Desenvolupament d’una Plataforma SaaS Autoescalable i Automatitzada amb Arquitectura Kubernetes

Resum

Aquest projecte s'ha centrat en el desenvolupament d'una plataforma SaaS (Software as a Service) escalable i modular, utilitzant Kubernetes com a base per gestionar entorns personalitzats per a cada client. L'objectiu ha estat crear una solució flexible que permetés desplegar entorns independents per a cada usuari, garantint la seguretat i l'optimització dels recursos de manera automàtica i eficient.

S'ha dissenyat una arquitectura que permet la creació d'entorns aïllats per als clients, assegurant la independència de les dades i serveis. Kubernetes ha estat la eina clau per garantir l'escalabilitat automàtica de la infraestructura, adaptant-se a la demanda del sistema en temps real.

El projecte ha inclòs tres casos d'ús concrets:

1. **Universitat**: Creació d'entorns de desenvolupament aïllats per als estudiants durant exàmens pràctics, garantint la seguretat i rendiment en moments de càrrega elevada.
2. **Hospital**: Aïllament de dades i serveis entre diversos hospitals en una única plataforma, per complir amb normatives de privacitat i seguretat.
3. **Startup de contingut multimèdia**: Gestió de trànsit massiu durant el llançament de contingut popular, assegurant una alta disponibilitat i un rendiment òptim a nivell global.

Els resultats obtinguts han mostrat que Kubernetes és una solució potent per desenvolupar sistemes autoescalables, modulars i personalitzats, amb un control eficient dels recursos. Aquest projecte ha destacat la capacitat d'adaptar-se a necessitats específiques i ha obert la porta a futurs desenvolupaments més complexos en entorns SaaS.

Resumen

Este proyecto se ha centrado en el desarrollo de una plataforma SaaS (Software as a Service) escalable y modular, utilizando Kubernetes como base para gestionar entornos personalizados para cada cliente. El objetivo ha sido crear una solución flexible que permita desplegar entornos independientes para cada usuario, garantizando la seguridad y optimización de los recursos de manera automática y eficiente.

Se ha diseñado una arquitectura que permite la creación de entornos aislados para los clientes, asegurando la independencia de los datos y servicios. Kubernetes ha sido la herramienta clave para garantizar la escalabilidad automática de la infraestructura, adaptándose a la demanda del sistema en tiempo real.

El proyecto ha incluido tres casos de uso concretos:

1. **Universidad**: Creación de entornos de desarrollo aislados para los estudiantes durante los exámenes prácticos, asegurando la seguridad y el rendimiento durante momentos de carga elevada.
2. **Hospital**: Aislamiento de datos y servicios entre varios hospitales en una única plataforma, para cumplir con normativas de privacidad y seguridad.

3. **Startup de contenido multimedia**: Gestión de tráfico masivo durante el lanzamiento de contenido popular, asegurando alta disponibilidad y rendimiento óptimo a nivel global.

Los resultados obtenidos han mostrado que Kubernetes es una solución potente para desarrollar sistemas autoescalables, modulares y personalizados, con un control eficiente de los recursos. Este proyecto ha destacado la capacidad de adaptarse a necesidades específicas y ha abierto la puerta a futuros desarrollos más complejos en entornos SaaS.

Abstract

This project focused on the development of a scalable and modular SaaS (Software as a Service) platform using Kubernetes as the foundation to manage customized environments for each client. The goal was to create a flexible solution that allows the deployment of independent environments for each user, ensuring security and resource optimization automatically and efficiently.

An architecture was designed that allows the creation of isolated environments for clients, ensuring the independence of data and services. Kubernetes was the key tool to ensure the automatic scalability of the infrastructure, adapting to the system's workload in real-time.

The project included three specific use cases:

1. **University**: Creation of isolated development environments for students during practical exams, ensuring security and performance during peak load periods.
2. **Hospital**: Isolation of data and services across multiple hospitals on a single platform to comply with privacy and security regulations.
3. **Multimedia startup**: Managing massive traffic during popular content launches, ensuring high availability and optimal performance globally.

The results showed that Kubernetes is a powerful solution for developing auto-scaling, modular, and customized systems with efficient resource management. This project highlighted its ability to adapt to specific needs and opened the door for more complex future developments in SaaS environments.

1. Introducció i motivació

1.1 Context del problema

En un moment on la tecnologia avança a una velocitat exponencial, les organitzacions es veuen abocades a integrar un nombre creixent de sistemes per satisfer les necessitats cada cop més complexes dels seus operatius. Aquesta tendència genera nous reptes, especialment pel que fa a la gestió de les infraestructures i la sostenibilitat econòmica. L'augment de les demandes tecnològiques obliga les empreses i institucions a gestionar sistemes d'entorns cada vegada més complexes, on la coordinació entre les diferents aplicacions i serveis és fonamental per al bon funcionament global del sistema. A mesura que els serveis es multipliquen, també ho fa la necessitat de garantir que aquestes infraestructures siguin segures, escalables i eficients.

En aquest context, els models basats en infraestructures SaaS (Software as a Service) han guanyat molta popularitat. Aquests models permeten que una mateixa infraestructura faci servir múltiples entorns aïllats, compartint recursos de manera eficient i proporcionant una gestió centralitzada de les aplicacions. La capacitat de replicar entorns de manera ràpida i segura, com és el cas dels entorns en Kubernetes, facilita la gestió d'entorns múltiples per a grans volums d'usuaris, millorant la flexibilitat operativa i reduint els costos associats.

Tanmateix, aquest tipus de models presenta desafiaments específics. Per exemple, l'aïllament adequat dels entorns dins d'una mateixa infraestructura és una necessitat crítica. Sense aquest aïllament, les aplicacions poden generar incidències imprevistes, amb riscos de contaminació de dades, vulnerabilitats en la seguretat, i fins i tot ciberatacs que poden comprometre les dades dels usuaris. En sectors com l'educació o la sanitat, on la privacitat i la seguretat de la informació són primordials, el compliment de normatives de protecció de dades (com el RGPD a Europa) és una prioritat innegociable. Per això, es requereix una solució tecnològica que permeti no només la creació d'entorns replicables, sinó també l'aïllament de dades i serveis per a cada entitat, per garantir que la seguretat i el compliment de normatives es mantinguin intactes.

