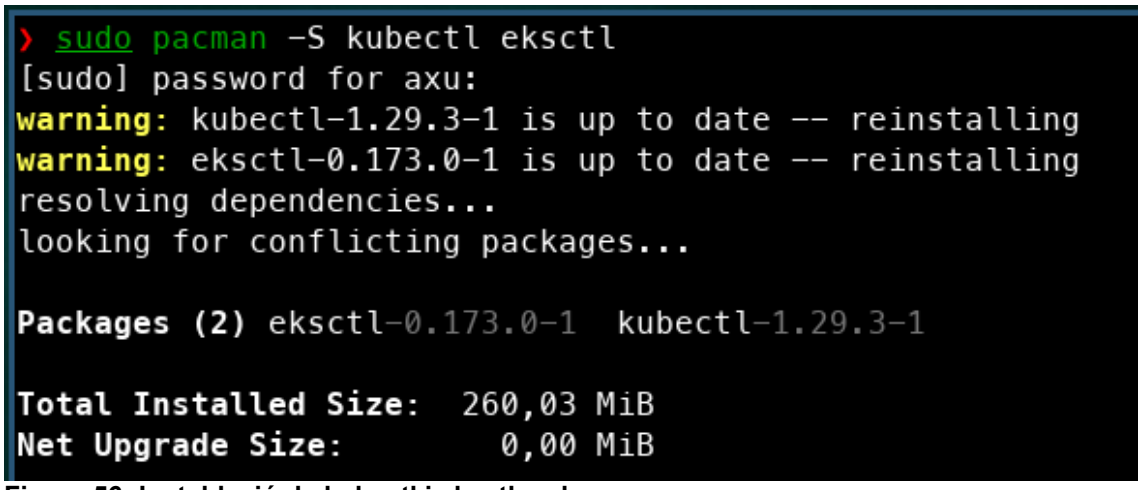


Figura 56: Instal·lació de kubectl i eksctl amb pacman

Per crear el clúster mitjançant eksctl s'ha utilitzat la següent comanda:

```
❯ eksctl create cluster -f eksctl.yaml
2024-04-14 19:39:29 [ℹ]  eksctl version 0.173.0-dev
2024-04-14 19:39:29 [ℹ]  using region us-east-1
2024-04-14 19:39:30 [✔]  using existing VPC (vpc-XXXXXXXXX) and subnets (private:map[us-east-1a:{subnet-XXXXXXXXX us-east-1a 172.16.0.0/28 0 } us-east-1b:{subnet- XXXXXXXXX us-east-1b 172.16.0.16/28 0 } us-east-1c:{subnet- XXXXXXXXX us-east-1c 172.16.0.32/28 0 }] public:map[])
2024-04-14 19:39:30 [!]  custom VPC/subnets will be used; if resulting cluster doesn't function as expected, make sure to review the configuration of VPC/subnets
2024-04-14 19:39:30 [ℹ]  nodegroup "a-ondemand-129-001" will use "ami-0000a441e1aeb64b2" [AmazonLinux2/1.29]
2024-04-14 19:39:31 [ℹ]  nodegroup "b-ondemand-129-001" will use "ami-0000a441e1aeb64b2" [AmazonLinux2/1.29]
2024-04-14 19:39:31 [ℹ]  nodegroup "c-ondemand-129-001" will use "ami-0000a441e1aeb64b2" [AmazonLinux2/1.29]
2024-04-14 19:39:31 [ℹ]  using Kubernetes version 1.29
2024-04-14 19:39:31 [ℹ]  creating EKS cluster "test" in "us-east-1" region with managed nodes
2024-04-14 19:39:31 [ℹ]  3 nodegroups (a-ondemand-129-001, b-ondemand-129-001, c-ondemand-129-001) were included (based on the include/exclude rules)
2024-04-14 19:39:31 [ℹ]  will create a CloudFormation stack for cluster itself and 0 nodegroup stack(s)
2024-04-14 19:39:31 [ℹ]  will create a CloudFormation stack for cluster itself and 3 managed nodegroup stack(s)
2024-04-14 19:39:31 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --cluster=test'
2024-04-14 19:39:31 [ℹ]  Kubernetes API endpoint access will use provided values {publicAccess=true, privateAccess=true} for cluster "test" in "us-east-1"
2024-04-14 19:39:31 [ℹ]  CloudWatch logging will not be enabled for cluster "test" in "us-east-1"
2024-04-14 19:39:31 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-east-1 --cluster=test'
2024-04-14 19:39:31 [ℹ]
2 sequential tasks: { create cluster control plane "test",
    2 sequential sub-tasks: {
        4 sequential sub-tasks: {
            wait for control plane to become ready,
            associate IAM OIDC provider,
            1 parallel sub-tasks: {
                2 sequential sub-tasks: {
                    create IAM role for serviceaccount "kube-system/aws-node",
                    create serviceaccount "kube-system/aws-node",
                },
```

```
        },
        restart daemonset "kube-system/aws-node",
      },
      3 parallel sub-tasks: {
        create managed nodegroup "a-ondemand-129-001",
        create managed nodegroup "b-ondemand-129-001",
        create managed nodegroup "c-ondemand-129-001",
      },
    }
}
2024-04-14 19:39:31 [ℹ]  building cluster stack "eksctl-test-cluster"
2024-04-14 19:39:33 [ℹ]  deploying stack "eksctl-test-cluster"
2024-04-14 19:40:03 [ℹ]  waiting for CloudFormation stack "eksctl-test-cluster"
.
.
.
2024-04-14 19:51:49 [ℹ]  waiting for CloudFormation stack "eksctl-test-cluster"
2024-04-14 19:53:54 [ℹ]  building iamserviceaccount stack "eksctl-test-addon-iamserviceaccount-kube-system-aws-node"
2024-04-14 19:53:55 [ℹ]  deploying stack "eksctl-test-addon-iamserviceaccount-kube-system-aws-node"
2024-04-14 19:54:26 [ℹ]  waiting for CloudFormation stack "eksctl-test-addon-iamserviceaccount-kube-system-aws-node"
2024-04-14 19:54:26 [ℹ]  serviceaccount "kube-system/aws-node" already exists
2024-04-14 19:54:26 [ℹ]  updated serviceaccount "kube-system/aws-node"
2024-04-14 19:55:26 [ℹ]  daemonset "kube-system/aws-node" restarted
2024-04-14 19:55:27 [ℹ]  building managed nodegroup stack "eksctl-test-nodegroup-c-ondemand-129-001"
2024-04-14 19:55:27 [ℹ]  building managed nodegroup stack "eksctl-test-nodegroup-b-ondemand-129-001"
2024-04-14 19:55:28 [ℹ]  building managed nodegroup stack "eksctl-test-nodegroup-a-ondemand-129-001"
2024-04-14 19:55:29 [ℹ]  deploying stack "eksctl-test-nodegroup-c-ondemand-129-001"
2024-04-14 19:55:29 [ℹ]  deploying stack "eksctl-test-nodegroup-b-ondemand-129-001"
2024-04-14 19:55:29 [ℹ]  waiting for CloudFormation stack "eksctl-test-nodegroup-c-ondemand-129-001"
2024-04-14 19:55:29 [ℹ]  deploying stack "eksctl-test-nodegroup-a-ondemand-129-001"
2024-04-14 19:55:29 [ℹ]  waiting for CloudFormation stack "eksctl-test-nodegroup-b-ondemand-129-001"
2024-04-14 19:55:29 [ℹ]  waiting for CloudFormation stack "eksctl-test-nodegroup-a-ondemand-129-001"
2024-04-14 19:55:59 [ℹ]  waiting for CloudFormation stack "eksctl-test-nodegroup-c-ondemand-129-001"
...
2024-04-14 19:58:27 [ℹ]  waiting for CloudFormation stack "eksctl-test-nodegroup-b-ondemand-129-001"
2024-04-14 19:58:27 [ℹ]  waiting for the control plane to become ready
2024-04-14 19:58:27 [✔]  saved kubeconfig as "/home/axu/.kube/config"
2024-04-14 19:58:27 [ℹ]  no tasks
2024-04-14 19:58:27 [✔]  all EKS cluster resources for "test" have been created
2024-04-14 19:58:28 [ℹ]  nodegroup "a-ondemand-129-001" has 1 node(s)
2024-04-14 19:58:28 [ℹ]  node "ip-172-16-5-154.ec2.internal" is ready
2024-04-14 19:58:28 [ℹ]  waiting for at least 1 node(s) to become ready in "a-ondemand-129-001"
2024-04-14 19:58:28 [ℹ]  nodegroup "a-ondemand-129-001" has 1 node(s)
2024-04-14 19:58:28 [ℹ]  node "ip-172-16-5-154.ec2.internal" is ready
2024-04-14 19:58:28 [ℹ]  nodegroup "b-ondemand-129-001" has 1 node(s)
2024-04-14 19:58:28 [ℹ]  node "ip-172-16-9-240.ec2.internal" is ready
2024-04-14 19:58:28 [ℹ]  waiting for at least 1 node(s) to become ready in "b-ondemand-129-001"
2024-04-14 19:58:29 [ℹ]  nodegroup "b-ondemand-129-001" has 1 node(s)
2024-04-14 19:58:29 [ℹ]  node "ip-172-16-9-240.ec2.internal" is ready
2024-04-14 19:58:29 [ℹ]  nodegroup "c-ondemand-129-001" has 1 node(s)
2024-04-14 19:58:29 [ℹ]  node "ip-172-16-13-249.ec2.internal" is ready
2024-04-14 19:58:29 [ℹ]  waiting for at least 1 node(s) to become ready in "c-ondemand-129-001"
2024-04-14 19:58:29 [ℹ]  nodegroup "c-ondemand-129-001" has 1 node(s)
2024-04-14 19:58:29 [ℹ]  node "ip-172-16-13-249.ec2.internal" is ready
2024-04-14 19:58:30 [ℹ]  kubectl command should work with "/home/axu/.kube/config", try 'kubectl get nodes'
2024-04-14 19:58:30 [✔]  EKS cluster "test" in "us-east-1" region is ready
```

