UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicació de microserveis a una infraestructura de carreteres intel·ligents.

Treball Fi de Màster

Màster Universitari en Enginyeria Informàtica

Autor: Axel Guzman Godia

Tutor: Joan Fons i Cors
Vicente Pelechano Ferragud

Curs 2017-2018

Resum

Avui en dia, el desenvolupament de software ha incrementat les seues capacitats aprofitant el potencial d'Internet. Els sistemes desenvolupats poden ser utilitzats potencialment a qualsevol lloc, per qualsevol persona. Això ha introduït nous requeriments per al programari. Entre aquests, podem trobar escalabilitat, ubiqüitat, disponibilitat, etc. Paral·lelament al sorgiment d'aquestes noves necessitats, han aparegut nous models arquitectònics per al desenvolupament de programari. L'objectiu de l'arquitectura de microserveis es solucionar aquests reptes, centrant el seu disseny en les necessitats actuals del programari. Aquest projecte aborda la transformació d'una aplicació prèviament desenvolupada cap a l'arquitectura de microserveis.

Paraules clau: Internet de les Coses, Ciutats intel·ligents Microserveis, Cloud Computing

Resumen

Actualmente, el desarrollo de software ha incrementado sus capacidades explotando el potencial de Internet. Los sistemas desarrollados pueden ser usados potencialmente en cualquier lugar, por cualquier persona. Esto ha introducido nuevos requisitos para el software. Entre estas necesidades, encontramos la escalabilidad, ubiquidad, disponibilidad, etc. Paralelamente al surgimiento de estas nuevas necesidades, han aprecido nuevos modelos arquitectónicos para el desarrollo de software. El objetivo de la arquitectura de microservicios es solucionar estos nuevos retos, centrando su diseño en las necesidades actuales del software. Este proyecto aborda la transformación de una aplicación ya desarrollada hacia la arquitectura de microservicios.

Palabras clave: Internet de las Cosas, Ciudades Inteligentes, Microservicios, Cloud Computing

Abstract

Nowadays software development has increased its capabilities by exploiting the potential of the Internet. The developed systems can potentially be used anywhere, by anyone. This has introduced several new requirements for the software. Among these necessities are escalability, ubiquity, availability, etc. Paralelly at the rising of these new necessities, new architectural models for software development have emerged. The microservice architectur patern aims to solve the new challenges by focusing its design on the needs of today software. This project performs

the transformation of a yet-developed application to the microservice architecture pattern.

Key words: Internet of things, SmartCities, Microservices, Cloud Computing

Índex

Índex de figures

Capítol 1

Introducció

1.1 Motivació

El desenvolupament d'aplicacions ha canviat de manera decisiva amb la irrupció imparable d'Internet. La possibilitat de produir aplicacions que puguin interactuar a través d'Internet comporta no només infinites possibilitats, sinó també nous reptes i problemes.

Els avantatges són molt grans, ja que les aplicacions poden arribar a un nombre enorme d'usuaris finals, proporcionen una gran disponibilitat, permeten desplaçar el processament del client al servidor o al contrari,etc...

En canvi, també apareixen una serie d'inconvenients que poden fer que els patrons de desenvolupament de software tradicional no siguin suficientment eficaços.

En aquest context, l'arquitectura de microserveis ofereix una serie d'avantatges que permeten adaptar-se amb èxit a l'entorn actual.

Aquest projecte aborda la transformació d'una aplicació a l'arquitectura de microserveis per tal d'aconseguir adaptar-se amb èxit al context actual.

1.2 Problemàtica

1.2.1 Aplicacions orientades a la xarxa

El desplegament d'aplicacions a la xarxa comporta grans avantatges. Malgrat això, també introdueix una serie de problemes que s'han de resoldre. Entre aquests, podem destacar els següents:

- **Nombre d'usuaris.** El nombre de usuaris connectats a una aplicació a la xarxa pot anar des d' uns quants usuaris fins a milions d'usuaris. Això fa necessari adoptar alguna estratègia per a poder donar suport a una quantitat tan gran de clients.

  Aquesta característica fa que l'escalabilitat de les aplicacions sigui un factor determinant en el seu èxit. Les aplicacions han de dissenyar-se per tal de permetre escalar-se a mateix temps que augmenta el nombre d'usuaris, mentre mantenen la seua funcionalitat.

  A més a més de que el nombre d'usuaris pot ser molt gran, aquests usuaris es connecten de forma arbitrària. Així, poden produir-se grans diferències entre el nombre d'usuaris en dos instants de temps diferents.

- **Disponibilitat.** Una de les característiques que els usuaris esperen de les aplicacions en xarxa es disponibilitat total. Les aplicacions deuen ser tolerants a fallades i oferir un servei ininterromput.

- **Adreçament dinàmic.** Una de les característiques actuals d'Internet es el adreçament dinàmic. Aquesta característica fa que un node pugui tindre diferents adreces al llarg de la seua vida. Això té una importància molt gran des de el punt de vista dels servidors, que han de tindre una adreça coneguda per els usuaris.

- **Gran diversitat de sistemes.** Un dels grans reptes de distribuir aplicacions per la xarxa es la diversitat de sistemes en la mateixa. Així, podem trobar connectats a la xarxa multitud de sistemes operatius, multitud d'arquitectures hardware, sistemes amb diferents capacitats, etc.

  Això fa necessari utilitzar mecanismes de comunicació estàndard, per tal de permetre que tots els elements puguin comunicar-se sense importar les seues particularitats.

- **Presència d'usuaris maliciosos.** El desplegament d'una aplicació a la xarxa comporta que la aplicació està exposada no només als usuaris normals, sinó també a qualsevol usuari maliciós, que pot intentar obtindre informació o alterar el funcionament normal de l'aplicació.

1.2.2 Aplicacions monolítiques

Les aplicacions monolítiques son aquelles aplicacions que estan formades per una única peça de codi que conté tots els elements necessaris per a la seva execució.

Aquest tipus d'aplicacions han sigut la forma tradicional de desenvolupament durant anys. No obstant, aquest model té una serie de limitacions:

- **Problemes d'escalabilitat.** Una de les característiques de les aplicacions monolítiques es que contenen tots els elements necessaris per a la seua execució. Quan es vol escalar una aplicació d'aquest tipus, el que es fa es llançar una altra instància de l'aplicació. Això comporta que tots els elements de l'aplicació estan replicats, encara que potser només una part de la aplicació té necessitats d'escalabilitat. Per exemple, imaginem que tenim una aplicació que ofereix un servei per a convertir imatges a diversos formats i un servei per a comprimir fitxers. En un determinat moment, l'aplicació rep moltes peticions per a convertir imatges i es decideix posar en execució una rèplica per atendre la demanda. Encara que només es necessita més capacitat per a convertir imatges, hem replicat també tota la part encarregada de comprimir fitxers.

- **Problemes de mantenibilitat.** Una de les característiques de les aplicacions monolítiques es que tendeixen a créixer molt. Així a mesura que la aplicació va madurant i es van afegint noves funcionalitats, el seu codi comença a créixer i créixer. Això acaba comportant problemes de mantenibilitat, ja que resulta complexe mantenir un codi tan gran, sobretot per a persones que no coneixen totalment tot el codi.

- **Falta d'agilitat.** directament relacionat amb el punt anterior, el fet de treballar amb un codi molt gran i complexe afecta al desenvolupament de l'aplicació. En un monòlit complexe, resulta difícil afegir nous components sense tindre un bon coneixement del funcionament de tot el codi. Això resta agilitat i dificulta el treball a les persones que no coneixen amb detall el codi.

- **Acoblament.** Un altre dels problemes es el acoblament. Com l'aplicació està continguda en una unitat, la fallada d'algun dels seus components pot comportar una fallada per a tota l'aplicació. Així si hi ha un error per exemple en la capa de persistència, es perd la funcionalitat de tota l'aplicació.

- **Compromís tecnològic.** Un altre aspecte important, es que les aplicacions monolítiques estableixen dependències fortes amb la tecnologia. Per exemple, la decisió inicial sobre quin llenguatge de programació utilitzar, s'arrastra al llarg de tot el cicle de vida de l'aplicació. Aixo pot comportar afegir complexitats quan existeixen altres alternatives que permeten fer les mateixes coses de forma més senzilla.

1.3 Arquitectura de microserveis

Malgrat que l'objectiu d'aquest treball no es explicar l'arquitectura de microserveis en detall, resulta imprescindible conèixer els seus fonaments. A continuació, s'expliquen les característiques clau de l'arquitectura de microserveis.

L'arquitectura de microserveis es un patró que es basa en desenvolupar aplicacions com un conjunt de serveis independents. Aquests serveis independents implementen parts de la funcionalitat i es comuniquen entre sí mitjançant mecanismes de comunicació (habitualment APIs REST, encara que existeixen més mecanismes).

Els avantatges que proporciona aquest patró de desenvolupament són els següents:

- **Components independents.** Amb aquest patró de disseny, cada un dels microserveis esdevé un component independent. Això permet que el desenvolupament de cada component es limiti a una funcionalitat concreta. Generalment, cada un dels microserveis serà un component relativament simple, la qual cosa permet facilitar la comprensió en el funcionament per als desenvolupadors.

  Aquest aspecte permet també què les millores implementades és puguin incorporar directament a l'entorn de producció independentment del desenvolupament d'altres parts, i afectant només a un component determinat.

- **Poliglot.** El fet de que tots els components siguin independents elimina la necessitat de fer tot el desenvolupament amb un únic llenguatge de programació. Com els components son independents i es comuniquen entre sí amb mecanismes estàndard, cada component pot ser desenvolupat amb el llenguatge que es consideri més apropiat.

- **Aïllament de fallades.** L'estratègia de desenvolupar components independents permet limitar les fallades als components. Així, quan un component falla, aquest error no s'estén a tota l'aplicació, sinó que es conté en el propi component.

  A més a més, com els components són reduïts en comparació a una aplicació monolítica tradicional, resulta més senzill trobar errors al codi.

- **Escalabilitat.** Una altra de les característiques fonamentals que proporciona aquesta arquitectura és que dona suport a l'escalabilitat. Com les aplicacions estan formades per un conjunt de components, es possible augmentar el nombre d'instàncies d'un component concret si es necessari o reduir-les, sense haver de replicar tota la aplicació (què es l'aproximació tradicional).

Per altra banda, malgrat els avantatges que aporta aquesta arquitectura, també existeixen una serie de desavantatges:

- **Sobrecàrrega.** Per la naturalesa de les aplicacions basades en microserveis, es requereix un ús intensiu de comunicacions. Aquestes comunicacions entre components resulten prou més costoses que les comunicacions dins d'un procés en una aplicació monolítica tradicional.

- **Desplegament complexe.** Posar en marxa una aplicació basada en microserveis requereix encarregar-se de molts aspectes. S'han de definir estratègies per a aprovisionar els microserveis, coordinar-los, monitoritzar l'estat de cadascun etc...

- **Coherència.** Mantindre la coherència en un sistema distribuït és un problema rellevant que no existeix en una aplicació monolítica. Així, cal definir mecanismes per a assegurar-se de que els diversos components tenen una visió coherent de la informació gestionada per l'aplicació.

- **Validació complexa.** Finalment, validar una aplicació basada en microserveis pot tindre certa dificultat. Si bé validar els components en sí pot resultar més senzill, a l'hora de validar el conjunt s'han de tindre en compte totes les interaccions de cadascun dels components i el possible impacte en el comportament de l'aplicació.

Com a conclusió, es desprèn que l'arquitectura de microserveis ofereix una serie d'avantatges que permeten agilitzar el desplegament d'aplicacions, la millora continua i permeten assolir major escalabilitat. Per contra, aquests avantatges venen amb una serie de complexitats afegides que requereixen una base tècnica més gran per a poder ser superades.

Cal destacar que no és recomanable aplicar aquesta arquitectura si no està justificat, ja que les complexitats i sobrecarregues que ofereix poden ser donar pitjors resultats que aplicar altres patrons de disseny.

1.4 Metodologia

La metodologia que s'ha seguit per a desenvolupar aquest projecte ha sigut el desenvolupament de prototips en cascada. Així, en cada etapa s'han anat desenvolupat dissenys i prototips, per a anar validant tot el procés.

Finalment, en aquesta memòria es recull el resultat del últim desenvolupament complet.

1.4.1 Planificació

Les metodologia que s'ha seguit en aquest projecte és el desenvolupament de prototips en cascada.

A continuació s'adjunta el diagrama de Gant amb la planificació del projecte:

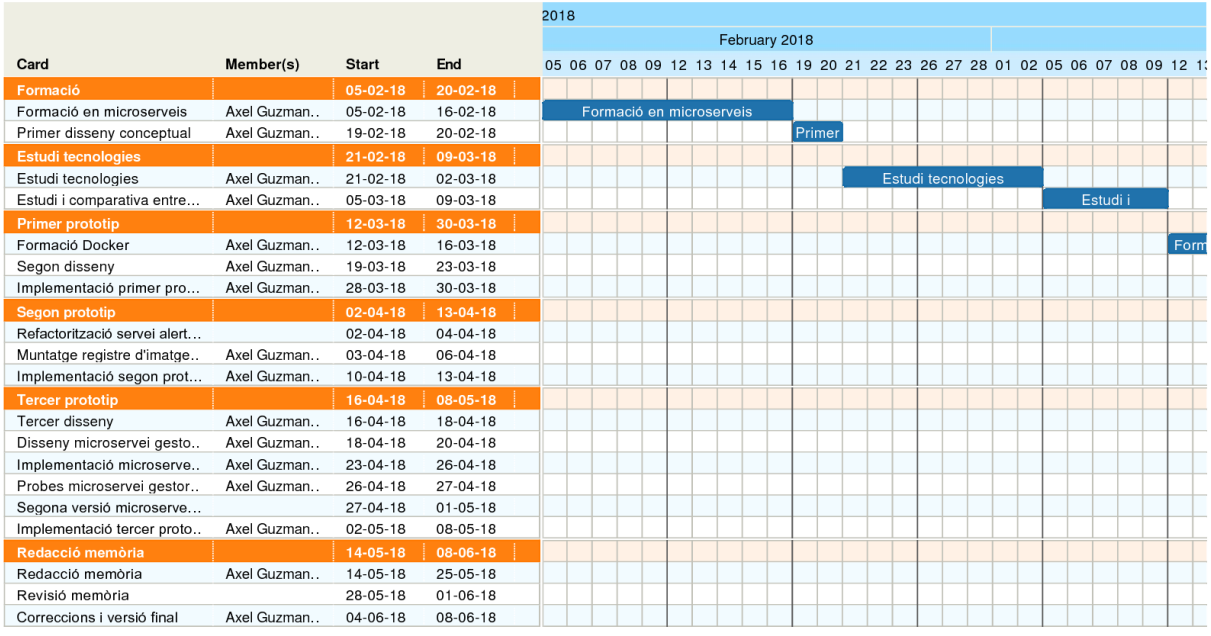

Figura 1.1: Diagrama de Gant del projecte.