A més, l'eficiència en la gestió dels recursos és un altre aspecte essencial a tenir en compte. Els costos derivats de l'ús de recursos en entorns de gran escala poden ser molt elevats, especialment quan aquests entorns es mantenen actius de manera continuada per a milers d'usuaris. Això pot generar un gran impacte econòmic si els recursos no es gestionen adequadament. Per això, és crucial dissenyar una infraestructura que sigui autoescalable i capaç de reduir-se en dimensió quan la demanda baixa, de manera que els costos operatius siguin tan eficients com sigui possible. L'escalabilitat no només ha de ser possible manualment, sinó que ha de ser automatitzada, per estalviar temps i evitar errors humans, que en entorns complexos podrien generar una pèrdua de rendiment o seguretat.

1.2 Estat de l'art

Per al desenvolupament d'aquest projecte, vaig realitzar una cerca exhaustiva i anàlisi comparativa de diversos sistemes i eines disponibles per gestionar entorns aïllats, escalabilitat automatitzada i gestió centralitzada de la infraestructura. L'objectiu era trobar la millor solució per a un sistema inicial amb requisits lleugers, però que tingués la capacitat de escalar a gran escala de forma automatitzada i centralitzada, tot mantenint la compatibilitat amb diferents entorns, com servidors físics, entorns cloud i CPDs (Centres de Processament de Dades). En aquesta recerca, vaig explorar diverses opcions, incloent Kubernetes, Docker, Docker Compose, AWS Lambda, sistemes de màquines virtuals, Ansible, Jenkins, entre altres.

Kubernetes

Kubernetes es va identificar com una de les solucions més robustes per a la gestió de sistemes a gran escala, gràcies a la seva capacitat d'autoescalatge, la gestió de contenidors i l'aïllament de recursos. Permet gestionar múltiples entorns de manera eficient, amb una orquestració automàtica dels recursos i la possibilitat de desplegar aplicacions en diversos nodos dins d'un clúster distribuït. Aquesta solució es destaca per la seva capacitat d'adaptar-se tant a servidors físics com a entorns cloud o CPDs, fent-la extremadament flexible i compatible amb diferents tipus d'infraestructura. Kubernetes permet l'orquestració automatitzada de serveis i el desplegament eficient d'aplicacions i entorns, facilitant la gestió centralitzada.

L'aspecte més destacat de Kubernetes és la seva escala dinàmica mitjançant eines com el Horizontal Pod Autoscaler (HPA) i Cluster Autoscaler, que permeten ajustar els recursos en temps real segons la demanda. La seva compatibilitat amb diversos entorns de desplegament és un altre avantatge important, ja que Kubernetes permet executar-se tant en núvols públics com privats, així com en màquines físiques locals.

Avantatges:

- Escalabilitat automàtica a mesura que augmenta la càrrega de treball.
- Aïllament d'entorns mitjançant namespaces, permetent separar i gestionar entorns independents.
- Compatibilitat amb múltiples entorns (clouds, CPDs, servidors físics).
- Gestió centralitzada a través de la consola de Kubernetes, evitant l'ús de múltiples eines.

Desavantatges:

- Curva d'aprenentatge alta per la configuració i gestió inicial.
- Requereix coneixements tècnics específics per gestionar i configurar adequadament l'orquestració del sistema.

Docker i Docker Compose

Docker i Docker Compose són eines que permeten crear contenidors per executar aplicacions de manera aïllada. Docker és útil per entorns petits i per a la creació de serveis individuals, però té limitacions pel que fa a la gestió d'escalabilitat en entorns de producció més grans. Per a entorns petits o per a ús local, Docker Compose és una eina pràctica per configurar múltiples contenidors, però no ofereix les capacitats d'orquestració automàtica i gestió de clústers que ofereix Kubernetes.

Aquesta solució seria apropiada per a entorns de desenvolupament o petits entorns de producció, però no proporciona una gestió automatitzada de la infraestructura ni una escalabilitat a gran escala com Kubernetes.

Avantatges:

- Simplicitat en la configuració i gestió de contenidors per a entorns petits.
- Portabilitat: permet executar els contenidors en qualsevol màquina amb Docker instal·lat.

Desavantatges:

- Escalabilitat limitada per a entorns de gran volum de trànsit.
- Falta de gestió d'infraestructura centralitzada i orquestració d'entorns a gran escala.

AWS Lambda (Serverless)

Els sistemes serverless, com AWS Lambda, permeten executar codi sense la necessitat de gestionar la infraestructura subjacient. Aquesta opció és especialment útil per a aplicacions d'escalabilitat automàtica i event-driven, on el codi s'executa com a resposta a esdeveniments (com peticions HTTP, canvis en base de dades, etc.). AWS Lambda es redueix al mínim la gestió d'infraestructura, permetent als desenvolupadors centrar-se només en el codi.

Tanmateix, AWS Lambda no és la solució ideal per a entorns multicontainers o aplicacions complexes amb múltiples components interdependents, ja que no ofereix un control complet sobre l'escalabilitat i la gestió de recursos com Kubernetes. A més, té un límit de temps d'execució per cada funció, la qual cosa pot ser un impediment per a aplicacions de llarga durada o amb requisits intensius de recursos.

Avantatges:

- Escalabilitat automàtica en funció de la càrrega de treball.
- Sense necessitat de gestionar servidors ni infraestructura, el que redueix la complexitat operativa.
- Costos de pagament per ús: es paga només pel temps d'execució de les funcions.

Desavantatges:

- Limitació de temps d'execució (15 minuts màxim per cada funció).

- Manca de control complet sobre la infraestructura i l'escalabilitat, cosa que pot ser un problema en sistemes complexos.
- No és ideal per a entorns amb múltiples components interdependents.

Sistemes de màquines Virtuals

Les màquines virtuals ofereixen una solució tradicional per executar aplicacions en entorns aïllats. A diferència dels contenidors, que comparteixen el sistema operatiu, cada màquina virtual té el seu propi sistema operatiu complet. Això proporciona un aïllament total, però també és més costós en termes de recursos (memòria, CPU i emmagatzematge).

Les màquines virtuals poden ser útils per a entorns on l'aïllament complet i la compatibilitat amb qualsevol sistema operatiu siguin necessàries, però la seva manca d'escalabilitat dinàmica i la gestió manual dels recursos poden limitar la seva utilització en entorns de gran escala.