Una vegada creat el clúster, podem provar a interactuar amb el clúster utilitzant kubectl. Podem utilitzar la variable d'entorn *KUBECONFIG* per especificar el fitxer de configuració que ens ha generat eksctl.

```
❯ export KUBECONFIG="/home/[redacted]/.kubeuoc/config"
❯ kubectl get node
NAME                              STATUS   ROLES    AGE   VERSION
ip-172-16-13-249.ec2.internal   Ready    <none>   22m   v1.29.0-eks-5e0fdde
ip-172-16-5-154.ec2.internal    Ready    <none>   22m   v1.29.0-eks-5e0fdde
ip-172-16-9-240.ec2.internal    Ready    <none>   22m   v1.29.0-eks-5e0fdde
```

Figura 57: Obtenció dels nodes mitjançant kubectl

Podem veure que tenim connexió amb el clúster, i podem veure que els nodes s'han creat correctament.

10.6: Instal·lació i configuració de Cilium

Primer de tot, desinstal·larem completament el CNI d'AWS, ja que no pot conviure amb Cilium degut a que Cilium realitzarà la mateixa assignació d'IPs que duu a terme el CNI.

```
❯   for kind in daemonSet clusterRoleBinding clusterRole serviceAccount; do
    kubectl -n kube-system delete $kind aws-node --wait=true
  done
daemonset.apps "aws-node" deleted
Warning: deleting cluster-scoped resources, not scoped to the provided namespace
clusterrolebinding.rbac.authorization.k8s.io "aws-node" deleted
Warning: deleting cluster-scoped resources, not scoped to the provided namespace
clusterrole.rbac.authorization.k8s.io "aws-node" deleted
serviceaccount "aws-node" deleted
```

Figura 58: Desinstal·lació del CNI d'AWS amb kubectl

Ara podem veure, que si creem un pod dummy, com que el clúster no li pot assignar una IP, aquest no podrà aixecar mai.

```
❯ k run nginx --image=nginx --restart=Never
pod/nginx created
❯ k get pod
NAME    READY   STATUS    RESTARTS   AGE
nginx   0/1     Pending   0          2s
❯ k get pod
NAME    READY   STATUS    RESTARTS   AGE
nginx   0/1     Pending   0          8s
```

Figura 59: Prova d'aixecar un pod, sense cap CNI instal·lat

El pod queda en estat «pending». És el comportament esperat.