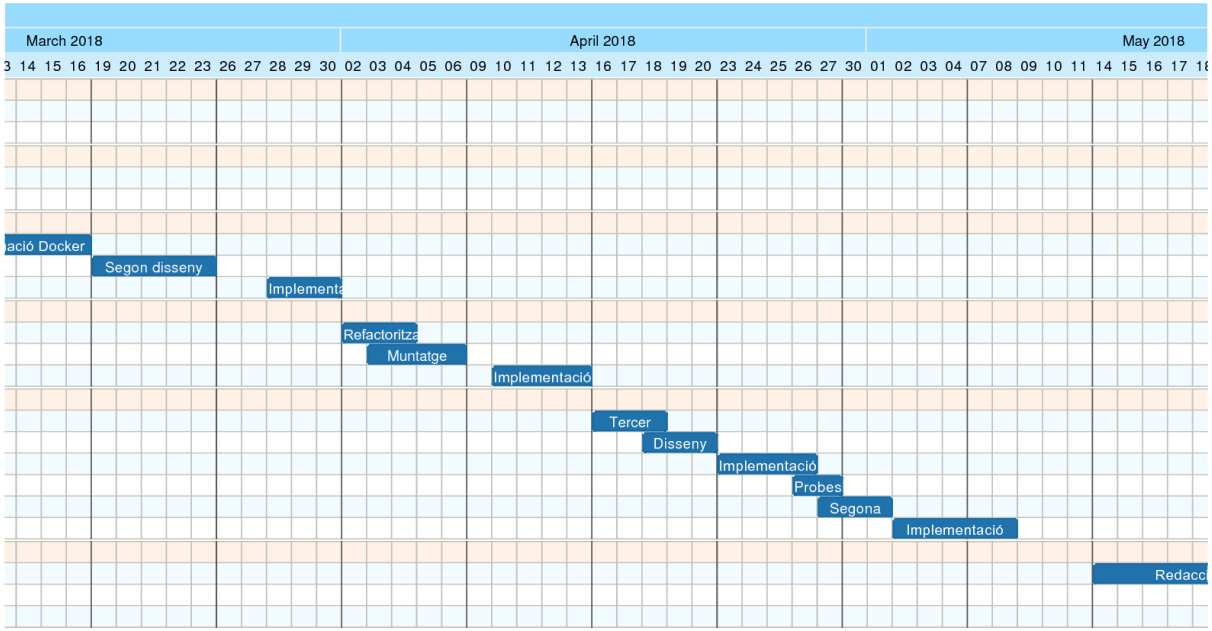

Figura 1.2: Diagrama de Gant del projecte.

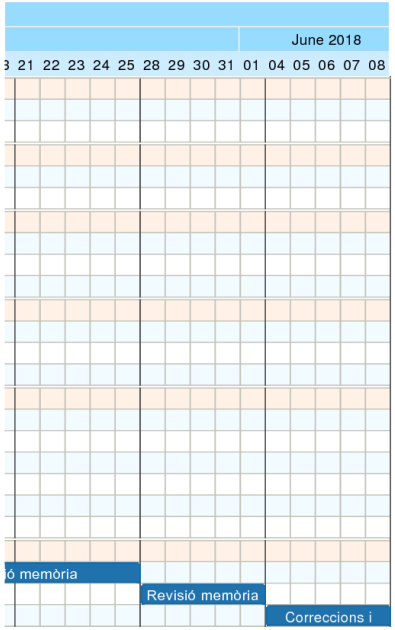

Figura 1.3: Diagrama de Gant del projecte.

1.4.2 Ferramentes utilitzades

Per a desenvolupar aquest projecte s'han utilitzat les següents ferramentes:

- **Git**. Programari de control de versions lliure i gratuït. Com en tot projecte informàtic, resulta imprescindible fer ús d'un programari de control de versions.
- **Node.js**. Entorn d'execució per a JavaScript. Es lleuger, te un model d'execució asíncron i és eficient. Compta amb funcionalitats natives que donen suport als programes orientats a la xarxa.
- **Docker**. Projecte de codi obert per a automatitzar el desplegament de programari en contenidors. Es la peça clau del plantejament del projecte i el seu disseny.
- **EVIR**. Servei de màquines virtuals del departament DSIC de la Universitat Politècnica de València. Proporciona màquines virtuals per a fer probes.

1.5 Objectius

L'objectiu d'aquest projecte es fonamenta en dos punts:

- **Dissenyar la transformació cap a una arquitectura de microserveis**. Això significa decidir quines tecnologies utilitzar, estudiar les necessitats específiques de l'aplicació per adaptar-la i determinar els desenvolupaments necessaris. També inclou el disseny de la infraestructura que s'encarregarà d'executar l'aplicació basada en microserveis.
- **Implementar la transformació en base al disseny**. Consisteix bàsicament, en posar en pràctica el disseny anterior. Com a resultat, es completa el procés de transformació d'una aplicació monolítica a una aplicació basada en microserveis.

1.6 Estructura del document

Aquest document s'organitza en sis capítols. La estructura que s'ha seguit s'explica a continuació.

El primer capítol es la introducció, on es proporciona informació sobre el context del projecte, la problemàtica, els objectius , la metodologia i la estructura del document.

A continuació, al segon capítol es fa un estudi tecnològic per a determinar quines son les característiques de la tecnologia utilitzada i quines tecnologies poden aportar valor al desenvolupament del projecte.

Seguidament, al tercer capítol es descriu la aplicació que es vol transformar a l'arquitectura de microserveis i les característiques més importants.

Amb tota aquesta informació, al capítol quatre s'exposa el disseny proposat per a abordar la transformació de l'aplicació inicial en una aplicació basada en microserveis.

Com a últim pas del desenvolupament, el capítol cinc aborda la implementació d'un prototip basat en el disseny del capítol anterior.

Finalment, el capítol sis recull les conclusions del projecte. També es proposen en aquest capítol ampliacions futures per a continuar amb el procés de transformació.

Capítol 2

Estudi de tecnologies

L'objectiu d'aquest capítol es analitzar les tecnologies que existeixen al mercat per a donar suport a la arquitectura de microserveis. La tendència actual en la industria que està enfocant-se cap a les arquitectures de microserveis es fer un ús intensiu de la tecnologia de contenidors.

Amb l'ús de contenidors, també sorgeix la necessitat de poder gestionar-los de forma eficient. Les tecnologies d'orquestració de contenidors s'encarreguen de donar suport al desplegament de contenidors en diferents màquines.

2.1 Contenidors

Un dels grans problemes a l'hora de desplegar una aplicació es la compatibilitat. Aquest problema sorgeix quan el codi produït en un entorn de desenvolupament determinat es porta a qualsevol altre entorn. En aquest punt poden aparèixer multitud de problemes degut a les diferències entre l'entorn de desenvolupament i l'entorn real on s'executa el codi.

La tendència actual en el mercat per a solucionar aquest problema és l'ús de contenidors. Els contenidors son una tecnologia de virtualització que permet executar una aplicació amb les seues dependències en un entorn aïllat. A més, els contenidors contenen les dependències necessàries per al funcionament del codi desenvolupat, la qual cosa elimina el problema de la compatibilitat.

Existeixen diferents tecnologies per a crear contenidors com per exemple Docker, Rocket, LXD, OpenVz entre altres. No obstant , entre les diverses tecnologies per a crear contenidors, destaca Docker degut a la seua simplicitat i facilitat d'ús.

Com ja s'ha comentat, les aplicacions basades en microserveis consisteixen en una serie de components independents. En el nostre cas, aquest components seran contenidors, i per a poder gestionar-los de manera efectiva, necessitarem una ferramenta per a donar suport a la orquestració de contenidors.

A continuació, anem a fer un anàlisis de dues de les tecnologies d'orquestració de contenidors més populars: Docker i Kubernetes.

2.2 Tecnologies de contenidors

2.2.1 Docker

Docker és un projecte de codi obert que s'encarrega d'automatitzar el desplegament de contenidors. Aquesta tecnologia ha experimentat recentment un gran creixement, i és una de les opcions més populars a l'hora de treballar amb contenidors. Malgrat què es una tecnologia relativament jove (la primera versió data del 13 de Març de 2013), s'ha fet molt popular degut a que proporciona una API senzilla i fàcil d'usar, i compta amb una gran comunitat.

Figura 2.1: Logotip de Docker.

Docker proporciona una serie de funcionalitats que no es limiten a crear contenidors. Docker també proporciona ferramentes per a poder gestionar grups de contenidors i grups de màquines físiques. Vegem-ho:

- **Contenidors**. La primera de les funcionalitats que ofereix Docker. Els contenidors es defineixen a través dels anomenats *Dockerfile*. Els *Dockerfile* son fitxers que defineixen els contenidors. En aquest fitxers s'especifiquen totes les dependències que necessita el contenidor per a executar-se, el sistema de fitxers, els arxius que contindrà, què executarà, etc... A partir d'aquí, es construeix una imatge, que es pot distribuir i executar en qualsevol altre màquina que executi Docker.

- **Registres**. Una part interessant de Docker son els registres. Els registres són servidors que permeten distribuir imatges Docker. Juguen un paper fonamental a l'hora de desplegar una aplicació basada en contenidors Docker,

ja que fan accessibles les imatges a totes les màquines on es volen desplegar contenidors. Existeixen registres públics (Docker oferix un registre públic) o registres privats. Per a muntar un registre privat, només cal baixar-se la imatge corresponent que Docker ofereix al seu registre públic. La pròpia imatge que permet muntar un registre privat és un clar exemple de la potencialitat de Docker: L'aplicació es descarrega i s'executa sense cap tipus de problema de compatibilitat i sense requerir ninguna acció per part de l'usuari per a posar-la en funcionament.

- **Docker Swarm**. Aquesta funcionalitat de Docker permet crear un *cluster* de màquines amb l'objectiu d'executar contenidors. Així, permet gestionar de manera ràpida i senzilla un grup de màquines, encarregant-se de la distribució de la càrrega, les connexions entre els contenidors que s'executen a les màquines, la monitorització de la càrrega, entre altres funcions. Existeixen dos tipus de nodes, els *Manager*, encarregats de gestionar el *cluster* i els *Worker*, encarregats de executar les tasques.

- **Docker Stack**. Aquesta es la part més alta de la jerarquia de Docker. Un *Stack* defineix una serie de serveis relacionats que comparteixen dependències. Aquests serveis, al seu torn, són en realitat contenidors Docker definits per imatges. Un *Stack* es defineix de manera similar a una imatge: S'usa un fitxer que descriu les seues característiques. Un cop definit, es pot desplegar en un *Swarm* executant una senzilla ordre. Docker s'encarrega de distribuir el treball entre els nodes disponibles, de descarregar les imatges necessàries dels registres, i de posar en marxa tota la infraestructura definida.

Després d'aquesta ullada general, podem observar com Docker s'encarrega de donar suport als problemes d'escalabilitat i portabilitat. La facilitat de ús que proporciona i la seva senzillesa a l'hora de desplegar-se (Només cal instal·lar Docker en una màquina per a poder accedir a totes les funcionalitats) han fet que sigui una de les opcions més utilitzades. Entre algunes empreses que fan ús d'aquesta tecnologia podem trobar VISA, PayPal o ADP entre altres. [1]

2.2.2 Kubernetes

Kubernetes es un sistema *open-source* per a automatitzar el desplegament, escalat, i gestió d'aplicacions basades en contenidors. Kubernetes està desenvolupat per Google, i es llança el 7 de juny de 2014.

Figura 2.2: Logotip de Kubernetes.

[1] Segons dades de la pròpia empresa.

Una de les principals diferències de Kubernetes amb Docker, es que Kubernetes no s'encarrega de crear contenidors. Per tant, qualsevol sistema desplegat amb aquesta tecnologia requereix primer haver construït els contenidors amb altres tecnologies, per exemple Docker, encara que no és la única alternativa.

Kubernetes es centra en gestionar els contenidors, i ho fa en les següents formes:

- **Kubernetes clúster** La base de Kubernetes es un *cluster* de màquines que està format per dos rols diferents: *Master* i *Node*.

  Per una banda, els nodes *Master* son els encarregats de gestionar el clúster. S'encarreguen de planificar les aplicacions, monitoritzar i mantindre el seu estat, escalar-les i del desplegament de noves versions de l'aplicació.

  Per un altre costat, tenim els que s'anomenen senzillament *Node*. La funció d'aquests es la de executar els contenidors, i per tant, realitzar el treball efectiu del *cluster*. Aquest nodes contenen a més a més un *Kubelet*, que és un agent que s'encarrega de comunicar-se amb el node *Master*. Finalment, els nodes han de ser capaços d'executar contenidors, per exemple, tindre disponible Docker.

  Cal destacar, que , al contrari de Docker, desplegar un *cluster* amb Kubernetes requereix un major esforç de configuració, i també descarregar paquets de software addicionals.