Avantatges:

- Aïllament complet entre entorns.
- Compatibilitat amb diversos sistemes operatius i plataformes.

Desavantatges:

- Més pesades i menys eficients en termes d'ús de recursos.
- Escalabilitat limitada: afegir nous recursos o nodes manualment pot ser lent i costós.
- Compatibilitat de llibreries i sistemes complexa

Ansible i Jenkins (Sistemes de CI/CD)

Ansible i Jenkins són eines d'automatització que ajuden a gestionar la infraestructura i els processos de desplegament continu. Ansible és útil per a la configuració i gestió dels servidors, mentre que Jenkins s'utilitza per a la creació d'automatització de CI/CD, facilitant els processos d'integració i desplegament de les aplicacions.

Tot i que aquestes eines són molt poderoses per automatitzar tasques, no proporcionen orquestració ni gestió d'escalabilitat com Kubernetes. Poden complementar altres sistemes com Docker o Kubernetes per automatitzar processos, però no gestionen la infraestructura de manera eficient en escenaris de gran escala.

Avantatges:

- Automatització de processos de desplegament i gestió de configuració.
- Integració amb altres eines per crear un flux de treball complet.

Desavantatges:

- No són una solució d'orquestració ni de gestió d'escalabilitat.
- Necessiten ser complementades amb altres eines, com Docker o Kubernetes, per gestionar la infraestructura a gran escala.

1.3 Motivació del projecte

La motivació d'aquest projecte rau en la meva experiència com a CTO d'una empresa de ciberseguretat, on des de fa més de 4 anys he estat involucrat en el desenvolupament de productes innovadors utilitzant tecnologies de ciberseguretat de codi obert i orientades a la creació de solucions eficients i escalables. Quan vaig entrar a l'empresa amb les pràctiques de la UB, la meva tasca es limitava a proporcionar serveis de seguretat a petites empreses, amb un equip tècnic de només dues persones, jo inclosa. Al poc temps de començar, vaig rebre la responsabilitat de dissenyar i començar el desenvolupament una plataforma de monitoratge amb intel·ligència artificial, capaç de detectar amenaces i adaptar-se al comportament d'una infraestructura específica.

En aquell moment, no tenia experiència prèvia en el desplegament de tecnologies a gran escala, ni en la unificació de sistemes tecnològics en una plataforma professional orientada a les necessitats d'un client empresarial. No havia realitzar cap projecte fora dels realitzats a la Universitat. L'empresa era petita i disposava de recursos limitats, fet que va marcar l'inici del meu procés de reflexió per identificar la millor solució. Després d'analitzar les diferents alternatives del mercat, vaig concloure que no tenia sentit desenvolupar una solució des de zero per competir amb les grans empreses del sector, que ja disposaven d'eines robustes i completament desenvolupades.

Així, vaig decidir utilitzar tecnologies de codi obert, que poguessin interconnectar-se de manera flexible per unificar les seves funcionalitats en una única plataforma SaaS, adaptada a les necessitats dels clients. Aquesta plataforma havia de ser replicable per cada client, mantenint una base comuna però permetent la personalització per a cada un, garantint així l'escalabilitat i l'optimització dels costos, factors crucials en una empresa petita però innovadora com la nostra.

L'elecció de Kubernetes com a tecnologia clau va ser fonamental per resoldre aquests reptes. Kubernetes permet orquestrar contenidors i gestionar múltiples entorns replicables de manera eficient, proporcionant una plataforma escalable i automatitzable que facilita l'optimització dels recursos en un model SaaS. A més, Kubernetes integra sistemes de computació cloud, CI/CD i programació declarativa mitjançant YAML, que són imprescindibles per un desplegament eficient i per a la integració de diverses tecnologies dins d'una sola plataforma. Aquest va ser el punt de partida per al meu Treball Final de Grau, on vaig decidir explorar tres casos d'ús, tots ells compartint un nexe comú en l'arquitectura d'entorns automatitzables i autoescalables que Kubernetes permet, i centrat en la capacitat de l'arquitectura per complir les necessitats no funcionals i optimitzar els costos operatius.

Aquest projecte és, per tant, una adaptació directa de la feina que vaig realitzar a l'empresa, on la meva tasca com a CTO és impulsar la creació de solucions innovadores i escalables, alineant les tecnologies utilitzades amb les necessitats reals del mercat i assegurant la sostenibilitat econòmica i operativa de les nostres solucions. A través d'aquest TFG, he volgut aprofundir en com l'arquitectura basada en Kubernetes pot ser utilitzada no només per desenvolupar solucions específiques, sinó per proporcionar un marc general que permeti l'escalabilitat, l'optimització de costos i l'automatització dels processos en entorns empresarials complexos, des del desenvolupament fins a la implementació.

2. Objectius generals del projecte

O1: Desenvolupar una plataforma SaaS basada en una arquitectura Kubernetes per gestionar múltiples entorns replicables per client

- Crear una solució modular que permeti desplegar entorns personalitzats per a cada client amb configuracions específiques adaptades a les seves necessitats.
- Utilitzar namespaces de Kubernetes per aïllar cada entorn de client, garantint la seguretat i independència dels seus recursos i dades.
- Dissenyar plantilles YAML reutilitzables per al desplegament ràpid i consistent de serveis com aplicacions web, bases de dades i altres components necessaris per a cada client.
- Facilitar la replicació i escalabilitat dels entorns per integrar nous clients o projectes sense interrompre els serveis existents.

O2: Implementar una infraestructura autoescalable i automatitzada que optimitzi la gestió de recursos mitjançant Kubernetes i CI/CD

- Configurar el Horizontal Pod Autoscaler (HPA) per ajustar dinàmicament el nombre de pods en funció de la càrrega de treball i l'ús de recursos com CPU i memòria.
- Habilitar el Cluster Autoscaler perquè el sistema afegeixi o elimini nodes del clúster segons sigui necessari, optimitzant els costos i els recursos disponibles.
- Integrar CI/CD per implementar una metodologia GitOps, sincronitzant els manifestos emmagatzemats en un repositori Git amb l'estat actual del clúster.
- Automatitzar processos com el desplegament continu, les actualitzacions de serveis i els rollbacks en cas d'errors, assegurant la disponibilitat constant de la plataforma.