Eliminarem també el kube-proxy, ja que Cilium amb l'ús de eBPF el suplirà completament.

```
❯ kubectl --ignore-not-found=true -n kube-system delete daemonset kube-proxy
daemonset.apps "kube-proxy" deleted
```

Figura 60: Desinstal·lació de kube-proxy amb kubectl

Ja podem instal·lar Cilium sense conflictes. Primer de tot afegirem el repositori de Cilium al nostre Helm. [39]

```
❯ helm repo add cilium https://helm.cilium.io/
"cilium" has been added to your repositories
```

Figura 61: Addició del repository de Cilium a helm

Ara prepararem el fitxer de valors que utilitzarà helm per a la instal·lació. Utilitzarem els valors per defecte, amb les següents modificacions:

```yaml
operator:
  rollOutPods: true # En cas que es modifiqui algun valor de configuració, els pods es reiniciaran
  replicas: 2 # Podríem tindre més rèpliques, per una alta disponibilitat

hubble:
  enabled: true # Habilitem hubble per tindre visibilitat dels paquets
  relay:
    enabled: true
  ui:
    enabled: true
    rollOutPods: true

kubeProxyReplacement: true # Activem el mode de reemplaçar el kube-proxy

localRedirectPolicy: true # Necessari per utilitzar eBPF

k8sServiceHost: BD6B398084A3B2FE7D5021F6531282A1.gr7.us-east-1.eks.amazonaws.com # L'endpoint del clúster
k8sServicePort: 443

rollOutCiliumPods: true

bpf:
  hostLegacyRouting: false # Habilitem el mode eBPF
  masquerade: true

ipam:
  mode: cluster-pool
  operator:
    clusterPoolIPv4PodCIDRList: ["100.64.0.0/10"]
    clusterPoolIPv4MaskSize: 24
```

Figura 62: Paràmetres modificats per a la instal·lació de cilium

Molts dels valors tenen comentaris explanatoris,

Per instal·lar cilium hem utilitzat la comanda de helm install, indicant-li el nostre fitxer de values.yaml

```
❯ helm install --namespace kube-system cilium cilium/cilium --version 1.15.4 -f values.yaml
NAME: cilium
LAST DEPLOYED: Tue Apr 16 14:19:53 2024
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
You have successfully installed Cilium with Hubble Relay and Hubble UI.

Your release version is 1.15.4.

For any further help, visit https://docs.cilium.io/en/v1.15/gettinghelp
```

Figura 63: Instal·lació de cilium amb helm

Una vegada instal·lat, podem veure que els pods s'han aixecat correctament

```
❯ k get deployment -n kube-system
k NAME              READY   UP-TO-DATE   AVAILABLE   AGE
cilium-operator   2/2     2            2           2m12s
coredns           2/2     2            2           42h
hubble-relay      1/1     1            1           2m12s
hubble-ui         1/1     1            1           2m12s
❯ k get pod -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
cilium-69vv4                        1/1     Running   0          2m16s
cilium-9z7j7                        1/1     Running   0          2m16s
cilium-operator-64d76b5684-786tv    1/1     Running   0          2m16s
cilium-operator-64d76b5684-g5mcl    1/1     Running   0          2m16s
cilium-wtzjt                        1/1     Running   0          2m16s
coredns-54d6f577c6-gtjdq            1/1     Running   0          115s
coredns-54d6f577c6-vkl2c            1/1     Running   0          2m10s
hubble-relay-78c7874676-xx9hk       1/1     Running   0          2m16s
hubble-ui-db9b7b844-w8dtr           2/2     Running   0          2m16s
```

Figura 64: Prova de crear pods amb Cilium ja instal·lat

I executant la comanda de cilium status dins un dels pods, podem veure l'estat de cilium

```
❯ k exec -ti cilium-9z7j7 -n kube-system -- cilium status
Defaulted container "cilium-agent" out of: cilium-agent, config (init), mount-cgroup (init), apply-sysctl-overwrites (
es (init)
KVStore:                 Ok   Disabled
Kubernetes:              Ok   1.29+ (v1.29.1-eks-b9c9ed7) [linux/amd64]
Kubernetes APIs:         ["EndpointSliceOrEndpoint", "cilium/v2::CiliumClusterwideNetworkPolicy", "cilium/v2::CiliumEn
kPolicy", "cilium/v2::CiliumNode", "cilium/v2alpha1::CiliumCIDRGroup", "core/v1::Namespace", "core/v1::Pods", "core/v1
KubeProxyReplacement:    True   [eth0   172.16.13.249 fe80::c69:2cff:fe45:9015 (Direct Routing), eth1   172.16.13.144
Host firewall:           Disabled
SRv6:                    Disabled
CNI Chaining:            none
Cilium:                  Ok   1.15.4 (v1.15.4-9b3f9a8c)
NodeMonitor:             Listening for events on 4 CPUs with 64x4096 of shared memory
Cilium health daemon:    Ok
IPAM:                    IPv4: 6/254 allocated from 100.64.0.0/24,
IPv4 BIG TCP:            Disabled
IPv6 BIG TCP:            Disabled
BandwidthManager:        Disabled
Host Routing:            BPF
Masquerading:            BPF   [eth0, eth1]   100.64.0.0/24 [IPv4: Enabled, IPv6: Disabled]
Controller Status:       39/39 healthy
Proxy Status:            OK, ip 100.64.0.183, 0 redirects active on ports 10000-20000, Envoy: embedded
Global Identity Range:   min 256, max 65535
Hubble:                  Ok              Current/Max Flows: 1203/4095 (29.38%), Flows/s: 10.09   Metrics: Disabled
Encryption:              Disabled
Cluster health:          3/3 reachable   (2024-04-16T12:21:27Z)
Modules Health:          Stopped(0) Degraded(0) OK(11) Unknown(3)
```

Figura 65: Comprovació de l'estat de Cilium

Podem veure que cilium està en estat correcte, utilitzant BPF, i en mode IPAM (assignació d'IPs dels pods).

Utilitzant la comanda de cilium-health podem revisar la connectivitat de cilium amb els altres nodes del clúster, i els temps de resposta.

```
❯ k exec -ti cilium-9z7j7 -n kube-system -- cilium-health status
Defaulted container "cilium-agent" out of: cilium-agent, config (init), mount-cgroup
es (init)
Probe time:   2024-04-16T12:28:27Z
Nodes:
  ip-172-16-13-249.ec2.internal (localhost):
    Host connectivity to 172.16.13.249:
      ICMP to stack:   OK, RTT=260.59µs
      HTTP to agent:   OK, RTT=192.662µs
    Endpoint connectivity to 100.64.0.230:
      ICMP to stack:   OK, RTT=242.921µs
      HTTP to agent:   OK, RTT=251.521µs
  ip-172-16-5-154.ec2.internal:
    Host connectivity to 172.16.5.154:
      ICMP to stack:   OK, RTT=984.538µs
      HTTP to agent:   OK, RTT=914.405µs
    Endpoint connectivity to 100.64.1.122:
      ICMP to stack:   OK, RTT=1.153819ms
      HTTP to agent:   OK, RTT=1.023137ms
  ip-172-16-9-240.ec2.internal:
    Host connectivity to 172.16.9.240:
      ICMP to stack:   OK, RTT=1.018769ms
      HTTP to agent:   OK, RTT=1.177239ms
    Endpoint connectivity to 100.64.2.138:
      ICMP to stack:   OK, RTT=1.374273ms
      HTTP to agent:   OK, RTT=1.208943ms
```