- **Kubernetes Pods** Un Pod es una abstracció creada per a allotjar una instància d'una aplicació. Així, un Pod pot contindre diversos contenidors i diversos recursos compartits. Per tant, un Pod es comporta com un *host* lògic, ja que se li assigna una única adreça IP dins del *cluster*, i conté recursos que els contenidors poden compartir.

  Els Pods són la unitat mínima de Kubernetes. Així, quan es fa un desplegament amb Kubernetes, es despleguen Pods.

- **Kubernetes Services** Un servei és una abstracció que defineix un conjunt de Pods i una política comú d'accés. Els serveis permeten agrupar diferents nodes que proporcionen una funcionalitat, i fer-los accessibles a través d'un punt d'entrada per al propi *cluster* o per a la xarxa exterior al *cluster*.

- **Escalabilitat i Desplegament** El desplegament d'una aplicació es fa de manera automàtica fent ús de les abstraccions definides. Quan es desplega un servei, Kubernetes s'encarrega de distribuir la carrega (els Pods) entre els nodes *Node* del *cluster* disponibles.

  Una característica destacable es el desplegament de noves versions sense aturar els serveis. Per a aconseguir això, es segueix una estratègia d'actualitza-

ció incremental, de manera que els Pods es van actualitzant progressivament, mantenint sempre la funcionalitat de la aplicació.

Pel que fa a la escalabilitat, Kubernetes permet augmentar i disminuir el nombre de Pods segons les necessitats. A més a més, també té opcions d'auto-escalat.

2.2.3 Comparativa

En aquest punt, s'han estudiat les característiques més rellevants de les dues tecnologies avaluades. A continuació s'analitzen les característiques més destacades de cada una de les dos opcions en una taula:

Característica	**Docker**	**Kubernetes**
Creació de contenidors	Si	No
Gestió d'un clúster	Si	Si
Equilibrat de càrrega manual	Si	Si
Auto-equilibrat de càrrega	No	Si
Facilitat d'instal·lació	Si	No *
Suport al desplegament del clúster	Si	No **

* A diferència de Docker, Kubernetes requereix un major esforç de configuració, ja que el desplegament del *cluster* no és tan senzill i requereix descarregar paquets de software addicionals.
** Encara que és pot muntar un *cluster* amb Kubernetes, la ferramenta no dona les mateixes facilitats que Docker, que només requereix un host amb Docker instal·lat i executar una ordre.

Taula 2.1: Comparativa de les tecnologies analitzades.

Després d'haver analitzat les dues alternatives, podem arribar a les següents conclusions:

- Les dos ferramentes proporcionen una funcionalitat molt similar. A nivell funcional, les úniques diferències son que Docker permet crear les imatges per als contenidors i Kubernetes no, i que Kubernetes compta amb auto-escalat.
- Muntar un *cluster* amb Kubernetes és una tasca que requereix major esforç i experiència. No obstant, el resultat es major flexibilitat i una configuració més adaptada a les necessitats concretes. Per contra, Docker permet desplegar un *cluster* de manera molt senzilla, sense requerir coneixements.
- Docker proporciona una ferramenta "tot en un", només s'ha de instal·lar Docker per a accedir a totes les funcionalitats. Per altra banda, Kubernetes

requereix descarregar més paquets (encara què permet una major configuració de tot el sistema).

Finalment, la ferramenta seleccionada per a aquesta implementació es Docker, ja que dona suport a les necessitats del projecte. Des de crear els contenidors amb les imatges, fins a monitoritzar un *cluster*, aprovisionar els nodes i escalar la aplicació, totes aquestes funcionalitats les obtenim amb només una ferramenta: Docker.

2.3 Comunicacions

Una part fonamental de l'arquitectura de microserveis son les comunicacions entre els diferents components. En aquest projecte, es parteix d'una aplicació que proporciona el director del projecte. Així, a continuació es descriu directament la tecnologia emprada.

2.3.1 MQTT

MQTT *Message Queue Telemetry Transport* és un protocol lleuger per a la comunicació entre màquines (*machine-to-machine*). Segueix un esquema de comunicació publicador/subscriptor.

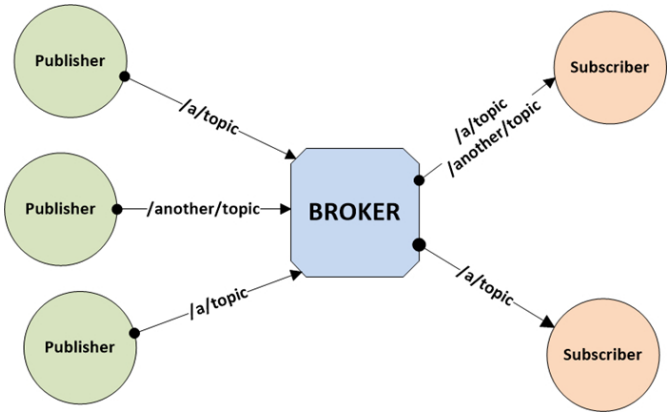

Figura 2.3: Exemple de comunicació MQTT.

En aquesta tecnologia, hi ha un node que s'encarrega de gestionar les comunicacions, anomenat *broker*. Els missatges s'organitzen en *topics*.

Per a comunicar-se, els clients es subscriuen i publiquen al *topics*. El broker s'encarrega de fer arribar als subscriptors els missatges publicats als *topics* als que s'han subscrit.

2.3.2 REST

REST (en anglès *REpresentational State Transfer*) és un estil d'arquitectura que defineix estàndards per a la comunicació entre sistemes. Aquesta arquitectura es caracteritza per ser *stateless* (El participants d'una comunicació no coneixen l'estat dels altres, però son capaços d'interpretar tots els missatges de la comunicació) i per des-acoplar les dues parts d'una comunicació.

En REST, els elements d'informació s'organitzen com a recursos. Aquest recursos es defineixen amb una URI (*Universal Resource identifier*). Així, per a interactuar amb un sistema REST, s'accedeix a la URI dels recursos mitjançant els *verbs* de HTTP (generalment GET, POST, PUT i DELETE).

Per tant, aquest protocol permet des-acoplar els participants d'una comunicació, i a més a més, compta amb el gran avantatge de que funciona amb HTTP. Actualment una gran quantitat de dispositius son capaços de comunicar-se a través de HTTP. Per tant, l'ús de REST permet garantir que el sistema és accessible des d'una gran varietat d'altres sistemes, sempre que siguin capaços de generar peticions HTTP.

2.3.3 JSON

JSON (en anglès *JavaScript Object Notation*) es un format de text lleuger per al intercanvi de dades. Encara que té el seu origen al llenguatge de programació JavaScript, s'ha popularitzat com a una alternativa a XML.

```
1 {
2     "Nom":"Axel",
3     "Cognoms":"Guzman␣Godia",
4     "Aficions":["musica","natacio","informatica"],
5 }
```

Listing 2.1: Exemple de missatge JSON.

El principal avantatge d'aquest format és que permet transmetre informació amb una quantitat de dades relativament petita. A més a més es un format independent del llenguatge (ja que realment és només una cadena de text organitzada seguint unes normes). Addicionalment, Javascript interpreta aquest format de forma nativa.

2.4 Conclusions

En aquest capítol s'ha fet un anàlisis de les tecnologies que donen suport a l'arquitectura de microserveis. Després, s'ha fet una comparativa per a analitzar quines són les tecnologies més adequades per al projecte.

A més a més, s'han presentat les tecnologies de comunicació que s'empren a l'aplicació objectiu d'aquest projecte. Conèixer la tecnologia de comunicació utilitzada resulta imprescindible per a poder generar un disseny vàlid.

Com a resultat d'aquest capítol, s'ha decidit la tecnologia a utilitzar. A més a més, s'ha estudiat la tecnologia que ja forma part de l'aplicació objectiu.

Abans de començar amb el disseny, cal avaluar l'aplicació que es vol transformar. En el pròxim capítol, s'analitzen les característiques de l'aplicació a transformar amb l'objectiu de conèixer quins són els reptes que cal superar per a avançar cap a una solució basada en microserveis..

Capítol 3

Cas d'estudi

L'objectiu d'aquest capítol es donar a conèixer el cas particular estudiat en aquest document i al què es proporciona una solució. A continuació s'analitza l'aplicació objectiu, les característiques identificades i les característiques desitjables [1].

3.1 Infraestructura de carreteres intel·ligents

L'aplicació objectiu es una aplicació que gestiona carreteres intel·ligents. L'objectiu d'aquesta, es permetre la connectivitat i l'intercanvi d'informació entre els vehicles que circulen per les carreteres que gestiona el sistema i tots els altres actors que hi intervenen (senyals de trànsit, semàfors, organismes de regulació del trànsit, etc...).

La aplicació en conjunt està formada per diversos nodes computacionals. Cadascún d'aquests nodes proporciona una funcionalitat. Els nodes interactuen els uns amb els altres mitjançant cues de comunicació.

Encara que l'apliació de partida està dividida en diferents nodes computacionals, aquests están implementats en forma de monolit, i poden proporcionar un o diversos serveis.

Per exemple, el servei de tràfic s'encarregar de publicar alertes, gestionar semàfors, tancar i obrir carreteres, etc. En canvi, els vehicles connectats actúen com a clients, consumint informació e interactuant amb aquest servei (entre altres).

[1] A partir d'aquest punt, es fa referència al cas d'estudi com a l'aplicació

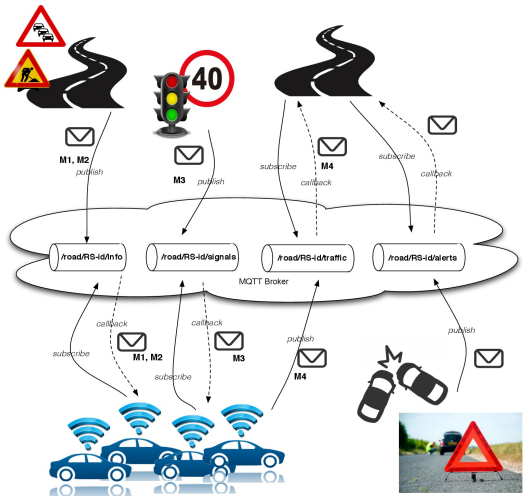

Figura 3.1: Diagrama de l'apliació.

La Figura 3.1 mostra un diagrama de l'apliació resumit. Aquest diagrama representa els diversos nodes computacionals. A la part de baix tenim els vehicles conectats, que actúen com a clients. Al centre, es representa una cua que permet la interacció entre els diversos nodes. Finalment, a la part superior es representen els nodes que proporcionen els serveis. Aquests s'encarreguen de processar la informació de les cues per a interactuar amb els vehicles o amb altres nodes computacionals.

Aquest diagrama mostra només una part de l'aplicació, ja que potencialment pot estar formada per altres nodes computacionals amb diverses funcionalitats.

La implementació d'aquesta aplicació a la que es fa referència durant tot el treball ha sigut proporcionada per el director del projecte. Així, el punt de partida es una aplicació monolítica funcional, mentre que l'objectiu en última instància, es transformar-la en una aplicació basada en microserveis.

No es disposa d'una especificació formal de les funcionalitats de l'aplicació, que està en fase de desenvolupament. No obstant, si es disposa d'informació sobre les següents característiques rellevants:

- La primera característica és l'ús intensiu de comunicacions. Totes les comunicacions entre els usuaris, l'aplicació i altres elements connectats es fan mitjançant la xarxa.
- Una altra característica important, es la variabilitat en el nombre d'usuaris. Com els usuaris de l'aplicació son els vehicles que circulen per la carretera, es

esperable que la càrrega pugui variar de manera important. Així, es possible que en hora punta (per exemple a l'hora d'entrada o sortida del treball) es produeixi un gran increment en el nombre d'usuaris. En canvi, durant la nit, el nombre d'usuaris serà més reduït.

- La aplicació proporciona diversos serveis diferenciats. Per exemple, la aplicació publica alertes quan es produeix un accident, però també s'encarrega d' actuar sobre elements connectats. Per exemple, enviar un missatge a un semàfor connectat per a tancar un carril d'una carretera, o enviar un missatge de text per a un panell informatiu.

Amb aquest primer anàlisis, ja podem detectar que la escalabilitat serà un factor important. L'ús de comunicacions a través de la xarxa també es un factor important, ja que a l'hora que d'escalar l'aplicació, s'ha de garantir que els usuaris continuen poden connectar-se als nodes. Finalment, també es despren que l'aplicació proporciona una serie de funcionalitats independents entre sí.

A continuació, analitzem els detalls d'implementació que tenen impacte a l'hora de desenvolupar una solució.

3.2 Implementació actual

Arribats a aquest punt, ja s'han exposat les característiques de l'aplicació a grans trets. A continuació, s'analitzen els detalls de la implementació actual, i el seu impacte:

- **Aplicació monolítica.** Es tracta d'una aplicació monolítica. Això vol dir que la funcionalitat de l'aplicació es concentra en un únic node, que executa tot el codi i s'encarrega de realitzar totes les accions. En aquest cas Aquesta característica es l'antagonista de la arquitectura de microserveis. La primera part de la nova solució consisteix en refactoritzar la aplicació per a transformar-la en una aplicació basada en microserveis.

- **Cues com a interfície amb els usuaris.** Una altra característica rellevant és l'us de les cues com a interfície. El funcionament de l'aplicació es basa en la publicació i subscripció a cues de missatgeria. Tant els clients com el servidor publiquen, llegeixen e interpreten missatges de les cues. Aquesta característica te l'avantatge de des acoplar les parts que intervenen en les comunicacions.

  Així, la gestió de les cues la fa un *broker* MQTT. Aquest *broker* és un element independent, que s'executa com a un servei addicional al codi de l'aplicació. el *broker* s'encarrega de gestionar les subscripcions dels clients i els missatges de les cues.