O3: Monitoritzar i optimitzar la solució amb eines com Prometheus i Grafana per garantir la fiabilitat i el control en temps real dels serveis desplegats

- Configurar Prometheus per recopilar mètriques detallades de rendiment del sistema, com ús de CPU, memòria, trànsit de xarxa i estat dels pods.
- Dissenyar dashboards personalitzats a Grafana per visualitzar el rendiment dels serveis i identificar possibles colls d'ampolla o anomalies.
- Implementar un sistema d'alertes automàtiques basades en mètriques clau per avisar de possibles problemes, com saturació de recursos o fallades en algun servei.
- Optimitzar la gestió de recursos a partir de l'anàlisi de dades de monitorització, ajustant configuracions per maximitzar l'eficiència i minimitzar els costos.

3. Introducció als casos d'ús

3.1. Exàmens finals de desenvolupament amb entorns aïllats

3.1.1. Descripció del cas d'ús

Aquest cas d'ús se centra en una universitat d'informàtica que necessita proporcionar entorns de desenvolupament personalitzats per a cada estudiant durant un examen pràctic. Els estudiants han de poder desplegar el seu codi i interactuar amb serveis com bases de dades o API's dins d'un entorn individual i controlat. Aquest tipus d'exàmens poden generar una càrrega molt elevada als sistemes de la universitat, ja que es requereixen recursos significatius de computació i xarxa en un curt període de temps.

El repte consisteix a garantir que:

1. Cada estudiant disposi d'un entorn de desenvolupament funcional i aïllat.
2. El sistema sigui capaç de gestionar un gran nombre d'entorns simultanis sense caigudes ni pèrdues de rendiment.
3. Els recursos utilitzats siguin eficaços i escalables, evitant malgastar capacitat durant períodes inactius.

3.1.2. Beneficis que aporta Kubernetes

1. **Escalabilitat automàtica amb HPA (Horizontal Pod Autoscaler):** Kubernetes ajusta dinàmicament el nombre de pods (unitats de treball) segons la càrrega del sistema, garantint que sempre hi hagi prou recursos per suportar el nombre d'estudiants actius. Això és especialment útil durant els pics de càrrega al començament d'un examen.
2. **Aïllament d'entorns amb namespaces:** Cada estudiant pot tenir el seu propi namespace dins del clúster de Kubernetes. Això assegura que cada entorn és totalment independent dels altres, evitant interferències i possibles errors derivats de compartir recursos.
3. **Gestió eficient de recursos:** Kubernetes permet assignar límits de recursos a cada pod (com CPU i memòria), garantint que cap estudiant pugui monopolitzar els recursos del sistema.
4. **Desplegament ràpid i automàtic:** Mitjançant plantilles prèviament definides (YAML), Kubernetes pot crear i configurar ràpidament els entorns per a cada estudiant, reduint el temps de preparació i eliminant errors manuals.
5. **Monitorització i gestió amb Prometheus i Grafana:** Les eines de monitorització proporcionen visibilitat en temps real del rendiment del sistema, permetent ajustar els recursos o solucionar problemes abans que afectin l'experiència dels estudiants.

3.2. Organització sanitària: Aïllament de dades entre hospitals

3.2.1. Descripció del cas d'ús

Aquest cas d'ús se centra en una organització sanitària que gestiona múltiples hospitals amb una única plataforma de gestió centralitzada. Tot i compartir la infraestructura, cada hospital ha de tenir els seus propis serveis i bases de dades aïllats per motius legals (compliment del RGPD o regulacions similars) i de seguretat, evitant la contaminació de dades o la interferència entre hospitals.

L'objectiu principal és crear una infraestructura on:

1. Cada hospital tingui accés a un entorn dedicat (aplicacions, bases de dades, serveis) que sigui independent dels altres.
2. Els recursos assignats a un hospital no puguin ser utilitzats per un altre.
3. Es mantingui un control eficient dels recursos disponibles per optimitzar costos operatius.

3.2.2. Beneficis que aporta Kubernetes

1. **Aïllament total amb Namespaces:**
    - Kubernetes permet crear **namespaces** independents per a cada hospital, on es poden desplegar els seus propis serveis i dades sense interferències amb altres entitats.
2. **Gestió eficient dels recursos:**
    - Amb les configuracions de **Resource Quotas** i **Limit Ranges**, es poden definir límits específics per a cada namespace, assegurant que cap hospital consumeixi més recursos del necessari.
3. **Escalabilitat i flexibilitat:**
    - Els namespaces permeten afegir nous hospitals o modificar els existents fàcilment sense afectar la resta de la infraestructura.
4. **Monitorització i seguiment:**
    - Kubernetes, juntament amb eines com Prometheus i Grafana, pot monitoritzar l'ús dels recursos de cada namespace per garantir que els sistemes funcionin de manera òptima.
5. **Compliment normatiu:**
    - L'aïllament garantit pels namespaces facilita el compliment de les normatives legals sobre privacitat i seguretat de dades.

3.3. Start-up de contingut multimèdia online

3.3.1. Descripció del cas d'ús

Aquest cas d'ús tracta sobre una startup que gestiona una plataforma de pel·lícules i sèries amb usuaris repartits arreu del món. Durant llançaments de contingut popular, com una nova temporada d'una sèrie d'èxit, el tràfic a la plataforma es dispara, provocant sobrecàrregues, latència i possibles interrupcions si la infraestructura no està ben optimitzada.

L'objectiu principal és assegurar que la plataforma sigui capaç de:

1. Proporcionar contingut amb temps de resposta mínims a milions d'usuaris simultàniament.
2. Gestionar pics d'ús sense interrupcions o pèrdues de qualitat.
3. Distribuir la càrrega de manera eficient entre diverses ubicacions per optimitzar els recursos i reduir els costos operatius.