Figura 66: Comprovació de la connectivitat de cilium amb els nodes

Si ara fem la mateixa prova d'aixecar un pod de nginx, podem veure que cilium li ha assignat una IP interna controlada per ell mateix.

```
❯ k run nginx --image=nginx --restart=Never
pod/nginx created
❯ k get pod
NAME    READY   STATUS              RESTARTS   AGE
nginx   0/1     ContainerCreating   0          2s
❯ k get pod -owide
NAME    READY   STATUS    RESTARTS   AGE   IP            NODE                          NOMINATED NODE   READINESS GATES
nginx   1/1     Running   0          8s    100.64.1.254  ip-172-16-5-154.ec2.internal  <none>           <none>
```

Figura 67: Comprovació de l'assignació d'IP a un nou pod amb cilium

Si mirem el recurs de «CiliumNode» podem veure la IP interna del cada node, I la seva IP interna de Cilium.

```
❯ k get ciliumnode
NAME                          CILIUMINTERNALIP   INTERNALIP      AGE
ip-172-16-13-249.ec2.internal   100.64.0.183       172.16.13.249   6m27s
ip-172-16-5-154.ec2.internal    100.64.1.121       172.16.5.154    6m26s
ip-172-16-9-240.ec2.internal    100.64.2.72        172.16.9.240    6m26s
```

Figura 68: IPs internes i externes de Cilium

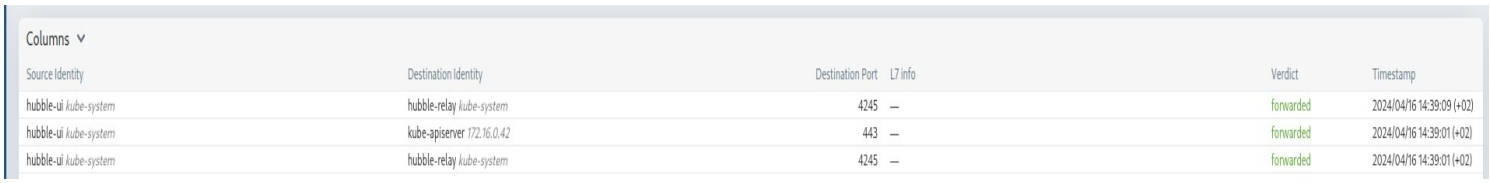
Com que estem utilitzant el mode «masquerade» de cilium, en cas que un pod decidís executar alguna cosa fora del clúster, aquest sortiria amb la IP del node, no del pod. [43]

Mitjançant un proxy invers podríem accedir al Hubble de Cilium directament. Ara per sortir del pas sense muntar-ne'n un, si volem accedir al hubble, podem fer un *port-forward* i accedir des del nostre navegador.

```
❯ kubectl port-forward -n kube-system deployment/hubble-ui 8081
Forwarding from 127.0.0.1:8081 -> 8081
Forwarding from [::1]:8081 -> 8081
```

Figura 69: Redireccionament de ports amb Kubectl per accedir a Hubble

Ara tenim molt poca cosa, però podem veure els paquets que s'envien els pods entre ells, o cap a fora, amb el permís de cilium.



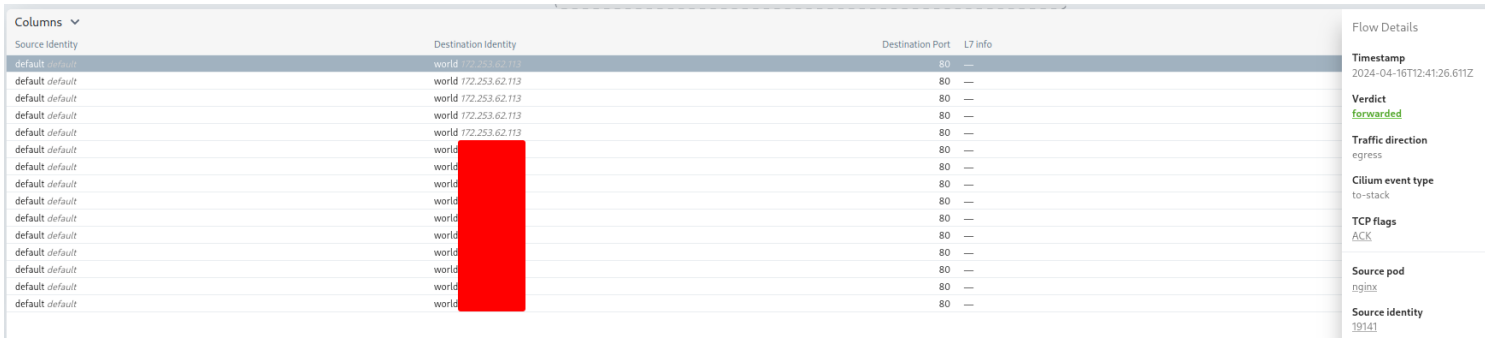
Per exemple, fent un curl des del pod de nginx a google

```
❯ k exec -ti nginx -- curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
E0416 14:41:26.689865  181126 v3.go:79] use of closed network connection
```

Figura 70: Prova de captura de paquets mitjançant curl

Podem capturar els paquets cilium ha reenviat.

En cas de tindre regles de seguretat configurades, podríem veure paquets bloquejats per el mateix cilium.



10.7: Instal·lació i configuració de Keda

Primer de tot afegirem el repositori de helm de keda dins el nostre helm.

```
❯ helm repo add kedacore https://kedacore.github.io/charts
"kedacore" has been added to your repositories
❯ helm repo ls | grep keda
kedacore        https://kedacore.github.io/charts
```

Figura 71: Addició del repositori de Keda a helm

A continuació, crearem un fitxer values.yaml per a modificar alguns paràmetres dels que venen per defecte.

```yaml
clusterName: test

crds:
  install: true

metricsServer:
  dnsPolicy: ClusterFirstWithHostNet
  useHostNetwork: true

podIdentity:
  aws:
    irsa:
      enabled: true
      roleArn: "arn:aws:iam::[redacted]:role/k8s-test-keda"

webhooks:
  useHostNetwork: true
```

Figura 72: Paràmetres modificats del helm de keda

Als paràmetres de dnsPolicy, i els de useHostNetwork se'ls ha configurat d'aquesta forma per tindre compatibilitat amb Cilium. [40]