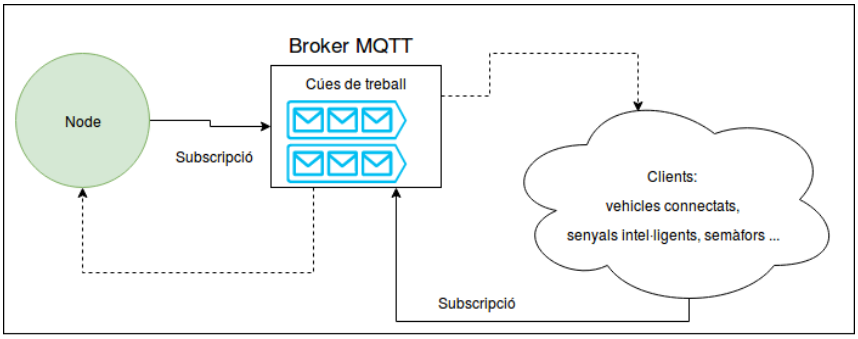

Figura 3.2: Esquema de comunicació amb les cues de l'aplicació inicial.

Malgrat els avantatges que aporta, també introdueix una certa complexitat a l'hora de gestionar la escalabilitat. Per una part, cal gestionar i escalar les cues. Per altra banda, quan l'aplicació es refactoritzi en microserveis, serà necessari repartir les subscripcions a la cua per tal d'evitar que diversos microserveis s'encarreguin de les mateixes tasques. Més endavant en parlarem amb més detall d'aquest aspecte.

- **Carreteres com a unitat de treball.** Com ja s'ha explicat, la aplicació s'encarrega de gestionar carreteres. Aquestes carreteres es divideixen en *road-segments* o seccions. Cada carretera es composa d'un nombre indeterminat de seccions. Aquesta divisió te relació amb l'apartat anterior, ja que cada cua representa un *road-segment*.

Capítol 4

Disseny de la solució

L'objectiu d'aquest apartat es descriure una solució que permet refactoritzar l'aplicació objectiu en una aplicació basada en microserveis, i a més a més, desplegar la infraestructura necessària per a donar-li suport.

El disseny de la solució s'estructura en 3 punts fonamentals:

- Refactoritzar l'aplicació en microserveis.
- Gestionar la interacció dels microserveis amb les cues.
- Muntar un clúster amb Docker per a executar els microserveis.

4.1 Limitacions

Degut a la extensió limitada del treball, en aquest disseny no s'inclouen els següents aspectes:

- **Escalabilitat de les cues.** La funcionalitat es basa en un ús intensiu de les cues de comunicació. Aquestes cues son gestionades per un *broker* MQTT. A mesura que s'incrementa el nombre d'usuaris, aquest punt es un potencial problema ja que arribara un moment en què no podrà donar suport a tanta càrrega. Es necessari avaluar i definir algun mecanisme per a garantir l'escalabilitat de les cues.
- **Seguretat.** En aquest disseny no s'inclouen els aspectes de seguretat. Es necessari avaluar els potencials riscos de seguretat e incorporar mecanismes per a garantir què la aplicació es fiable i evitar usuaris maliciosos.

4.2 Refactorització de l'aplicació

El primer pas per a poder canviar a una arquitectura basada en microserveis es refactoritzar l'aplicació. Aquesta tasca consisteix bàsicament en analitzar la funcionalitat de la aplicació, i aïllar aquelles funcionalitats independents entre sí.

Per tant, s'ha d'estudiar la aplicació i veure quines parts son candidates a convertir-se en microserveis. Com que l'aplicació està en desenvolupament, no es disposa d'una especificació formal. No obstant, vejam un exemple amb dues funcionalitats conegudes que ja estan implementades:

- Publicar alertes d'accidents. Quan un vehicle connectat publica a una cua que ha patit un accident, la aplicació s'encarrega de generar una alerta i publicar-la a la cua corresponent.
- Tancar un semàfor. Sota determinades condicions (volum de tràfic, accidents, vehicles prioritaris...) l'aplicació pot enviar un missatge per a tancar un semàfor.

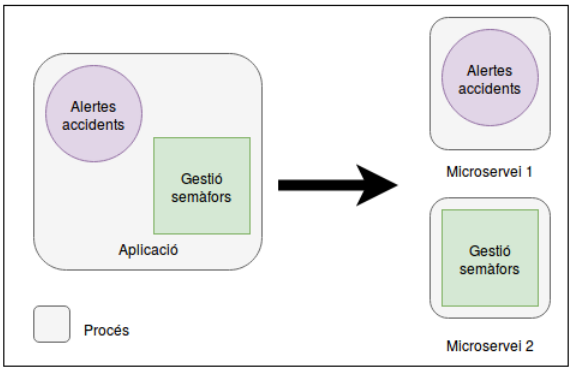

Figura 4.1: Exemple de refactorització en microserveis.

Aquestes dues funcionalitats estan ben diferenciades. Ambdues processen els missatges publicats a les cues, i realitzen determinades accions. No obstant, no hi ha ninguna dependència entre elles i per tant poden convertir-se cada una en un microservei independent.

L'aproximació habitual és comunicar els microserveis entre ells a través d'APIs. Per exemple, quan es converteix una funció del codi en un microservei, les futures crides a la funció es substitueixen per una petició a l'API del node que executa el microservei.

En aquest cas, per al microservei d'alertes, resulta interessant disposar d'una manera de comunicar-li quins son els segment de carretera que ha de gestionar. Per tant , se li ha afegit una API que permet afegir, llevar o consultar els segments de carretera que gestiona el node.

Finalment, només queda empaquetar cada un dels microserveis com a una imatge Docker. Això ens permetrà distribuir les imatges per a poder desplegar fàcilment nous nodes. Més endavant s'explica de manera més detallada com es distribueixen les imatges dels microserveis.

4.3 Gestió de les cues

4.3.1 Problemàtica

La gestió de les cues es un aspecte crític degut a que gran part de les funcionalitats que presenta l'aplicació es basen en l'ús de les cues.

Per a poder veure perquè s'ha de gestionar aquest aspecte,tornem a l'exemple anterior dels microserveis. Una vegada tenim el microservei que s'encarrega de gestionar les alertes d'accidents, podem desplegar-ne una o més instàncies, segons les necessitats de cada moment.

Com tenim el microservei empaquetat en una imatge Docker, en qualsevol moment podem llançar una altra instància per a aconseguir escalabilitat. Com les imatges són idèntiques, cada cop que llancem una imatge, el codi què s'executa es exactament el mateix: Cada instància es subscriu a les mateixes cues i les processa de forma idèntica.

La següent imatge il·lustra aquest problema: (explicació baix)

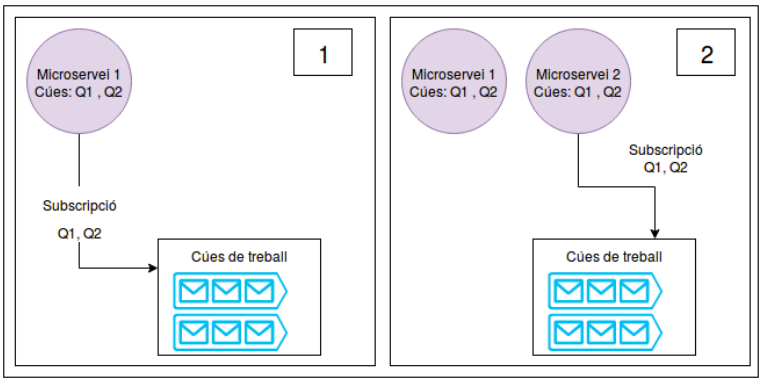

Figura 4.2: Exemple del problema amb les cues.

1. S'inicia una instància i es subscriu a les cues Q1 i Q2.

2. S'inicia una altra instància. Com té el mateix codi que la instància anterior, es subscriu també a Q1 i Q2.

Aquesta problemàtica és específica de l'aplicació, i Docker no dona suport per a gestionar-la. Per tant, necessitem algun mecanisme per a repartir la càrrega cada cop que despleguem una nova instància del microservei d'alertes.

4.3.2 Microservei gestor de cues

La solució proposada, es desenvolupar un microservei addicional que s'encarregui de controlar l'assignació de cues als nodes. Aquest microservei s'haurà d'encarregar de repartir les cues cada cop que s'afegeixen o s'eliminen nodes. Això requereix a més a més, què els nodes es comuniquin amb el gestor de cues per a poder obtindre la càrrega.

La següent seqüencia d'imatges és un exemple de la tasca que ha de realitzar el microservei gestor de cues. En aquest exemple, una nova instància del microservei d'alertes comença a executar-se en un sistema en el que ja estan funcionant dues instàncies del mateix microservei:

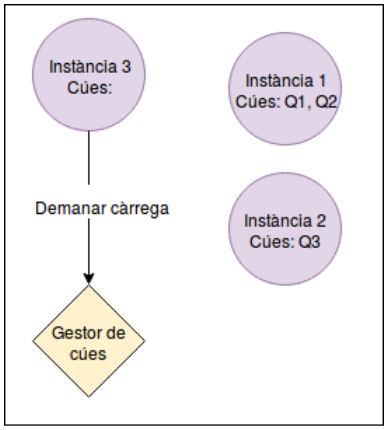

Figura 4.3: S'acaba d'iniciar una nova instància (Instància 3) del microservei d'alertes. La nova instància es comunica amb el gestor de cues per a demanar que se li assigni càrrega.

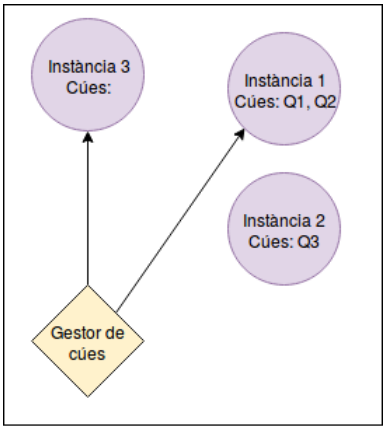

Figura 4.4: El microservei gestor de cues processa la sol·licitud. Després envia els missatges necessaris per equilibrar la càrrega. En aquest exemple, es notifica a la instància 1 que ha de deixar de gestionar la cua Q1, i a la vegada, es notifica a la instància 3 que ha de gestionar aquesta cua.

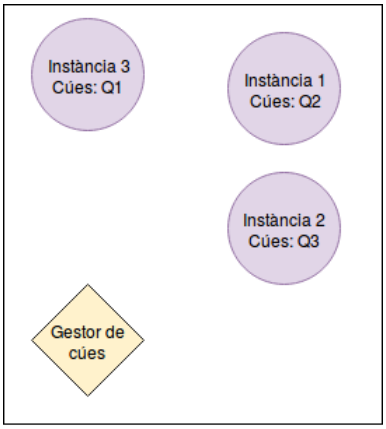

Figura 4.5: Finalment, la càrrega s'ha distribuït entre les instàncies disponibles.

El microservei de gestió de cues ha de comptar amb les següents característiques:

- **API.** Es necessita exposar una API per a que les noves instàncies puguin demanar càrrega. El gestor ha d'escoltar i processar els missatges que li arriben per aquesta API.

- **Algorisme per a distribuir el treball.** L'algorisme que s'usa per a distribuir la càrrega. Idealment, l'algorisme ha de distribuir la càrrega de la manera més equilibrada possible entre les instàncies disponibles.
- **Comunicació amb els microserveis.** Finalment, el gestor ha de ser capaç de comunicar-se amb els microserveis per a modificar la seua assignació de cues. Aquesta comunicació es fa interactuant amb l'API que exposen els microserveis.

4.4 Clúster amb Docker Swarm

Per últim, per a executar els microserveis, es desplegarà un *cluster* amb Docker Swarm.

Per a poder crear el *cluster* i començar a executar els microserveis es necessiten una serie d'accions. A continuació s'expliquen amb detall cada una de les accions necessàries.

4.4.1 Definir un Docker Stack

Com ja s'ha estudiat anteriorment , un Docker Stack es una abstracció que defineix una serie de serveis relacionats entre sí. Per a definir un Docker Stack s'ha de crear un fitxer amb una sintaxis concreta, que defineix els aspectes fonamentals de l'aplicació. Després, Docker s'encarregarà d'interpretar la definició i realitzar les accions necessàries per a desplegar l'aplicació sobre el *cluster*.

Per la nostra aplicació s'han de definir les següents característiques rellevants:

- **Imatges.** Les imatges que es desplegaran, i també, des d'on s'obtindran. Aixo fa referència al registre d'imatges que s'utilitzarà per a descarregar les imatges en cada node.
- **Rèpliques.** Es defineix el número d'instàncies de cada un dels microserveis. Docker s'encarregarà d'aconseguir el número de rèpliques distribuint el treball entre els nodes disponibles al *cluster*.
- **Xarxes i connexions.** Es defineix la xarxa interna al *cluster*, així com les correspondències entre els ports de cadascun dels microserveis i els ports accessibles des de la xarxa interna.
- **Altres paràmetres.** També es defineixen alguns paràmetres com el límit en l'us de recursos de cada microservei (respecte al host que l'executa), política de reinicii (per exemple quan es produeix un error en un node) entre altres característiques.

4.4.2 Desplegar un servei registry

Un avantatge de desplegar un *cluster* amb Docker es que l'aprovisionament dels nodes es fa automàticament. Per a poder possibilitar aquesta característica, els nodes necessiten descarregar-se les imatges que contenen els microserveis des d'un registre d'imatges. En aquest punt tenim dues alternatives: utilitzar el servei públic que ofereix la pròpia empresa Docker o muntar un servei privat. S'ha optat per la segona opció per a garantir els aspectes de confidencialitat de l'aplicació inicial.

Aquest servei encarregat de proporcionar imatges als nodes s'anomena *registry*. La pròpia empresa Docker proporciona una imatge gratuïta que conté la implementació d'un *registry*. Per tant, per a resoldre aquest aspecte farem ús d'aquesta imatge, evitant les complexitats de desenvolupar un servei propi.