3.3.2. Beneficis que aporta Kubernetes

1. **Balanceig de càrrega intel·ligent amb Ingress Controller:**
    - Kubernetes utilitza un **Ingress Controller** per gestionar el tràfic d'entrada, distribuint-lo entre diversos pods que executen el servei. Això evita la saturació d'un únic punt i garanteix temps de resposta baixos per als usuaris.
2. **Alta disponibilitat amb múltiples rèpliques de serveis:**
    - Kubernetes permet mantenir múltiples rèpliques de cada servei actives, assegurant que, fins i tot si un pod o node falla, els usuaris poden continuar accedint al contingut sense interrupcions.
3. **Escalabilitat automàtica amb HPA:**
    - Durant pics d'ús, Kubernetes pot crear automàticament nous pods per gestionar l'augment del tràfic, assegurant que sempre hi ha prou capacitat per servir tots els usuaris.
4. **Desplegament global:**
    - Amb Kubernetes, la startup pot distribuir els pods a diversos centres de dades o regions del núvol per apropar el contingut als usuaris i reduir la latència.
5. **Monitorització en temps real:**
    - Kubernetes, juntament amb Prometheus i Grafana, proporciona mètriques detallades del tràfic, l'ús de recursos i el rendiment, permetent ajustar la infraestructura segons les necessitats en temps real.

4. Cas d'ús 1: Exàmens finals de desenvolupament amb entorns aïllats

4.1 Objectius, requisits funcionals i no funcionals

4.1.1 Objectius específics

O1: Crear una solució que permeti a la universitat proporcionar entorns de desenvolupament independents, utilitzant Kubernetes per gestionar i aïllar els recursos de cada entorn, amb la capacitat de garantir un temps de desplegament inferior a 5 minuts per entorn.
02: Permetre a un usuari modificar el seu codi en un repositori i que s'actualitzi sense haver de rellançar les aplicacions de l'entorn, aconseguint que els canvis en el codi es sincronitzin en temps real amb una latència inferior a 1 minut.
03: Automatitzar el desplegament i eliminació dels entorns per a cada estudiant mitjançant CI/CD i plantilles YAML, reduint el temps de preparació i el risc d'errors manuals, i garantint que el procés de creació d'entorns es faci en menys de 5 minuts.
04: Monitoritzar en temps real el rendiment dels entorns de desenvolupament de cada estudiant, utilitzant Prometheus per recopilar mètriques com l'ús de CPU, memòria i trànsit de xarxa.
05: Assegurar que els recursos utilitzats pels entorns dels estudiants siguin eficients, evitant la sobrecàrrega del sistema durant els períodes inactius entre sessions d'examen, reduint el consum de recursos en almenys un 30% respecte a l'ús habitual i optimitzant l'ús dels recursos disponibles durant els períodes de baixa activitat.

4.1.2 Requisits funcionals

Requisit Funcional	Objectiu
El sistema ha de permetre que els usuaris (estudiants) puguin crear i eliminar el seu entorn de forma automatitzada a través d'una API, amb un temps de creació inferior a 5 minuts per entorn.	O1,O3
El sistema ha de permetre als estudiants modificar el seu codi dins d'un repositori i que els canvis es reflecteixin automàticament en el seu entorn de treball, sense necessitat de reiniciar l'entorn. Els canvis s'han de sincronitzar en temps real amb una latència inferior a 1 minut.	O2
El sistema ha de permetre als administradors visualitzar les mètriques de rendiment de cada entorn de desenvolupament (per exemple, ús de CPU, memòria i trànsit de xarxa), a través de taules i gràfics que mostrin l'estat del sistema en temps real.	O4
El sistema ha de permetre als administradors gestionar els permisos d'accés per als estudiants i professors, amb la capacitat d'assignar rols específics (per exemple, professor, estudiant) i controlar què pot fer cada usuari en el seu entorn de desenvolupament. Això inclou la creació d'entorns, l'accés al codi, la modificació de recursos i la visualització de mètriques.	O1, O4
El sistema ha de proporcionar una interfície d'usuari senzilla i intuïtiva per als estudiants, permetent-los gestionar els seus entorns de desenvolupament,	O2

com ara accedir-hi, modificar el seu codi, veure les mètriques de rendiment, i interactuar amb altres serveis necessaris com bases de dades o API's.	
El sistema ha de ser capaç de gestionar entorns de desenvolupament que suportin diversos llenguatges de programació (per exemple, Python, Java, C++, etc.), amb la capacitat de configurar automàticament els entorns de desenvolupament per cada estudiant en funció del llenguatge que estigui utilitzant en el seu examen.	O1,O2
El sistema ha de permetre als estudiants accedir als logs d'activitat dels seus entorns, de manera que puguin veure l'històric d'execució de les seves aplicacions i depurar errors. A més, els professors i administradors han de poder accedir als logs d'activitat per controlar el rendiment i detectar possibles anomalies en els entorns.	O4

4.1.3 Requisits no funcionals

Requisit No Funcional	Objectiu
El sistema ha de garantir la seguretat i l'aïllament complet dels entorns de desenvolupament de cada estudiant, de manera que les dades i recursos d'un estudiant no siguin accessibles per altres. El sistema ha de ser capaç de garantir que qualsevol error o vulnerabilitat d'un entorn no afecti els altres entorns.	O1
El sistema ha de ser capaç de manejar un augment significatiu de la càrrega de treball durant els pics de demanda (per exemple, en moments d'examen). Això inclou l'escalabilitat automàtica de la infraestructura de Kubernetes per assegurar que la plataforma pugui gestionar un nombre elevat de clients simultanis (més de 100 estudiants) sense una degradació del servei.	O1
El sistema ha de garantir una disponibilitat de servei superior al 99,9% durant el període d'examen, assegurant que els entorns de desenvolupament estiguin disponibles en tot moment per als estudiants. Això inclou la recuperació ràpida en cas de fallades, així com la capacitat d'automatitzar els processos de recuperació de dades.	O3
El sistema ha de garantir que els temps de resposta del sistema siguin inferiors a 1 segon per a operacions de creació i actualització d'entorns, i que els canvis en el codi dels estudiants es sincronitzin en temps real amb una latència inferior a 1 minut.	O2
El sistema ha de garantir que els recursos (CPU, memòria, xarxa) siguin utilitzats de manera òptima, evitant la sobrecàrrega del sistema durant els períodes d'inactivitat entre exàmens, i ajustant-se a les necessitats específiques de cada estudiant. Això inclou la capacitat de reduir el consum de recursos en almenys un 30% respecte a l'ús habitual durant els períodes d'inactivitat.	O5
El sistema ha de ser compatible amb Git, per facilitar la integració amb les eines existents que utilitzen els estudiants per gestionar el seu codi. Aquesta compatibilitat ha de garantir una fàcil importació, modificació i actualització del codi en els entorns de desenvolupament.	O2
El sistema ha de registrar totes les accions i esdeveniments crítics dins de la plataforma (creació i eliminació d'entorns, canvis en el codi, ús de recursos, etc.) per garantir que es pugui realitzar una auditoria completa en cas de necessitat. Això inclou la possibilitat de generar informes detallats de les accions dels estudiants i administradors.	O4