Respecte l'IRSA, en aquest s'ha creat un rol d'AWS amb permisos de llegir els atributs de la cua SQS (per veure el número de missatges encuats) i, mitjançant OIDC, se li ha donat permís al service account de keda-operator, al namespace de kube-system per assumir aquest rol, seguint les indicacions de la documentació oficial. [41]

```
sceptre_user_data:
  Name: k8s-dev-keda
  PolicyDocument:
    - Action: "sts:AssumeRoleWithWebIdentity"
      Principal:
        Federated: "arn:aws:iam::[redacted]:oidc-provider/oidc.eks.us-east-1.amazonaws.com/id/[redacted]"
      Condition:
        StringEquals:
          "oidc.eks.us-east-1.amazonaws.com/id/[redacted]:aud": "sts.amazonaws.com"
          "oidc.eks.us-east-1.amazonaws.com/id/[redacted]:sub": "system:serviceaccount:kube-system:keda-operator"
  Policies:
    - Name: sqs-attributes
      Statements:
        - Action:
            - "sqs:GetQueueAttributes"
          Resource:
            - "arn:aws:sqs:us-east-1:[redacted]"
```

Figura 73: Codi del rol IAM que crearem per obtenir les dades de les cues

Finalment instal·larem Keda

```
❯ helm upgrade --namespace kube-system keda kedacore/keda --version 2.13.2 --install --wait --atomic -f values.yaml
Release "keda" has been upgraded. Happy Helming!
NAME: keda
LAST DEPLOYED: Fri Apr 26 21:05:42 2024
NAMESPACE: kube-system
STATUS: deployed
REVISION: 6
TEST SUITE: None
NOTES:
:::^.     .::::^:     :::::::::::::::    .::::::::::.                  .^.
????~   .^?????~.     ?????????????.   :!!!!!!!!!!^.              .???.
????~  ^?????~.       ~!!!!!!!!!!!!!.   :????!!!!??????~.           .?????.
????~^?????~.                           :????:    :~?????.         :???????.
??????????!.          :::::::::::.      :????:      .????!        :???????7.
??????????:           ???????????~      :????:       :????:     :?????5?????.
?????!~????^          !77777777777^     :????:       :????:     ^?????#P??????.
????~  ^????~                           :????:      :????!     ^????J#@J???????.
????~   :????!.                         :????:   .:~?????.    ~????Y&@#77777777.
????~    .?????:      !!!!!!!!!!!!!!    :?????!!7??????^     ~?????@@@GJJYJ??????.
????~     .!????^     ???????????????.   :!!!!!!!!!!7!~:      !????G@@@@@@@@5???????:
::::.       :::::     :::::::::::::::    .:::::::::..        .::::JGGGB@@@@&7::::::::
                                                                    ?@@#~
                                                                    P@B^
                                                                  :&G:
                                                                  !5.
                                                                  .Kubernetes Event-driven Autoscaling (KEDA) - Application autoscaling made simple.
```

Figura 74: Instal·lació de Keda mitjançant helm

Podem veure que els pods de Keda estan funcionant correctament

```
❯ k get pod | grep keda
keda-admission-webhooks-5dddd899cb-fxhhf                  1/1     Running     0
keda-operator-5c9798556d-554rw                            1/1     Running     0
keda-operator-metrics-apiserver-7d46fc64f5-kvqt6          1/1     Running     0
```

Figura 75: Validació que els pods de Keda estan executant-se correctament

Amb el Keda ja funcionant, podrem crear els nous elements que escalaran el *Deployment* basat en events.

Figura 76: Codi dels objectes a escalar

Seguint la documentació [42], hem configurat el deployment (spec.scaleTargetRef), el nombre mínim i màxim de rèpliques, intervals de actualització de mètriques de la cua, URL de la cua, i nombre de pods per cada missatge que la cua tingui. Una vegada creat l'objecte podrem veure'l de la següent manera:

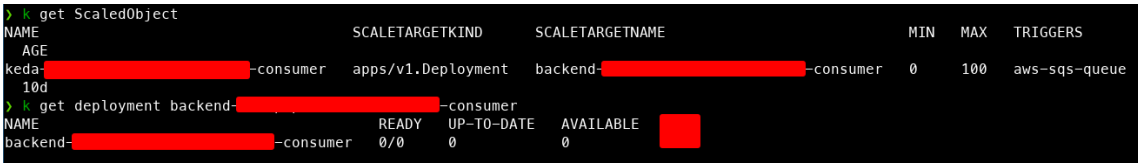


Figura 77: Objectes a escalar creats, amb els deployment connectat

Podem veure que el Deployment, com que no hi ha res a la cua, és a 0 pods.

10.8: Instal·lació i configuració de Karpenter

La instal·lació i configuració de Karpenter és una mica més complexa que altres paquets, sobretot en clústers de Kubernetes que ja estan aixecats. Per aquesta raó utilitzarem la guia oficial de migrar de clúster autoscaler, ja que assumeix que ja tenim un clúster actualment. [45]

Començarem creant el rol que els nodes utilitzaran

```
sceptre_user_data:
  Name: k8s-test-karpenter-nodegroup
  PolicyDocument:
    - Action: "sts:AssumeRole"
      Principal:
        Service: "ec2.amazonaws.com"
  ManagedPolicies:
    - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
    - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
    - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
    - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

Figura 78: Codi del rol IAM de Karpenter amb les polítiques requerides

Aquest rol utilitzarà les policies oficials d'AWS de node d'eks, cni, ssm i lectura de repositoris d'ECR.

I crearem també el rol que utilitzarà el mateix pod controlador de Karpenter mitjançant OIDC.

```
sceptre_user_data:
  Name: k8s-test-karpenter-controller
  PolicyDocument:
    - Action: "sts:AssumeRoleWithWebIdentity"
      Principal:
        Federated: "arn:aws:iam::XXXXX:oidc-provider/oidc.eks.us-east-1.amazonaws.com/id/XXXXX"
      Condition:
        StringEquals:
          "oidc.eks.us-east-1.amazonaws.com/id/XXXXX:aud": "sts.amazonaws.com"
          "oidc.eks.us-east-1.amazonaws.com/id/XXXXX:sub": "system:serviceaccount:kube-system:karpenter"
  Policies:
    - Name: karpenter
      Statements:
        - Action:
          - "ssm:GetParameter"
          - "ec2:DescribeImages"
          - "ec2:RunInstances"
          - "ec2:DescribeSubnets"
          - "ec2:DescribeSecurityGroups"
          - "ec2:DescribeLaunchTemplates"
          - "ec2:DescribeInstances"
          - "ec2:DescribeInstanceTypes"
          - "ec2:DescribeInstanceTypeOfferings"
          - "ec2:DescribeAvailabilityZones"
          - "ec2:DeleteLaunchTemplate"
```