4.4.3 Inicialitzar el clúster i afegir nodes

Un cop ja tenim resolt l'aspecte de les imatges i la definició dels serveis que executarà el *cluster*, només queda posar en marxa el nodes. Per a fer això , el primer pas es iniciar un node mànager. Un cop s'inicia un node *manager*, es genera un *token* que permet afegir nodes al *cluster* executant una simple ordre.

Quan ja tenim els nodes afegits al *cluster*, podem desplegar l'aplicació definida prèviament (com a un Docker Stack).

4.4.4 Connectivitat del clúster

Per el que respecta a la solució que s'ha dissenyat, tenim dos tipus de connectivitats: La connectivitat interna i la connectivitat externa.

Connectivitat externa del clúster

Tots els nodes que formen part d'un Docker Swarm poden rebre connexions als ports dels serveis desplegats al *cluster*. Això vol dir que si es desplega un servei al *cluster* amb un port determinat, tots els nodes del *cluster* acceptaran connexions a aquest port. Fins i tot, aquells nodes que no tenen ninguna instància en execució que doni suport al servei.

Per a poder possibilitar això Docker s'encarrega d'encaminar internament les peticions externes. La Figura 4.6 descriu el funcionament d'aquest encaminament, que s'anomena *ingress routing mesh*.

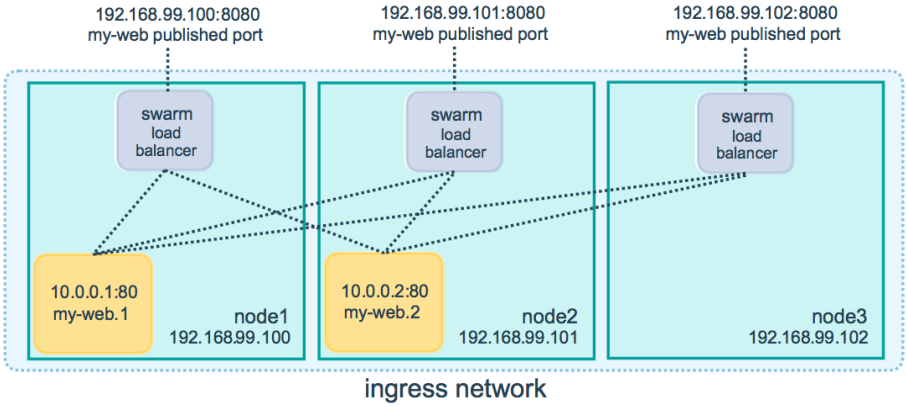

Figura 4.6: Connectivitat interna de Docker Swarm

Aquesta imatge representa un Docker Swarm format per tres nodes. Tots els nodes atenen peticions per al servei. Un cop han rebut una petició, els nodes la dirigeixen a un contenidor del *cluster*, be dins del mateix node o bé cap a un altre node.

Aquest encaminament, es fa a nivell d'equilibrat de càrrega. Això es fa seguint una política *Round-Robin*.

Connectivitat interna del clúster

Per a que els microserveis puguin comunicar-se, es imprescindible que tinguen connectivitat els uns amb els altres.

Aquesta connectivitat s'ofereix mitjançant una xarxa especial de Docker que s'anomena *overlay*.

Una xarxa de tipus *overlay* és un tipus de xarxa que connecta els contenidors. Això vol dir, que dins d'aquesta xarxa cada contenidor té una adreça IP única.

Aquesta xarxa fa que la connexió entre contenidors es faci de manera transparent. Així, els contenidors d'aquesta xarxa es comuniquen entre ells com si es trobaren dins de la mateixa xarxa. Docker s'encarrega de transportar els missatges per la xarxa entre els diferents nodes.

Aquesta connectivitat interna resulta imprescindible per a les comunicacions entre microserveis, i a més a més, possibilita que un contenidor es pugui desplaçar d'un node a un altre sense afectar el funcionament, ja que els contenidors conserven la seua adreça IP interna.

4.5 Conclusions

En aquest capítol s'exposen totes les accions que s'han de prendre per a poder transformar l'aplicació inicial. Com a resultat d'aquest capítol, s'han determinat totes les accions que s'han de prendre i s'ha justificat el paper que juguen en el resultat final.

Per a poder veure millor les implicacions del disseny proposat, a continuació s'adjunta una imatge comparativa del disseny inicial de l'arquitectura i el disseny proposat:

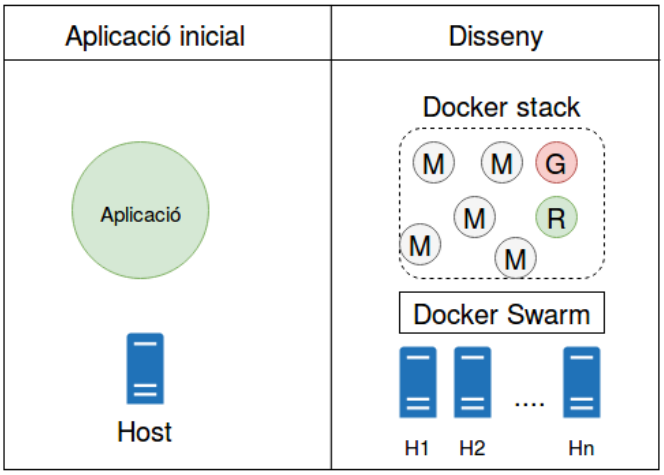

Figura 4.7: Comparativa entre el disseny inicial i el disseny proposat.

Aquesta figura representa els components de l'aplicació, i alhora els hosts que els executen.

A l'esquerra, tenim el disseny de l'aplicació de la que partim. En aquest disseny, l'aplicació es un únic component que s'executa sobre un *host*.

A la dreta, el resultat del disseny està format primer per un grup de *hosts*, que integren el *cluster* dirigit per Docker Swarm. Al mateix temps, sobre aquesta infraestructura s'executa una col·lecció de microserveis definida en un Docker Stack. Aquest Docker Stack equival a la figura d'aplicació en el disseny inicial. El microserveis que s'encarreguen del registre d'imatges i la gestió de les cues s'identifiquen amb una 'R' i una 'G', respectivament. La resta de microserveis (representats amb una 'M') poden ser qualsevol tipus de microserveis, com ara el gestor d'alertes.

Partint d'aquesta visió global, el disseny proposat permet obtindre els següents avantatges:

- **Escalabilitat.** La escalabilitat la podem gestionar de dues formes. Per una part, podem afegir més nodes al *cluster* per a augmentar la capacitat. Per altra part, podem escalar de manera individual cada un dels microserveis segons les necessitats específiques. Això proporciona un avantatge front al punt de partida, en el que si volíem escalar l'aplicació, s'havia de replicar sencera.

- **Agilitat.** Amb aquest nou disseny el desenvolupament de cadascun dels components (o microserveis) queda deslligat dels altres. Així , es pot treballar en millorar l'algoritme de gestió de cues sense haver de conèixer amb detall la resta de components. A més a més, els canvis que es despleguin en cada versió d'un microservei concret només afectaran a aquesta part de l'aplicació. Això afavoreix la integració contínua de noves funcionalitats i millores.

- **Mantenibilitat.** El manteniment de cadascun dels components resulta ara més senzill. Això es deu a que ara, els components són unitats independents amb una mida de codi molt més reduïda. Malgrat això, cal tenir en compte que la mantenibilitat del conjunt resulta ara més complexa quan s'introdueixen canvis que afectin a les interfícies dels components (les seues APIs).

- **Deslligament d'una tecnologia específica.** Amb aquest disseny es possible introduir noves funcionalitats utilitzant diversos llenguatges de programació. Això ajuda a poder desenvolupar nous components amb el llenguatge de programació més apropiat, enlloc d'estar limitat per la decisió del llenguatge inicial.

Ara que ja s'ha presentat tot el disseny de la solució, al pròxim capítol es detalla la implementació d'aquesta.

Capítol 5

Prototip i proves

En aquest capítol es desenvolupa el disseny presentat a l'aprtat anterior. Al final d'aquest apartat obtenim un prototip funcional que implementa el disseny proposat.

Aquesta tasca té dos parts fonamentals. Per una part, s'ha de desenvolupar el codi necessari per a implementar els microserveis. Per una altra banda, s'ha de posar en marxa la infraestructura que executarà els microserveis.

5.1 Desenvolupament de codi

En aquesta primera part, s'aborden els següents punts:

- **Refactorització.** Procés per el qual es transforma l'aplicació inicial en una serie de microserveis. En aquest apartat es refactoritza un microservei: el microservei d'alertes.
- **Desenvolupament del gestor de cues.** En aquesta secció es descriu la implementació del microservei encarregat d'assignar les cues als microserveis d'alertes.

5.1.1 Refactorització

El primer pas per a desplegar la infraestructura es convertir l'aplicació monolítica en una aplicació basada en microserveis. Per a aquest prototip no és disposa de tota l'aplicació. Així, només s'ha realitzat aquest pas amb una funcionalitat de la que ja s'ha parlat amb anterioritat: El servei d'alertes d'accidents.

Aquesta funcionalitat s'encarrega de llegir l'estat de les carreteres i publicar alertes quan es produeix un accident.

La primera part es aïllar el codi que implementa aquesta funcionalitat. A més, per a fer que el nou microservei pugui interactuar amb l'exterior, s'ha afegit una API que permet gestionar la càrrega. Les opcions que ofereix la API son afegir,llevar o consultar els segments de carretera gestionats per el microservei.

Microservei	Funcionalitat	API
road_alerts	Es subscriu a una serie de road-segments. Quan es publica un accident en un road-segment, publica una alerta per la cua d'alertes.	- Consultar road-segments gestionats - Afegir road-segment - Llevar road-segment

Taula 5.1: Microservei road_alerts

El següent pas és empaquetar aquest microservei en una imatge Docker.

5.1.2 Creació d'imatges Docker

Un cop tenim ja tot el codi d'un microservei, abans de poder crear un contenidor, s'ha de crear un Dockerfile. Aquest fitxer, conté la definició del contenidor, es a dir, els fitxers que conté i les dependències, entre altra informació rellevant.

```
1  FROM node:alpine
2
3  #Create app directory
4  WORKDIR /usr/src/app
5
6  #App dependencies
7  COPY package*.json ./
8
9  #Install dependencies
10 RUN npm install
11
12 #App source
13 COPY . .
14
```

```
15 EXPOSE 3000
16
17 #Start application
18 CMD ["npm", "start"]
```

Listing 5.1: Dockerfile del gestor de cúes.

Un cop tenim el Dockerfile preparat, construïm la imatge. Per a construir la imatge executem:

```
1 docker build -t microservei .
```

Listing 5.2: Ordre per a construir una imatge Docker.

```
axguzgo@HP:/tmp/APITrain$ docker build  -t microservei .
Sending build context to Docker daemon  22.37MB
Step 1/7 : FROM node:alpine
 ---> 1c517a20b2fd
Step 2/7 : WORKDIR /usr/src/app
 ---> Using cache
 ---> 7d3e3ea21823
Step 3/7 : COPY package*.json ./
 ---> Using cache
 ---> a58a17b0d4c2
Step 4/7 : RUN npm install
 ---> Using cache
 ---> d65549b27523
Step 5/7 : COPY . .
 ---> 755c6f5c64f4
Step 6/7 : EXPOSE 3000
 ---> Running in 75628472d3c0
Removing intermediate container 75628472d3c0
 ---> cf1c92779fec
Step 7/7 : CMD ["npm", "start"]
 ---> Running in 8aa85ed51670
Removing intermediate container 8aa85ed51670
 ---> 1a11d6e88899
Successfully built 1a11d6e88899
Successfully tagged microservei:latest
```

Figura 5.1: Construcció d'una imatge Docker.

Immediatament, Docker comença a executar les instruccions del Dockerfile i a descarregar tots els paquets necessaris. Com a resultat, obtenim una imatge Docker amb el microservei.

Aquesta tasca s'ha de realitzar per a cada un dels microserveis. En aquest prototip, només tenim dos microserveis, per una banda el servei d'alertes, i per l'altra el gestor de cues.

Com a resultat d'aquest punt, tenim les dues imatges Docker dels microserveis.

5.1.3 Desenvolupament del gestor de cues

Una de les parts importants per al funcionament de la infraestructura és la gestió de les cues. Per a resoldre aquesta part, s'ha implementat un microservei que s'encarregarà de gestionar aquest aspecte.

Per a implementar aquest microservei s'ha decidit utilitzar Node.js. Aquesta elecció es basa en les característiques específiques d'aquest llenguatge que esta fortament orientat cap al desenvolupament web.

Figura 5.2: Logo de Node.js.

Aquesta elecció es possible degut a que amb el nou disseny basat en microserveis, ja no es necessari utilitzar el llenguatge de programació original de l'aplicació per a les noves funcionalitats.

El microservei pot resumir-se en els següents aspectes:

- **API.** El microservei exposa una API REST. Mitjançant aquesta API, els nous microserveis d'alertes notifiquen que s'han posat en funcionament. Aquesta notificació es fa mitjançant una petició de tipus POST, notificant l'identificador del node, la llista de segments de carretera que gestiona actualment i l'adreça de l'API del microservei.

```
1        {
2            "NodeId":"Node1",
3            "roadSegments":["CV17","CV12","CV11","CV0","CV10"],
4            "address":"http://localhost:4000/alerts"
5        }
```

Listing 5.3: Contingut del misatge per a donar d'alta una nova instància de microservei d'alertes.

  A partir d'aquests missatges, el microservei pot conèixer quants microserveis d'alertes hi ha en funcionament, quins road-segments gestionen i quines són les seues adreces per a poder interactuar amb ells.