El sistema ha de permetre als administradors personalitzar fàcilment la configuració dels entorns per a diferents cursos, exàmens o tipus de tasques. Aquesta flexibilitat ha de ser simple d'implementar, amb la capacitat de modificar les plantilles YAML per adaptar-se a diferents requisits sense necessitat d'intervenció tècnica complexa.	O1,O3
El sistema ha de ser capaç de recuperar-se de fallades de qualsevol component (com nodes de Kubernetes o eines de monitorització) sense afectar els entorns dels estudiants ni interrompre l'examen. La recuperació ha de ser automàtica i garantir que els entorns es recuperin ràpidament sense pèrdues de dades.	O3,O4

4.2 Anàlisi i disseny

4.2.1 Idea i plantejament inicial

La idea principal d'aquest cas d'ús és crear una plataforma que permeti a una universitat d'informàtica proporcionar entorns de desenvolupament personalitzats i aïllats per a cada estudiant durant els exàmens pràctics. Aquest sistema ha de permetre que cada estudiant disposi d'un entorn independent per desplegar el seu codi i interactuar amb altres serveis com bases de dades o API's, tot mantenint la seguretat i el rendiment del sistema. La necessitat d'aquest tipus d'entorns neix de la creixent demanda per a la realització d'exàmens pràctics que requereixen un alt nivell de seguretat, ja que els entorns han de ser completament aïllats entre estudiants per evitar la contaminació de dades o la interacció no autoritzada entre ells.

La solució plantejada per aquest cas d'ús proposa utilitzar una arquitectura basada en Kubernetes, un sistema de contenidors i orquestració que permet gestionar entorns aïllats mitjançant namespaces dins del mateix cluster. Cada estudiant disposarà d'un namespace personalitzat, amb la possibilitat de gestionar-hi serveis com bases de dades i altres recursos de manera independent, garantint així que els recursos del sistema no siguin compartits entre estudiants. Això permetrà a la universitat proporcionar un sistema d'exàmens escalable, eficaç i fàcil de gestionar, ja que cada entorn es desplegarà de manera automatitzada mitjançant CI/CD i YAML templates, amb el propòsit de garantir la disponibilitat i consistència dels entorns durant tota la durada de l'examen.

Aquest plantejament inicial busca resoldre els problemes derivats de l'escalabilitat, l'aïllament de dades i la seguretat, tot mantenint un alt rendiment i la possibilitat d'automatitzar la creació i l'eliminació dels entorns d'examen de forma ràpida i eficient. A més, s'incorpora una estratègia de monitoratge en temps real per assegurar que els recursos es gestionen de manera òptima, utilitzant Prometheus i Grafana per monitoritzar l'ús de CPU, memòria i altres mètriques crítiques, garantint així que els entorns no tinguin un impacte negatiu en el rendiment global de la plataforma.

4.2.2 Estudi i anàlisi de tecnologies

Per aquest cas d'ús, s'han seleccionat diverses tecnologies que permeten complir els requisits de seguretat, aïllament d'entorns, escalabilitat i automatització del sistema. Aquestes tecnologies proporcionen la flexibilitat, eficiència i robustesa necessàries per gestionar entorns personalitzats per a cada estudiant durant els exàmens pràctics.

Kubernetes

Kubernetes és una plataforma d'orquestració de contenidors que facilita la gestió, automatització, escalabilitat i desplegament de contenidors en entorns distribuits. La seva adopció és fonamental per aquest projecte ja que permet la creació de namespaces independents per cada estudiant dins d'un mateix cluster. Cada entorn de desenvolupament s'aïlla a nivell de namespace, evitant qualsevol interacció no autoritzada entre estudiants i protegint les dades de manera eficaç.

Funcionalitats:

- **Aïllament de recursos**: mitjançant l'ús de namespaces, es poden gestionar múltiples entorns independents sense que un afecti els altres.
- **Escalabilitat**: la infraestructura es pot escalar automàticament en funció de la demanda, afegint més nodes i recursos a mesura que el nombre d'estudiants creix.
- **Alta disponibilitat**: Kubernetes garanteix que els entorns siguin sempre disponibles, fins i tot en cas de fallades de nodes o altres components de la infraestructura.
- **Automatització**: el desplegament i la gestió dels entorns es poden automatitzar mitjançant YAML templates i sistemes de CI/CD, reduint els errors humans i millorant l'eficiència operativa.

Prometheus i Grafana

Per garantir el rendiment òptim dels entorns i evitar problemes de recursos durant l'examen, s'ha seleccionat Prometheus per a la monitorització i Grafana per a la visualització de dades. Aquestes eines proporcionaran informació en temps real sobre l'estat dels recursos del sistema i permetran identificar ràpidament possibles anomalies.

Prometheus és una eina de monitorització de codi obert dissenyada per recollir i emmagatzemar mètriques de diversos serveis i recursos dins d'una infraestructura de Kubernetes. Grafana s'utilitza per visualitzar aquestes mètriques de manera visual, oferint als administradors de la plataforma una eina potent per fer seguiment de la salut del sistema i identificar qualsevol anomalia o col·lapse de recursos.

Funcionalitats:

- **Monitorització de mètriques**: control de l'ús de CPU, memòria, trànsit de xarxa i l'estat de cada pod.
- **Alertes en temps real**: la plataforma generarà alertes automàtiques basades en criteris com l'ús excessiu de CPU o memòria, permetent una resposta ràpida per evitar problemes de rendiment.

- **Visualització de dades**: els dashboards de Grafana proporcionaran una visió clara de l'estat de cada entorn, facilitant la presa de decisions i la resolució de problemes de manera eficient.

GitLab

Git és la tecnologia de control de versions utilitzada per gestionar el codi font del sistema, així com per a la gestió de les configuracions dels entorns. GitLab serà la plataforma utilitzada per allotjar els repositoris Git i per gestionar els fluxos de treball de CI/CD (Integració contínua i Desplegament continu). L'automatització amb CI/CD permetrà que els entorns es despleguin automàticament amb el codi més actualitzat sense necessitat d'intervenció manual.