```
      - "ec2:CreateTags"
      - "ec2:CreateLaunchTemplate"
      - "ec2:CreateFleet"
      - "ec2:DescribeSpotPriceHistory"
      - "pricing:GetProducts"
      Resource:
        - "*"
      Effect: Allow
  - Name: ConditionalEC2Termination
    Statements:
      - Action:
      - "ec2:TerminateInstances"
      Resource:
        - "*"
      Effect: Allow
      Condition:
        StringLike:
          "ec2:ResourceTag/karpenter.sh/nodepool": "*"
  - Name: PassNodeIAMRole
    Statements:
      - Action:
      - "iam:PassRole"
      Effect: Allow
      Resource:
        - "arn:aws:iam::XXXXX:role/k8s-test-karpenter-nodegroup"
  - Name: EKSClusterEndpointLookup
    Statements:
      - Action:
      - eks:DescribeCluster
      Effect: Allow
      Resource:
        - "arn:aws:eks:us-east-1:XXXXX:cluster/test"
  - Name: AllowScopedInstanceProfileCreationActions
    Statements:
      - Action:
      - iam:CreateInstanceProfile
      Resource:
        - "*"
      Effect: Allow
      Condition:
        StringEquals:
          "aws:RequestTag/kubernetes.io/cluster/test": "owned"
          "aws:RequestTag/topology.kubernetes.io/region": "us-east-1"
        StringLike:
          "aws:RequestTag/karpenter.k8s.aws/ec2nodeclass": "*"
  - Name: AllowScopedInstanceProfileTagActions
    Statements:
      - Action:
      - iam:TagInstanceProfile
      Effect: Allow
      Resource:
        - "*"
      Condition:
        StringEquals:
          "aws:ResourceTag/kubernetes.io/cluster/test": "owned"
          "aws:ResourceTag/topology.kubernetes.io/region": "us-east-1"
          "aws:RequestTag/kubernetes.io/cluster/test": "owned"
          "aws:RequestTag/topology.kubernetes.io/region": "us-east-1"
        StringLike:
          "aws:ResourceTag/karpenter.k8s.aws/ec2nodeclass": "*"
          "aws:RequestTag/karpenter.k8s.aws/ec2nodeclass": "*"
  - Name: AllowScopedInstanceProfileActions
    Statements:
      - Action:
      - "iam:AddRoleToInstanceProfile"
      - "iam:RemoveRoleFromInstanceProfile"
      - "iam:DeleteInstanceProfile"
      Effect: Allow
      Resource:
        - "*"
      Condition:
        StringEquals:
```

```
          "aws:ResourceTag/kubernetes.io/cluster/test": "owned"
          "aws:ResourceTag/topology.kubernetes.io/region": "us-east-1"
        StringLike:
          "aws:ResourceTag/karpenter.k8s.aws/ec2nodeclass": "*"
- Name: AllowInstanceProfileReadActions
  Statements:
    - Action:
      - "iam:GetInstanceProfile"
      Effect: Allow
      Resource:
      - "*"
```

Aquest rol és el que permetrà al controlador de Karpenter dur a terme les principals accions:

- Obtindre informació del clúster EKS, i les instàncies que hi ha al compte, amb el seu preu.
- Crear perfils d'instàncies
- Aixecar i eliminar instàncies dins del clúster, de Karpenter
- Afegir rols i tags a les instàncies que ha aixecat

Una vegada tenim els rols creats, el que farem serà afegir tags per a indicar-li a Karpenter quins recursos utilitzar. Començarem afegint tags a les subnets on Karpenter afegirà les noves instàncies:

```
EksA:
  az: us-east-1a
  nat: true
  private:
    cidr: 172.16.4.0/22
    extraTags:
      kubernetes.io/role/internal-elb: 1
      karpenter.sh/discovery: test
EksB:
  az: us-east-1b
  nat: true
  private:
    cidr: 172.16.8.0/22
    extraTags:
      kubernetes.io/role/internal-elb: 1
      karpenter.sh/discovery: test
EksC:
  az: us-east-1c
  nat: true
  private:
    cidr: 172.16.12.0/22
    extraTags:
      kubernetes.io/role/internal-elb: 1
      karpenter.sh/discovery: test
```

Figura 79: Addició dels tags d'autodescobriment de les subnets de Karpenter

Editarem el configmap d'aws-auth per a afegir el nou rol que hem creat per als nodes

kubectl edit configmap aws-auth -n kube-system

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::[redacted]:role/eksctl-test-nodegroup
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::[redacted]:role/k8s-test-karpenter-nodegroup
      username: system:node:{{EC2PrivateDNSName}}
kind: ConfigMap
metadata:
  creationTimestamp: "2024-04-14T17:56:06Z"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "4991387"
  uid: b03767c8-c3db-485e-b14a-dbdbce6ab690
```

Figura 80: Actualització del configmap d'aws-auth amb el nou rol

Una vegada realitzades aquestes modificacions, instal·larem Karpenter mitjançant Helm. Crearem el fitxer de valors amb les següents dades:

```
controller:
  image:
    tag: 0.36.1
  resources:
    requests:
      cpu: 1
      memory: 1Gi
    limits:
      cpu: 1
      memory: 1Gi
settings:
  clusterName: test
serviceAccount:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::[redacted]:role/k8s-test-karpenter-controller
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: karpenter.sh/nodepool
              operator: DoesNotExist
        - matchExpressions:
          - key: eks.amazonaws.com/nodegroup
            operator: In
            values:
            - a-ondemand-129-001
            - b-ondemand-129-001
            - c-ondemand-129-001
```

Figura 81: Fitxer values de Karpenter amb els paràmetres modificats

Li indicarem el nom del clúster, el rol per al controlador de Karpenter que hem creat abans, i una afinitat per a que el controlador de Karpenter sempre estigui en un node no gestionat per ell mateix.