- **Algorisme.** El gestor de cues processa les peticions. L'algorisme per a repartir els segments de carretera es senzill. Primer s'acumulen tots els segments de carretera gestionats per cada node. A continuació, s'anul·la

l'assignació de cada node, es a dir s'arriba a un estat on ningun node gestiona ningun road-segment. Un cop tenim tots els nodes sense cap road-segment assignat, el conjunt dels segments de carretera es reparteix entre totes les instàncies a l'estil *Round-Robin*.

- **Comunicació amb els microserveis d'alertes.** El microservei implementa diverses funcions per a interactuar amb l'API dels microserveis d'alertes.

La implementació que s'ha fet d'aquest microservei assumeix que els microserveis sempre estan actius. Aquesta assumpció òbviament no és realista, ja que els microserveis poden deixar d'estar disponibles al llarg del temps per diverses causes. No obstant, aquesta part no s'ha cobert en aquest primer prototip.

Com a resum, el microservei gestor de cues que s'ha implementat proporciona la següent funcionalitat:

- Escoltar peticions de les noves instàncies dels microserveis d'alertes.
- Processar les peticions i re distribuir els segments de carretera entre totes les instàncies conegudes (Que ja s'han comunicat prèviament amb el gestor de cues).
- El gestor no monitoritza l'activitat de les instàncies a les que ha assignat treball, ni es capaç de detectar si estan sobrecarregades o han deixat de funcionar.

Microservei	Funcionalitat	API
road_manager	S'encarrega de repartir els road-segments entre els microserveis d'alertes disponibles.	- Consultar assignació de nodes i road-segments - Donar d'alta node

Taula 5.2: Microservei road_manager

5.2 Desplegament de la infraestructura

Un cop ja s'han desenvolupat els microserveis necessaris per a aquest prototip, s'ha de posar en marxa la infraestructura necessària. La infraestructura que es desplegarà per a aquesta prova és descriu en el següent diagrama:

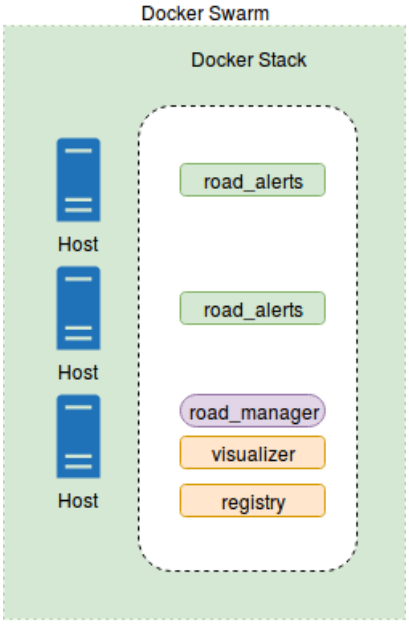

Figura 5.3: Prototip.

Aquesta imatge representa un *cluster* muntat amb Docker Swarm que està format per tres *hosts* (màquines, virtuals o físiques). A més a més, dos d'ells executen només el microservei d'alertes, mentre que l'últim executa els microserveis de visualització, el registre d'imatges i el gestor de carreteres. Tots aquest contenidors estan desplegats dins d'un Docker Swarm.

Per a arribar a aquest desplegament, s'han seguit els següents passos:

- **Posada en marxa del servei de visualització.** Per a poder observar la evolució de la infraestructura, s'utilitza un microservei que permet observar gràficament l'estat del *cluster*.
- **Creació d'un clúster amb Docker Swarm.** Aquest secció explica com s'ha preparat el *cluster* que serà l'encarregat d'executar els microserveis.
- **Creació d'un registre d'imatges.** Aquest apartat descriu com s'ha preparat el registre d'imatges i quins passos s'han donat per a fer disponibles les imatges als usuaris del registre.
- **Desplegament de l'aplicació sobre el clúster.** Un cop es disposa de tota la infraestructura necessària, s'ha de desplegar l'aplicació. Això significa

repartir i començar a executar els microserveis entre els nodes disponibles del *cluster*.

- **Escalat de microserveis d'alertes.** En aquest punt s'afegeixen noves instàncies del microservei d'alertes. L'objectiu es comprovar com els missatges arriben al gestor de carreteres i es redistribueix la càrrega.

5.2.1 Servei de visualització

Per a poder observar la evolució de la infraestructura, Docker compta amb algunes ordres que permeten conèixer l'estat del *cluster*. Encara que aquestes ordres son suficients per a poder examinar l'estat, existeix una opció més convenient: el servei Docker Swarm visualizer.

Docker Swarm visualizer és un servei que mostra l'estat d'un *swarm* de forma gràfica a través d'una interfície gràfica. A aquesta interfície s'accedeix a través d'un API.

Seguint la filosofia de Docker, aquest servei es pot descarregar des de el repositori d'imatges de Docker com a un contenidor més. Aquest servei també resulta de gran utilitat per a depurar el funcionament del desplegament i detectar problemes.

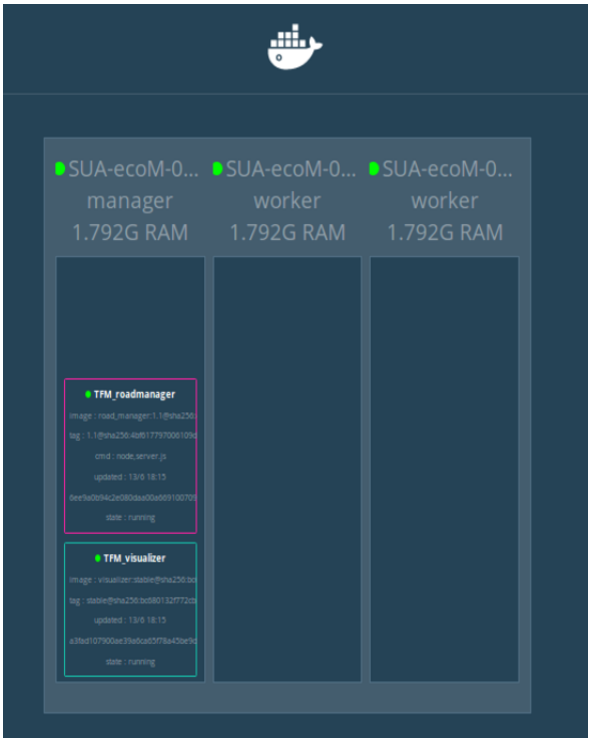

Figura 5.4: Servei de visualització.

La imatge superior correspon a la interfície del servei. En aquest exemple, hi ha un *cluster* format per tres nodes. En cadascun dels nodes es pot observar si es tracta d'un *manager* o un *worker*.

A més a més, també es mostren els contenidors que està executant cada node. En aquest cas, només el primer node està executant. Aquest node té en marxa un contenidor que executa el servei de visualització, i un contenidor que executa el microservei *road_ manager*.

Microservei	Funcionalitat	API
visualizer	Mostra l'estat d'un Docker Swarm	- Mostrar l'estat del Swarm actual

Taula 5.3: Microservei visualizer.

5.2.2 Creació del clúster amb Docker Swarm

Un cop s'han desenvolupat els microserveis necessaris, s'ha de configurar el *cluster*.

Per a iniciar un *cluster* amb Docker Swarm, el primer pas es crear un node *manager*. Per a això , es suficient amb anar a una de les màquines disponibles i executar:

```
1 docker Swarm init
```

Listing 5.4: Ordre per a iniciar un clúster amb Docker Swarm.

```
axguzgo@HP:~$ docker swarm init
Swarm initialized: current node (l0su9ujfrx19uyrpfomi1uq71) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3dyi0ezm3fhnz3d0mi7i1r7xx5u4f7ymvcanw0ysw9m2lnccny-ev1v7fszc

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Figura 5.5: Inicialització d'un Docker Swarm.

Aquesta ordre transforma la màquina actual en un node *manager*. Addicionalment, també ens torna un exemple de l'ordre que hem d'executar per a unir més màquines al *cluster*.

Per a continuar, només s'ha d'executar la ordre que s'obté al iniciar el *cluster* a cada un dels nodes que volem unir. Seguint amb el diagrama presentat (Figura 5.3) a l'inici d'aquest capítol, s'afegeixen dos nodes més.

Cal destacar què és important tindre cura amb les regles del tallafocs dels nodes. Es importar assegurar-se de que els tallafocs de cada una de les màquines permeten les comunicacions entre el *manager* i els *workers*.

Al final del procés, ja es disposa del *cluster*. La següent imatge mostra l'estat del desplegament en aquest punt:

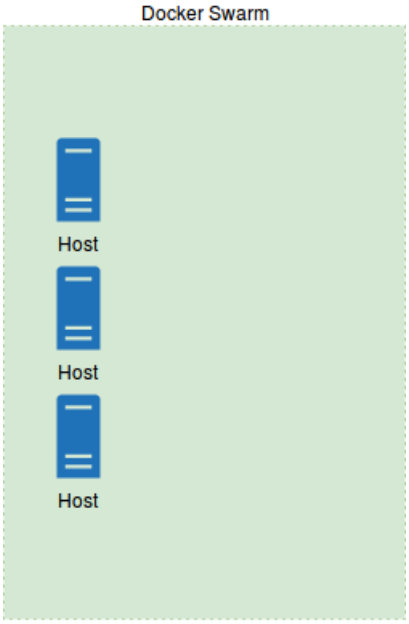

Figura 5.6: Muntatge del clúster.

En aquest punt, el *cluster* està preparat per a executar els microserveis. A continuació, cal desplegar els serveis necessaris per al funcionament.

5.2.3 Creació d'un registre d'imatges

Una part fonamental per al funcionament del *cluster* es poder obtindre les imatges dels microserveis. Per a que els nodes tinguin accés a les imatges dels microserveis, cal posar en marxa un servei *registry*.

Muntar un registre privat per a ser utilitzar localment resulta senzill. Ara bé, configurar-lo per a donar suport a altres nodes no resulta trivial.

Per a poder aconseguir que els nodes es puguin comunicar amb el servei de registre, cal configurar-lo amb seguretat usant *TLS* (*Transport Layer Security*).

Aquesta configuració s'ha fet seguint els següents passos:

- **Generar un certificat autosignat.** Mitjançant la ferramenta *openssl*, es genera un certificat autosignat. Un cop s'ha generat, cal distribuir aquest certificat als nodes que participen del *cluster*.

- **Configurar els nodes per a acceptar un registre insegur.** Conseqüència d'utilitzar un registre autosignat, es que Docker no el considera segur. Per a que els nodes puguin utilitzar-lo, s'ha de configurar cada node. Aquesta configuració s'indica a cada node que ha de confiar en el registre que s'ha desplegat.

Superats aquest petits inconvenients, només queda posar en marxa el registre i pujar les imatges dels serveis.

Per a posar en marxa aquest servei, executem:

```
docker run -d  --restart=always  --name registry  -v /certs:/certs \
-e REGISTRY_HTTP_ADDR=0.0.0.0:443  -e   \
REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
-e REGISTRY_HTTP_TLS_KEY=/certs/domain.key  -p 443:443 \
registry:2
```

Listing 5.5: Ordre per a posar en martxa el registre d'imatges.

Amb aquesta ordre es copien al contenidor del registre els certificats que s'han generat. A més es defineixen variables d'entorn que defineixen on s'hi troben els certificats i quin es el port del registre. Finalment, es mapeja el port del registre al port de la màquina actual.

Per a poder treballar amb comoditat, el *host* on es troba el registre s'ha anomenat *myregistry.com*.

Ara cal pujar les imatges al servei. Això es fa executant:

```
docker push myregistry.com:443/imatge
```

Listing 5.6: Ordre per a pujar una imatge al registre d'imatges.

Després d'aquest pas, la imatge ja està disponible en el registre. A partir d'ara, els nodes poden descarregar les imatges del registre. Això permet afegir màquines al *cluster* sense necessitat de preocupar-se per el aprovisionament de les mateixes.

Quan una màquina del *cluster* rep l'ordre del node *manager* d'executar una imatge, Docker s'encarrega de descarregar-la del registre apropiat si el node actual no la té a disposició localment.

5.2.4 Desplegament de l'aplicació sobre el clúster

L'últim pas per a posar en marxa l'aplicació es crear un Docker Stack. Com ja s'ha comentat anteriorment, un Docker Stack defineix una serie de serveis (imatges Docker) relacionats entre sí, configuració de xarxa interna al *cluster*, número de rèpliques per a cada servei...

Abans de continuar, cal refrescar els microserveis que s'han de desplegar:

Microservei	Funcionalitat	API
road_alerts	Es subscriu a una serie de road-segments. Quan es publica un accident en un road-segment, publica una alerta per la cua d'alertes.	- Consultar road-segments gestionats - Afegir road-segment - Llevar road-segment
visualizer	Mostra l'estat d'un Docker Swarm	- Mostrar l'estat del Swarm actual
road_manager	S'encarrega de repartir els road-segments entre els microserveis d'alertes disponibles.	- Consultar assignació de nodes i road-segments - Donar d'alta node

Taula 5.4: Microserveis a desplegar.

Ara, cal definir un fitxer *docker-compose* amb tots els microserveis que es volen desplegar. A continuació s'adjunta el fitxer que s'ha configurat:

```
version: "3"
services:
  alerts:
    image: myregistry.com:443/road_alerts:1.3
    command: /autorun/start_docker 8182 tcp://tambori.dsic.upv.es
        :1883 ina/2018/traffic "CV30;CV40;CV50" http
        ://192.168.232.13:8888/instance
    ports:
      - "8182:8182"
    deploy:
      replicas: 0
      restart_policy:
        condition: on-failure
    networks:
      - overlay
  roadmanager:
    image: myregistry.com:443/road_manager:1.1
    command: node server.js
    ports:
      - "8888:3000"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - overlay
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
```