Funcionalitats:

- **Control de versions**: cada entorn de desenvolupament estarà vinculat a una branca de codi específica, permetent actualitzacions i gestió eficient del codi dels entorns d'examen.
- **Automatització del flux de treball**: l'automatització amb CI/CD automatitza els processos de creació i desplegament dels entorns, garantint que el codi i les configuracions estiguin sempre actualitzats.

Flask i Python

Flask és un framework de desenvolupament web lleuger i flexible per a Python, dissenyat per facilitar la creació d'aplicacions web i APIs. A diferència d'altres frameworks més complets, Flask proporciona una base mínima però ampliable, ideal per a projectes que requereixen control total sobre l'arquitectura de l'aplicació. Python és un llenguatge de programació altament llegible i potent, amb una comunitat activa i una gran varietat de biblioteques i eines que faciliten la creació i gestió de serveis com API RESTful, gestió de bases de dades i comunicació amb altres sistemes. Flask i Python es complementen perfectament per crear solucions escalables i ràpides de desplegar, amb un alt nivell d'abstracció que facilita l'automatització de processos i la gestió de recursos dins d'un sistema distribuït com Kubernetes.

Funcionalitats:

- **Creació d'API lleugeres i eficients**: Flask permet crear rutes específiques per gestionar la creació i eliminació d'entorns de reproducció per als usuaris de la plataforma de manera ràpida i eficient. La simplicitat de Flask i la potència de Python permeten un desenvolupament ràpid de la API, estalviant temps i recursos a l'hora de dissenyar i desplegar nous serveis.
- **Integració amb altres sistemes**: La integració senzilla amb Kubernetes, bases de dades i altres serveis fa que Flask sigui ideal per orquestrar la interacció entre els diversos components de la plataforma.
- **Flexibilitat i escalabilitat**: Flask permet desenvolupar una API flexible i personalitzable que es pot escalar a mesura que creix el nombre d'usuaris i serveis, gestionant adequadament múltiples instàncies d'entorns d'usuari.

Base de dades SQL

Les bases de dades SQL (com MySQL o PostgreSQL) són essencials per emmagatzemar i gestionar les dades relacionals de la plataforma, com les metadades de contingut multimèdia, usuaris i preferències. Les bases de dades relacionals són idònies per a sistemes que requereixen operacions transaccionals consistents i la gestió de grans volums de dades relacionades.

Funcionalitats:

- **Consultes eficients**: SQL permet consultar i filtrar dades de manera eficient, essencial per a una plataforma que gestiona un gran volum de contingut i/o usuaris simultanis.
- **Persistència de dades**: SQL garanteix que totes les dades estiguin emmagatzemades de manera fiable i segures.
- **Escalabilitat de les dades**: Amb bases de dades relacionals, el sistema pot gestionar grans volums de dades relacionades, mantenint la coherència i la integritat de les dades de manera eficient.

Kubernetes plugin i PyYAML

El client de Kubernetes per Python, és una biblioteca que permet interactuar amb Kubernetes utilitzant el llenguatge de programació Python. Aquesta eina és essencial per la gestió i l'automatització de la infraestructura Kubernetes, ja que permet gestionar de manera programàtica recursos com Pods, Services, Deployments, Namespaces, Persistent Volumes i altres objectes de Kubernetes a través de l'API de Kubernetes. Aquesta biblioteca és especialment útil per a la creació i manipulació dinàmica de recursos en un clúster Kubernetes.

L'ús del client de Kubernetes permet automatitzar el procés de creació i eliminació d'entorns, gestionar els recursos associats a cada namespace, i actualitzar els manifestos YAML necessaris per a la configuració de serveis i aplicacions dins del clúster. La seva integració amb Python permet desenvolupar scripts i funcions que poden interactuar directament amb el clúster de Kubernetes, facilitant tasques com la creació de namespaces per als usuaris, la configuració de PVCs per a l'emmagatzematge, la creació de serveis i desplegaments, i molt més.

Funcionalitats:

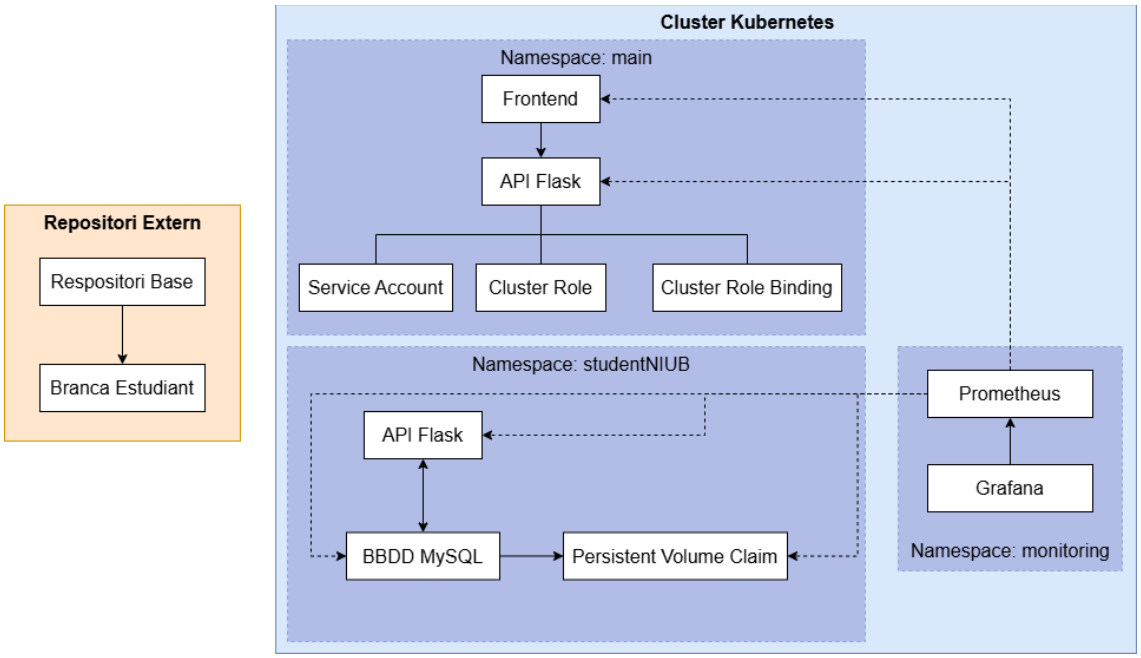
- **Gestió automatitzada de Kubernetes:** La biblioteca permet crear i gestionar de manera programàtica recursos com Deployments i Namespaces, essencial per a un sistema de múltiples entorns.
- **Integració amb Python i Flask:** La combinació del Kubernetes Python Client amb Flask permet crear una API que automàticament gestiona el desplegament d'entorns per cada usuari, simplificant la creació d'entorns personalitzats mitjançant Kubernetes a partir d'una única interfície API.
- **Manipulació de manifestos YAML:** Kubernetes utilitza manifestos YAML per descriure l'estat desitjat de la infraestructura. La biblioteca PyYAML permet llegir, modificar i aplicar aquests manifestos YAML dinàmicament, garantint una configuració constant i consistent dels recursos.