Finalment, crearem la pool de nodes, i la classe que utilitzarà el Karpenter.

```yaml
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  template:
    spec:
      requirements:
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
        - key: kubernetes.io/os
          operator: In
          values: ["linux"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["m"]
        - key: karpenter.k8s.aws/instance-family
          operator: In
          values: ["m5","m5a","m5ad","m5d","m5dn","m5n","m6i"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["4"]
        - key: "karpenter.k8s.aws/instance-cpu"
          operator: In
          values: ["4"]
      nodeClassRef:
        apiVersion: karpenter.k8s.aws/v1beta1
        kind: EC2NodeClass
        name: default
  limits:
    cpu: 1000
  disruption:
    consolidationPolicy: WhenUnderutilized
    #expireAfter: 720h # 30 * 24h = 720h
---
apiVersion: karpenter.k8s.aws/v1beta1
kind: EC2NodeClass
metadata:
  name: default
spec:
  amiFamily: AL2
  role: "k8s-test-karpenter-nodegroup"
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "test"
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "test"
  amiSelectorTerms:
    - id: "ami-057f49c54d950e56c"
```

Figura 82: Objectes NodePool i EC2NodeClass amb les dades actualitzades

En el NodeClass indicarem la AMI, el rol dels nodes, i amb els tags que hem creat abans, descobrirà els subnets i els SG on afegirà els nous nodes.

En el NodePool indicarem quin tipus de node voldrem. En el nostre cas arquitectura amd64, Linux, de la família m, generació mínima 4... Aqui podrem restringir els tipus de nodes que volem utilitzar. En el meu cas he utilitzat el mateix que he utilitzat prèviament. També podrem configurar dades com un temps màxim per a cada node per a que es vagin rotant periòdicament.

10.9: Instal·lació i configuració de external-secrets
[44]

10.9.1. Helm

Primer de tot afegirem el repositori d'external-secrets al helm.

```
❯ helm repo add external-secrets https://charts.external-secrets.io
"external-secrets" has been added to your repositories
```

Figura 83: Addició del repositori helm d'external-secrets

A continuació, crearem un fitxer values.yaml per a modificar alguns paràmetres dels que venen per defecte.

```yaml
installCRDs: true

webhook:
  # Specifies if the webhook should be started in hostNetwork mode.
  #
  # Required for use in some managed kubernetes clusters (such as AWS EKS) with custom
  # CNI (such as calico), because control-plane managed by AWS cannot communicate
  # with pods' IP CIDR and admission webhooks are not working
  #
  # Since the default port for the webhook conflicts with kubelet on the host
  # network, `webhook.securePort` should be changed to an available port if
  # running in hostNetwork mode.
  hostNetwork: true
  port: 10251
```

Figura 84: Fitxer values amb els paràmetres actualitzats d'external-secrets

En el fitxer values li indicarem que instal·li ell mateix els CRDs, i el webhook s'executi en mode hostNetwork per poder funcionar amb Cilium.

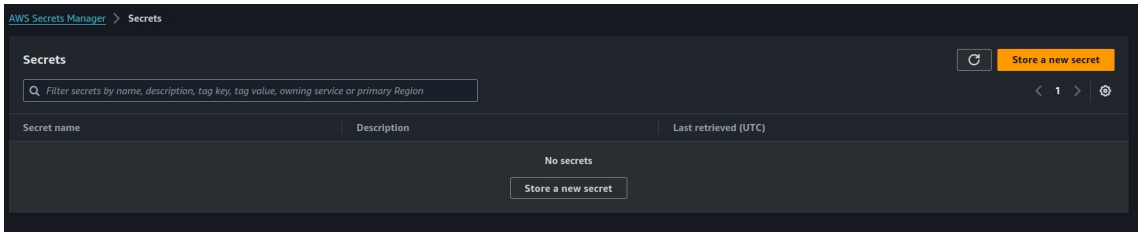
Finalment instal·larem el chart.

```
❯ helm install external-secrets external-secrets/external-secrets --version "v0.9.16" -f values.yaml
NAME: external-secrets
LAST DEPLOYED: Sun Apr 28 21:47:54 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
external-secrets has been deployed successfully in namespace default!

In order to begin using ExternalSecrets, you will need to set up a SecretStore
or ClusterSecretStore resource (for example, by creating a 'vault' SecretStore).

More information on the different types of SecretStores and how to configure them
can be found in our Github: https://github.com/external-secrets/external-secrets
```

Figura 85: Instal·lació d'external-secrets mitjançant helm

10.9.2. Creació del secret i rols necessaris

Primer de tot crearem un secret *dummy* a AWS



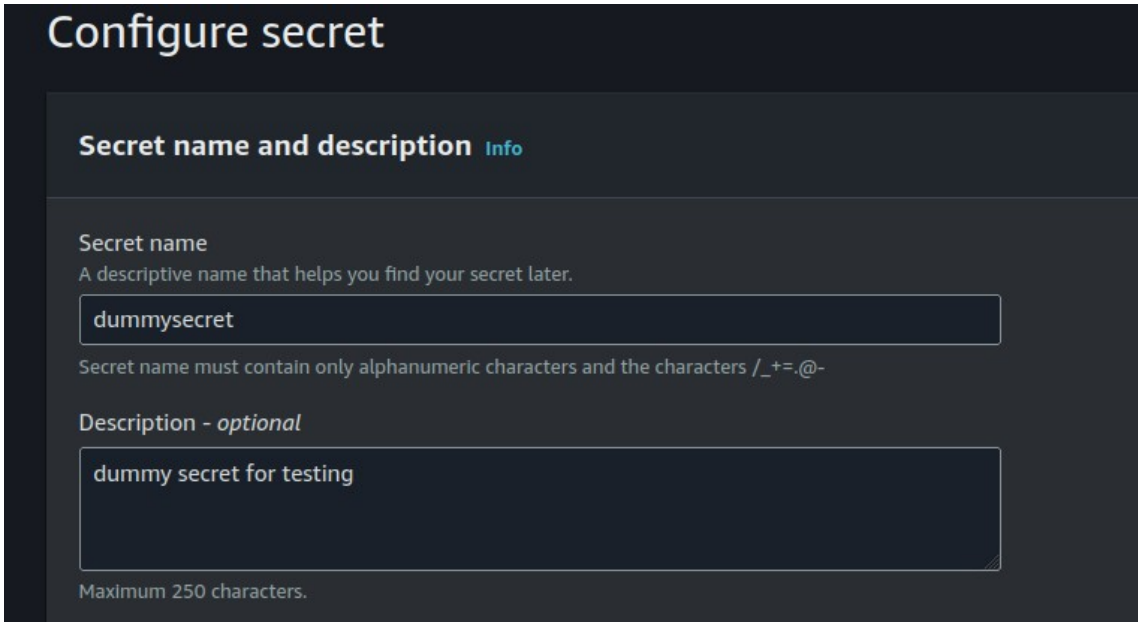


Figura 86: Creació d'un secret amb la consola d'AWS

A continuació crearem un rol que, mitjançant OIDC permetrà al mòdul d'external-secrets accedir al secret, i sincronitzar-lo amb el clúster.

```
sceptre_user_data:
  Name: k8s-lab-secret
  PolicyDocument:
    - Action: "sts:AssumeRoleWithWebIdentity"
      Principal:
        Federated: "arn:aws:iam::[redacted]:oidc-provider/oidc.eks.us-east-1.amazonaws.com/id/[redacted]"
      Condition:
        StringEquals:
          "oidc.eks.us-east-1.amazonaws.com/id/[redacted]:aud": "sts.amazonaws.com"
          "oidc.eks.us-east-1.amazonaws.com/id/[redacted]:sub":
            - "system:serviceaccount:default:test-secret"
  Policies:
    - Name: secret-reader
      Statements:
        - Action:
            - "secretsmanager:GetResourcePolicy"
            - "secretsmanager:GetSecretValue"
            - "secretsmanager:DescribeSecret"
            - "secretsmanager:ListSecretVersionIds"
          Resource:
            - "arn:aws:secretsmanager:us-east-1:[redacted]:secret:dummysecret/*"
```