```
27        - "8080:8080"
28      volumes:
29        - "/var/run/docker.sock:/var/run/docker.sock"
30      deploy:
31        placement:
32          constraints: [node.role == manager]
33      networks:
34        - overlay
35  networks:
36    overlay:
37       driver: overlay
```

Listing 5.7: Dockerfile del gestor de cúes.

Aquest fitxer conté tots els microserveis que formen part del *Stack*. Per a cadascun d'ells es defineix el nombre de rèpliques a desplegar, la xarxa que han d'utilitzar i en quin node tipus de node s'han d'executar.

Sobre aquest fitxer cal destacar el següent:

- **Microservei gestor de cues.** La implementació actual del gestor de cues no està preparada per a tindre més de un gestor de cues al mateix *cluster*. Per tant, només es desplegarà una rèplica. Tanmateix, tampoc es compta amb un servei per a que els nodes puguin descobrir on es troba el servei gestor de cues. Per a superar això, es proporcionarà als nodes l'adreça externa del gestor, es a dir, l'adreça que connecta la xarxa externa amb el Stack.

- **Microserveis alertes inicialment parats.** Inicialment no es llança cap instància dels microserveis d'alertes. Això permet garantir que el servei gestor d'alertes es posa en marxa abans que els microserveis d'alertes. Per a posar en marxa microserveis d'alertes. Serà suficient amb donar l'ordre d'escalar el servei des d'el node *manager*.

- **Xarxa.** Tots els microserveis han de formar part de la mateixa xarxa per a garantir la seua comunicació. A més, aquesta xarxa ha de ser de tipus *overlay* per a permetre la comunicació entre els diferents nodes del *cluster*.

Un cop tenim definit el nostre fitxer, per a desplegar l'aplicació s'ha d'executar l'ordre:

```
1 docker stack deploy -c docker-compose.yml TFM
```

Listing 5.8: Ordre per a desplegar l'aplicació al clúster.

Un cop s'executa l'ordre, el node mànager s'encarrega de repartir el treball entre els nodes disponibles del *cluster*. En aquest punt els nodes que ho necessiten descarreguen les imatges del registre que s'ha posat en marxa en l'apartat anterior.

L'estat del *cluster* un cop s'ha executat l'ordre es el següent:

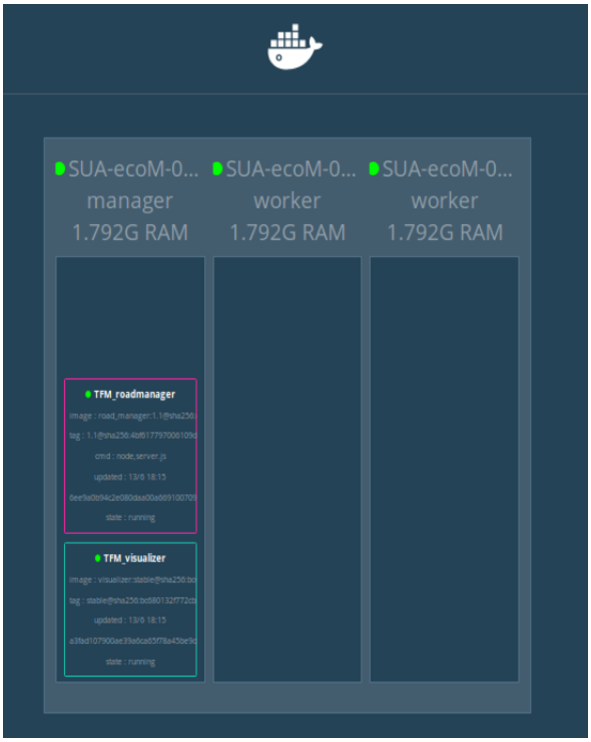

Figura 5.7: Servei de visualització.

Com es pot observar, s'han desplegat els serveis de visualització i el gestor de carreteres al node *manager*, tal i com s'havia definit. A continuació, només queda començar a afegir instàncies del microservei d'alertes.

5.2.5 Escalat de microserveis d'alertes.

Per a concloure amb el primer prototip, només queda afegir dos instàncies de microserveis d'alertes. En aquest punt, el servei d'alertes forma part del *Stack*, encara que no hi ha ninguna instància en marxa.

Per a afegir una nova instància, només s'ha d'indicar a Docker que es vol escalar el servei d'alertes. Això es fa amb la següent ordre:

```
1 docker service scale TFM_alerts=1
```

Listing 5.9: Ordre per a canviar el nombre d'instàncies del microservei d'alertes a 1.

Amb aquesta ordre, el node *manager* s'encarregarà de posar en marxa una instància del microservei d'alertes. El node *manager* també s'encarrega de seleccionar el

node encarregat d'executar la nova instància. A continuació es mostra l'estat del *cluster* en aquest punt:

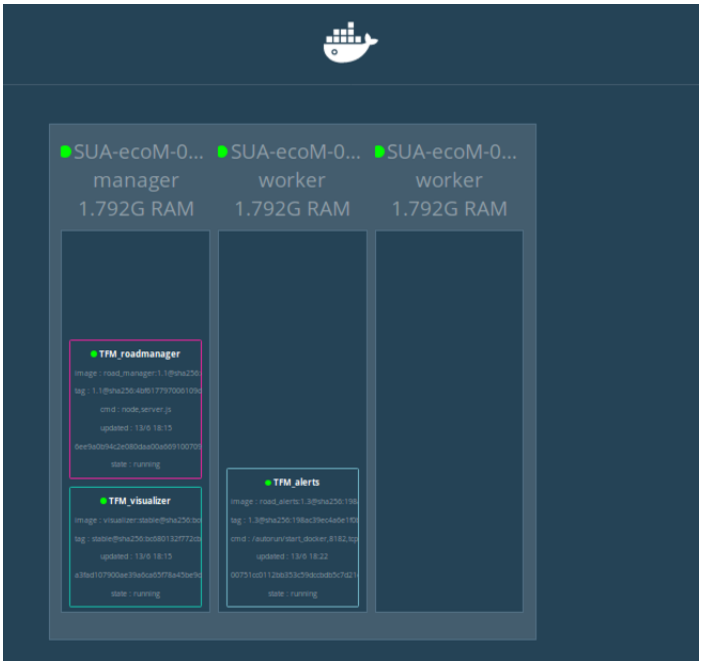

Figura 5.8: Desplegament d'un microservei.

S'observa com la nova instància s'ha assignat a un node diferent del *manager*.

Per una altra banda, d'acord amb el disseny, el nou node s'ha de comunicar amb el gestor de cues per a notificar-li la seua disponibilitat. Per a veure les comunicacions, observem els *logs* del servei de gestió de carreteres executant:

```
1 docker service logs TFM_roadmanager
```

Listing 5.10: Ordre per a observar els logs del servei de gestió de carreteres.

I el resultat que obtenim:

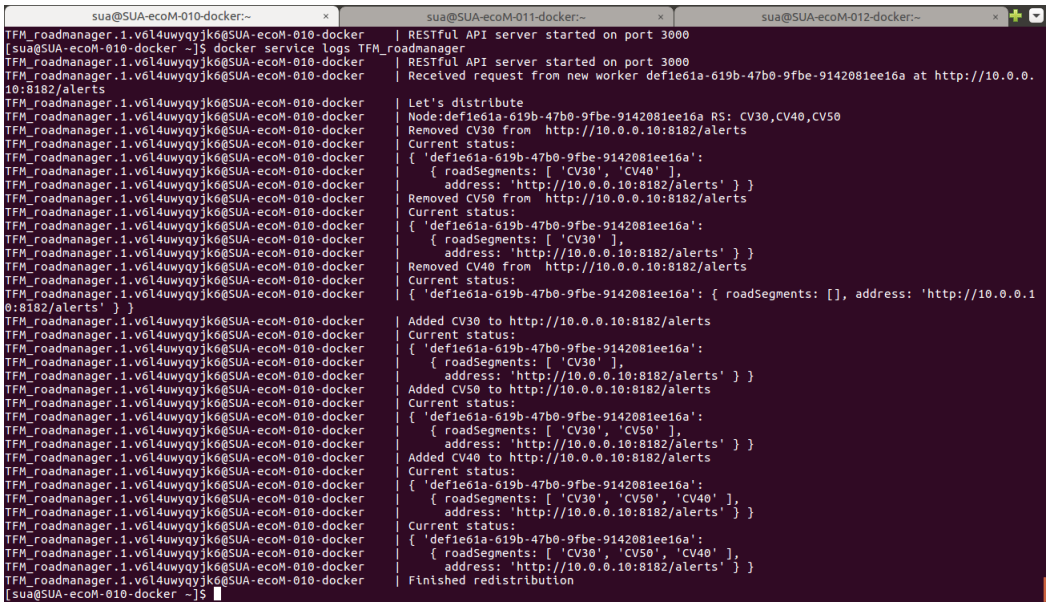

Figura 5.9: Logs del servei de gestió de carreteres.

Aquesta imatge mostra la informació que genera el servei de gestió de carreteres per la sortida estàndard. Si anem des del principi de l'ordre cap a baix, els missatges signifiquen el següent.

Primer, el servei indica que està escoltant a port 3000 (encara que aquest port s'ha redireccionat al port 8888 a la definició del *Stack* i és al 8888 al que realment accedeixen els microserveis). A continuació, es mostra un missatge que indica que s'ha rebut una sol·licitud d'un nou microservei d'alertes.

Acte seguit, s'executa l'algorisme de reorganització de les cues. Es mostren les accions realitzades, i després de cadascuna l'estat de tots els nodes coneguts i les cues associades. En aquest cas, com no hi han encara més instàncies amb qui repartir el treball, l'algorisme no té cap efecte.

Ara, s'afegeix una segona instància del microservei d'alertes, amb l'ordre:

```
1 docker service scale TFM_alerts=2
```

Listing 5.11: Ordre per a canviar el nombre d'instàncies del microservei d'alertes a 2.

L'estat del *cluster* ara ja es l'estat objectiu descrit a la Figura 5.3. Es pot observar al servei de visualització:

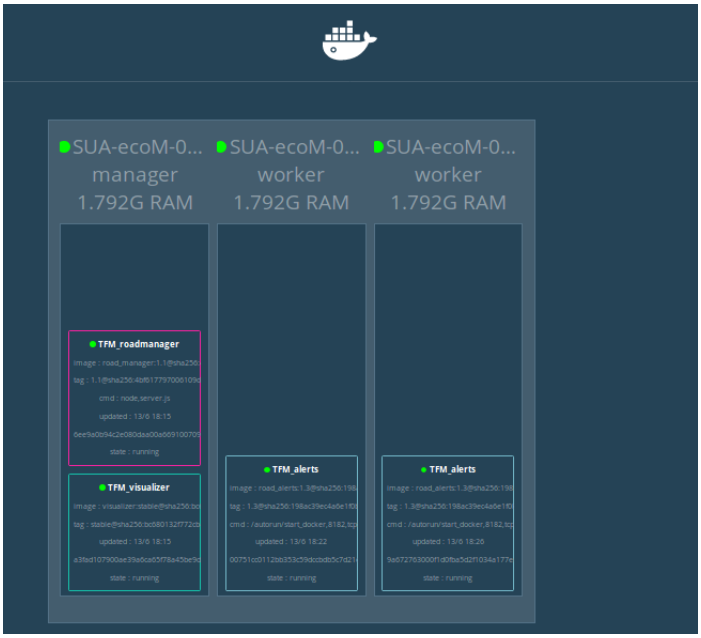

Figura 5.10: Estat final del clúster.

Per un altra banda, s'observen noves comunicacions al gestor de carreteres:

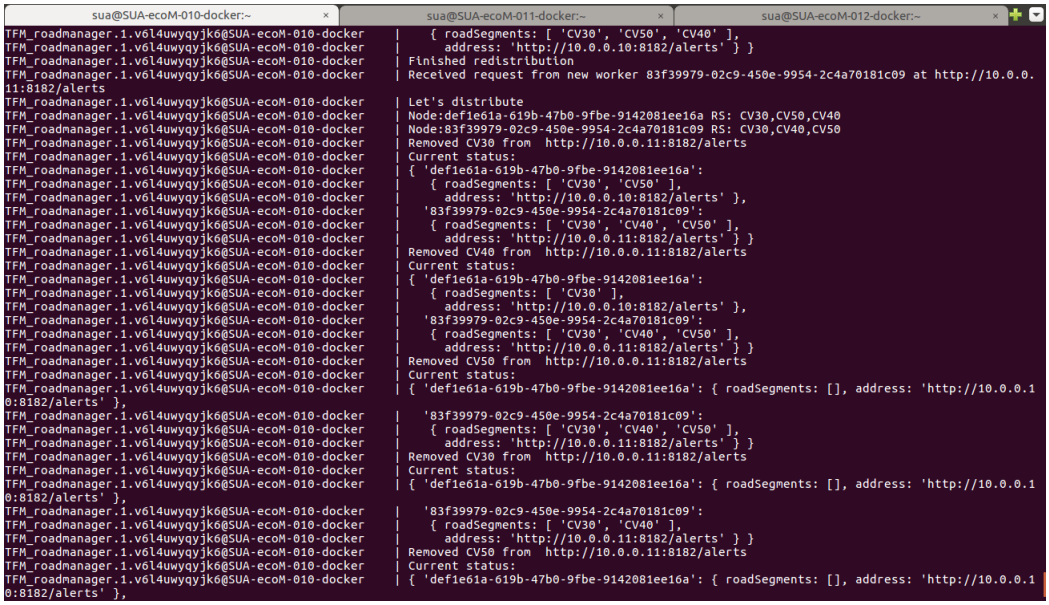

Figura 5.11: Logs ditribució (1).

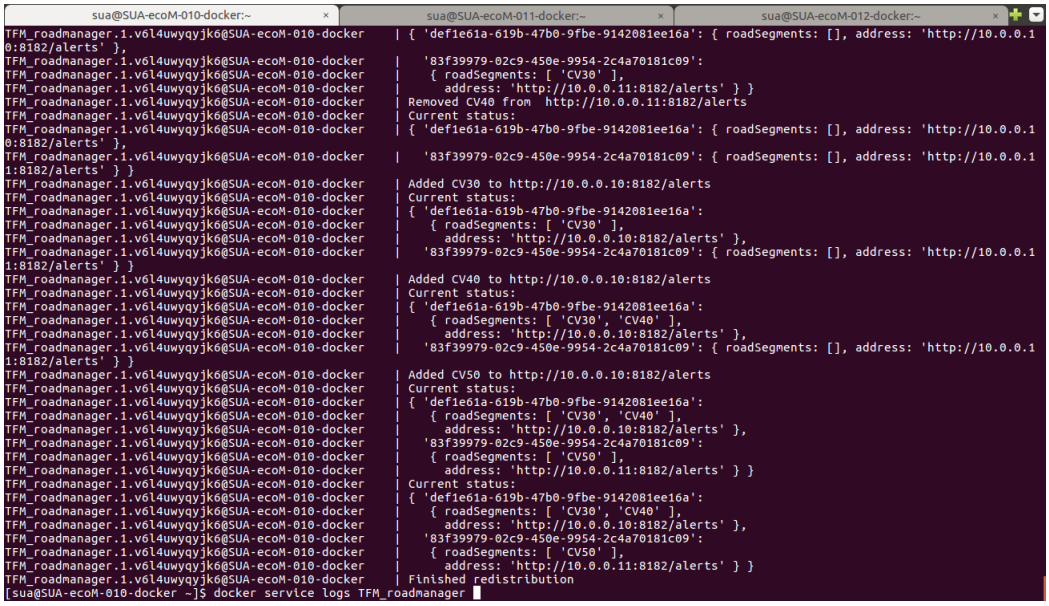

Figura 5.12: Logs distribució (2).

Com en el cas anterior, la nova instància s'ha posat en contacte amb el gestor de carreteres. Com que les noves instàncies es despleguen utilitzant l'ordre definida al fitxer docker-compose, aquesta nova instància informa al gestor de carreteres de que gestiona les tres mateixes carreteres que l'altra instància.

Els *logs* del gestor mostren primer que s'ha rebut una petició d'una nova instància. A continuació, es mostra l'estat que coneix el gestor, que son els dos nodes amb els segments de carretera que tenen assignades. Com ja s'ha comentat, els dos nodes tenen la mateixa llista de segments de carretera.

A continuació es posa en marxa l'algorisme de redistribució. A la Figura 5.12 podem veure els últims passos de l'algorisme. Al final de tot es mostra l'estat actual després de realitzar les accions de redistribució. El resultat es que s'han redistribuït els segments de carretera. Com hi han tres segments de carretera, dos s'han assignat a una instància i el restant, a l'altra.

5.3 Conclusions

En aquest capítol s'ha portat a terme el disseny proposat en el capítol anterior. S'ha portat a terme el desenvolupament del codi necessari i s'ha proposat un escenari de prova.

A continuació, s'han realitzar totes les accions necessàries per a poder muntar el escenari. Finalment, s'ha provat l'escenari i s'ha demostrat la funcionalitat, tant a nivell de microserveis com a nivell d'infraestructura.

Capítol 6

Conclusions

6.1 Treball realitzat

El treball realitzat per a avançar en la refactorització de l'aplicació inicial cap a una solució basada en microserveis es pot resumir en els següents punts:

- **Estudi i valoració de la tecnologia a utilitzar.** El punt de partida és la transformació d'una aplicació monolítica en una aplicació basada en microserveis. Per a poder començar amb aquesta tasca, s'ha fet un estudi de les ferramentes disponibles que donen suport a aquesta arquitectura. Finalment, s'ha fet un anàlisis de les alternatives i s'ha elegit la que millor s'adapta a les característiques del projecte.

- **Disseny de la solució.** Abans de començar a implementar, s'ha fet un disseny per a donar suport a l'arquitectura de microserveis. En aquest disseny s'han tingut en compte els aspectes particulars de l'aplicació objectiu, les funcionalitats de Docker, i les interaccions entre els microserveis.

- **Extracció d'un microservei de l'aplicació inicial.** Si bé no s'ha fet la transformació completa de l'aplicació en un conjunt de microserveis; si que s'ha transformat una funcionalitat concreta en un microservei. Per a fer això s'han implementat les adaptacions necessàries al codi original per a obtindre un microservei que pugui interactuar amb altres microserveis. Finalment, s'ha empaquetat el microservei en una imatge Docker per tal de facilitar el desplegament del microservei en qualsevol entorn.

  Tanmateix, aquest procés marca la pauta a seguir en les altres funcionalitats de l'aplicació que siguin candidates a ser convertides en un microservei.

- **Implementació d'un microservei per a repartir la càrrega.** Un aspecte imprescindible de la transformació de l'aplicació inicial. El microservei implementat s'encarrega de repartir la càrrega entre les instàncies disponibles cada cop que s'afegeix una nova instància. Sense aquest microservei no s'obté ningun avantatge amb l'ús de microserveis.
- **Desplegament d'un clúster per a executar els microserveis.** S'ha creat un *cluster* amb Docker Swarm per a proporcionar les màquines que executen els microserveis. A més a més, aquest *cluster* permet gestionar el nombre de rèpliques de cada microservei, monitoritzar l'activitat dels nodes, proporcionar connectivitat interna als microserveis que formen part del *cluster*, entre altres funcions.

6.2 Problemes trobats

6.2.1 Connectivitat del clúster

Tot el disseny d'aquest projecte es fonamenta en les comunicacions entre tots els components. Per a poder dissenyar un prototip efectiu, ha sigut necessari estudiar amb deteniment tot el tractament que fa Docker de les xarxes.

Un dels primer problemes que va sorgir va ser què no es podien afegir nodes al *cluster*. Aquest problema es devia a que per comunicar-se, Docker utilitza uns ports específics. En canvi, les màquines on s'havia desplegat el *cluster* estaven configurades per a no acceptar tràfic més enllà d'alguns ports coneguts.

La solució passa simplement per configurar el tallafocs per a permetre les connexions entre el *manager* i els nous nodes [1] .

Cal destacar que els ports que utilitza Docker per a gestionar la infraestructura del *cluster* (afegir i eliminar nodes) s'utilitzen exclusivament per a aquesta tasca.(Taula 6.1)

Taula 6.1: Resum de ports utilitzats per Docker

Port	Protocol	Descripció
2377	TCP	Comunicacions de gestió del clúster
7946	TCP	Comunicacions entre nodes
7946	UDP	Comunicacions entre nodes
4789	UDP	Xarxes de tipus overlay

[1] En aquest cas, el software encarregat de gestionar el tallafocs era iptables. La configuració depèn del software que utilitzi cada màquina en particular.

Més endavant, un cop ja s'havia aconseguit desplegar el *cluster*, es va desplegar l'aplicació (el Docker Stack). En aquest punt van tornar a sorgir problemes de connectivitat. Els contenidors que formaven part del Stack només podien comunicar-se si es trobaven dins del mateix node.

Aquest problema es devia a que la xarxa *overlay* no s'estava configurant correctament. Un altre cop, el problema estava en que les màquines no acceptaven el tràfic al port específic que utilitza Docker per a configurar les xarxes *overlay* (Taula 6.1). Com a conseqüència, Docker no podia comunicar-se amb els nodes i no podia muntar la xarxa interna que comunica tots els nodes del *cluster*.

6.2.2 Registre d'imatges

El registre d'imatges permet aprovisionar automàticament a tots els nodes amb les imatges necessàries per a executar els microserveis. Ara bé, a l'hora de muntar un registre privat han sorgit algunes dificultats.

La primera aproximació va ser muntar un registre sense ningun tipus de seguretat. Aquesta aproximació funciona correctament de manera local. En canvi, quan es vol accedir el registre des de altres màquines, la cosa es complica una mica.

Per defecte, Docker no confia en els registres insegurs. Per tant per a poder muntar aquesta part va ser necessari configurar el registre amb una seguretat mínima. Això s'ha aconseguit mitjançant certificats auto signats, que s'han de distribuir a cadascun dels nodes. Aquesta part va resultar un punt bloquejant ja que la configuració no es senzilla, i s'han de tindre nocions bàsiques del funcionament de comunicacions segures amb TLS.

6.2.3 Cues i replicació

Un dels problemes que sorgeixen més prematurament es el problema de les cues. Quan es fa el primer disseny conceptual, encara sense haver decidit cap tecnologia, el problema de les cues no es fa visible. Això es deu a que en un principi, el disseny es fa en base a l'arquitectura de microserveis, encara sense conèixer el funcionament de l'aplicació a transformar.

Un cop es coneixen els detalls de l'aplicació, es fa evident que s'ha de definir un mecanisme per a poder repartir les subscripcions a les cues i evitar que dues rèpliques d'un mateix microservei atenguin els mateixos missatges.

Aquest problema es un problema natural en la transformació d'una aplicació monolítica a l'arquitectura de serveis, i que depèn de cada cas específic. Des del moment en que es detecta aquesta necessitat, es tracta com a un requeriment més del disseny, i passa a formar part de la solució.

6.3 Aportacions

6.4 Personals

A nivell personal aquest projecte m'ha ajudat a iniciar-me en el món de l'arquitectura de microserveis. Gràcies a aquest exercici, he pogut comprendre els fonaments dels microserveis, els punts claus de l'arquitectura, i els avantatges i reptes que comporta.

Per un altra banda, a nivell de tecnologia, m'ha permès conèixer una gran quantitat de tecnologies que permeten aconseguir el repte de desenvolupar una arquitectura basada en microserveis. Cal destacar els mecanismes de comunicació i l'us de Docker com a infraestructura.

6.5 Aplicació

Les aportacions específiques a l'aplicació son l'avanç en el canvi a una arquitectura de microserveis. Per una part s'ha convertit una part del codi en un microservei. Per l'altra part, s'ha desplegat un *cluster* que permetrà executar els futurs microserveis que s'extrauran de l'aplicació. A més a més, es proporciona una primera solució per a gestionar la problemàtica de la interacció amb les cues.

6.6 Acadèmiques

Pel que fa a les aportacions acadèmiques, aquest treball fa un exercici complet sobre com transformar una aplicació a l'arquitectura de microserveis. Així, es pot veure el procés que va des de buscar les tecnologies disponibles fins a desenvolupar les adaptacions necessàries. En concret, el cas del microservei gestor de cues és un exemple de com la transformació cap a una arquitectura de microserveis pot variar segons el cas específic.

6.7 Ampliacions futures

6.7.1 Escalabilitat a les cues

Una de les ampliacions futures més importants es abordar el problema de l'escalabilitat a les cues. Malgrat que com a resultat hem obtingut una aplicació preparada per a ser escalada i créixer en el nombre d'usuaris, les cues no tenen

ningun mecanisme per a atendre una demanda major. Això pot convertir-se en un coll de botella i per tant es una ampliació futura important.

6.7.2 Millorar el gestor de cues

Si bé la implementació que s'ha fet s'encarrega de repartir la càrrega, té una principal limitació: el gestor no es capaç de detectar si els nodes estan actius o no, o si estan saturats o ociosos.

El següent pas en aquesta aspecte seria implementar funcionalitat per a monitoritzar l'activitat dels nodes als que se'ls hi han assignat les cues. Així el microservei podria prendre decisions com repartir les cues d'un node caigut entre la resta, o reduir el nombre de cues gestionades per un node que estigui sobrecarregat.

6.7.3 Ocultació de les APIs dels microserveis d'alertes

Tal i com s'ha fet el disseny i la implementació, els microserveis d'alertes i el microservei gestor de cues es comuniquen per a repartir la càrrega.

Quan es desplega un microservei d'alertes, se li indiquen els segments de carretera que ha de gestionar. A continuació, aquest li ho comunica al gestor de carreteres per a indicar que es troba disponible i quins són els segments de carretera que pretén gestionar.

Aquesta aproximació pot canviar-se per una aproximació més eficient. Es poden desplegar els microserveis d'alertes sense cap segment assignat i deixar que el gestor de carreteres s'encarregui de centralitzar les carreteres a gestionar.

Per a fer això, quan es vol gestionar un segment de carretera nou se li comunica al gestor de carreteres (enlloc de a un microservei d'alertes). El gestor de carreteres s'ha d'encarregar de decidir a quins dels microserveis d'alertes disponibles li assigna el segment.

Aquesta aproximació permet centralitzar la gestió de segments de carreteres en un microservei. De la mateixa forma, les accions d'eliminar segments de carretera o consultar els gestionats podrien centralitzar-se en aquest microservei. A més a més, tal i com s'ha fet el disseny, aquest canvi no implica modificar els microserveis d'alertes, només el gestor de carreteres.

6.7.4 Continuar amb la refactorització

En aquest projecte només s'ha convertit a microserveis una part de la funcionalitat de partida. Per tant, s'ha de continuar amb aquesta tasca, per a transformar la resta de funcionalitats en microserveis.

Això si, s'ha d'avaluar sempre per a cada cas si convé o no transformar una part de la funcionalitat en un microservei.

6.7.5 Aspectes de seguretat

En aquest projecte no s'han considerat els aspectes de seguretat, per a limitar la extensió del mateix i no excedir el temps del que es disposa.

Així, cal fer un anàlisis de les implicacions de seguretat que tenen tots els components. Cal avaluar les interaccions entre microserveis, l'accessibilitat externa a cada un dels components i de les APIs que exposen, etc...

Bibliografia

[1] Chris Richardson. *Microservices a definition of this new architectural term*. 2015. URL: https://www.nginx.com/blog/introduction-to-microservices/.

[2] Martin Fowler. *Introduction to Microservices*. 2015. URL: https://www.martinfowler.com/articles/microservices.html.

[3] Kubernetes. *Kubernetes Basics*. 2018. URL: https://kubernetes.io/docs/tutorials/.

[4] Docker. *Get started with Docker*. 2018. URL: https://docs.docker.com/get-started/.

[5] Sam Newman. *Building Microservices*. 1st. O'Reilly Media, Inc., 2015. ISBN: 1491950358, 9781491950357.