Figura 87: Codi del rol que utilitzarà external-secrets per a obtindre el secret

10.9.3. Sincronització del secret amb el clúster

Per a sincronitzar el secret amb el clúster de Kubernetes, crearem un service-account, un secretstore i un externalsecret:

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::[redacted]:role/k8s-lab-secret
  name: test-secret
  namespace: default
---
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: test-secret
  namespace: default
spec:
  provider:
    aws:
      service: SecretsManager
      region: us-east-1
      auth:
        jwt:
          serviceAccountRef:
            name: test-secret
---
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: test-secret
  namespace: default
spec:
  refreshInterval: 24h
  secretStoreRef:
    name: test-secret
    kind: SecretStore
  target:
    name: test-secret
    creationPolicy: Owner
  data:
  - secretKey: dummysecret
    remoteRef:
      key: dummysecret
```

Figura 88: Objectes de ServiceAccount, SecretStore i ExternalSecret amb els rols i secrets a utilitzar

El serviceaccount és el mateix que al OIDC del rol hem afegit que pugui assumir el rol que tindrà accés al contingut del secret. El secretstore tindrà les dades de quin servei utilitzarà (SecretsManager d'aws). El ExternalSecret tindrà les dades de quin secret vol obtindre. El funcionament seria aquest:

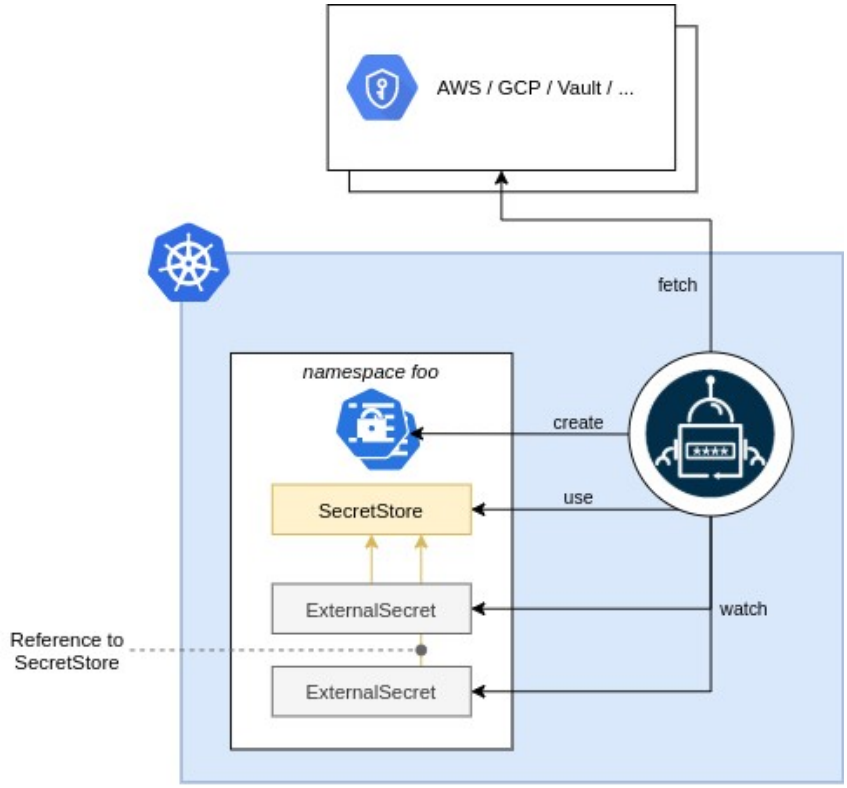


Figura 89: Funcionament d'external-secrets

Crearem els recursos:

```
❯ k create -f test.yaml
serviceaccount/test-secret created
secretstore.external-secrets.io/test-secret created
externalsecret.external-secrets.io/test-secret created
```

I podem observar que s'han creat i estan funcionant correctament:

```
❯ k get SecretStore
NAME          AGE   STATUS   CAPABILITIES   READY
test-secret   42s   Valid    ReadWrite      True
❯ k get ExternalSecret
NAME          STORE         REFRESH INTERVAL   STATUS         READY
test-secret   test-secret   24h                SecretSynced   True
```

Figura 90: Validació que el secret s'està sincronitzant

De fet, podem obtindre el valor del secret que hem guardat, obtenint les dades del secret, i descodificant el secret que es guarda en format base64.

```
❯ k get secret
NAME                                  TYPE                 DATA   AGE
external-secrets-webhook              Opaque               4      34m
sh.helm.release.v1.external-secrets.v1   helm.sh/release.v1   1      34m
test-secret                           Opaque               1      65s
❯ k get secret test-secret -oyaml
apiVersion: v1
data:
  dummysecret: aWFtYWR1bW15c2VjcmV0
immutable: false
kind: Secret
metadata:
  annotations:
    reconcile.external-secrets.io/data-hash: f0353948739ebeecb584b34b5e960da0
  creationTimestamp: "2024-04-28T20:21:43Z"
  labels:
    reconcile.external-secrets.io/created-by: b7cd1ff8ebb944021154c194f5043a44
  name: test-secret
  namespace: default
  ownerReferences:
  - apiVersion: external-secrets.io/v1beta1
    blockOwnerDeletion: true
    controller: true
    kind: ExternalSecret
    name: test-secret
    uid: c3e5f67c-a5cd-45b7-ba27-ff841dad1c00
  resourceVersion: "3687064"
  uid: 408c0344-0bda-4052-9a7b-a823d8d6bc7a
type: Opaque
❯ echo "aWFtYWR1bW15c2VjcmV0" | base64 -d
iamadummysecret%
```

Figura 91: Validació que podem obtindre el valor del secret