- **Escalabilitat i eficiència:** La gestió automàtica i programàtica dels recursos mitjançant Python facilita l'escalabilitat del sistema, ajustant la infraestructura de Kubernetes segons la demanda sense necessitat de configuració manual.

4.2.3 Disseny de l'arquitectura

L'arquitectura proposada es basa en una infraestructura de Kubernetes per garantir l'aïllament de recursos i l'escalabilitat dels entorns de desenvolupament per cada estudiant. A continuació es detalla el disseny de l'arquitectura tenint en compte les necessitats de seguretat, eficiència i automació del sistema.

1. Arquitectura General



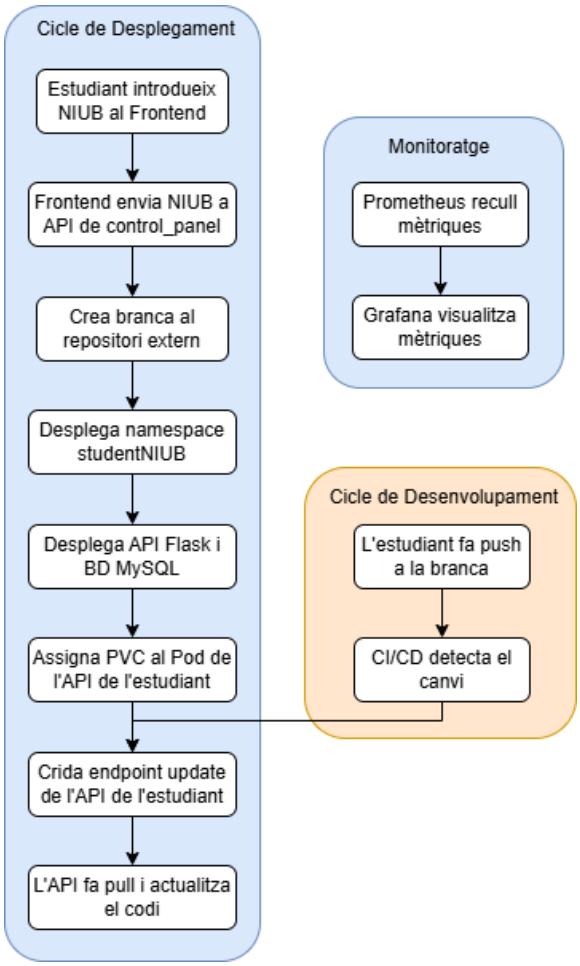
L'arquitectura de la plataforma es compon de diversos components clau, cadascun amb una funció específica en el procés d'examen. A continuació es descriuen els elements clau de l'arquitectura:

- **Kubernetes Cluster**: La plataforma es desplega sobre un clúster de Kubernetes que permet gestionar de manera eficient i automàtica els recursos dels entorns d'examen. Cada estudiant disposarà del seu propi namespace, una entitat de Kubernetes que permet aïllar els recursos de cada entorn per garantir que no interfegeixin entre ells.
- **Frontend**: El frontend és una aplicació web on els estudiants introdueixen el seu NIUB (Núm. Únic d'Identificació de l'Estudiant) per iniciar el procés de creació del seu entorn d'examen. Aquesta aplicació es desplega en un namespace principal dins de Kubernetes i és responsable d'enviar el NIUB al servei API Flask, que és el backend que gestionarà la creació dels entorns.
- **API Flask**: El backend utilitza Flask, un micro-framework de Python, per gestionar la comunicació entre el frontend i els entorns Kubernetes. Quan un estudiant introdueix el seu NIUB, l'API Flask crea una branca en el repositori extern, activa la creació del namespace per l'estudiant i desplega l'entorn d'examen amb les configuracions adequades. L'API també gestiona la creació d'un Persistent Volume

Claim (PVC) per garantir que els recursos del sistema s'assignin de manera apropiada al pod de cada estudiant.

- **Repositori extern**: Un repositori extern conté el codi base de la plataforma, així com les configuracions YAML per a cada entorn. Quan un estudiant sol·licita un nou entorn, una nova branca es crea al repositori, i el codi es sincronitza amb els entorns de desenvolupament personalitzats.
- **Namespace studentNIUB**: Cada estudiant té assignat un namespace independent dins del clúster de Kubernetes. En aquest espai aïllat, el sistema desplega les aplicacions necessàries, com l'API Flask, la base de dades MySQL i els volums persistents associats als entorns d'examen. Aquest aïllament garanteix que no hi hagi interacció entre els entorns de diferents estudiants.
- **Namespace de monitoratge**: Per garantir el bon funcionament dels entorns d'examen, es disposa d'un namespace dedicat a monitorització, que utilitza Prometheus per recollir dades sobre l'ús de recursos (CPU, memòria, trànsit de xarxa) i Grafana per visualitzar aquestes mètriques en temps real. Els administradors poden així identificar ràpidament qualsevol problema de rendiment o ús excessiu de recursos en els entorns dels estudiants.

2. Flux de treball del sistema



El flux de treball del sistema es pot dividir en diversos passos clau que es mostren a continuació:

1. **Introducció del NIUB per part de l'estudiant**: L'estudiant introdueix el seu NIUB al frontend de la plataforma, iniciant així el procés de creació del seu entorn d'examen.
2. **Creació de la branca al repositori extern**: Quan l'estudiant envia el seu NIUB, el frontend comunica amb l'API Flask, que crea una nova branca al repositori extern. Aquesta branca conté la configuració específica per a l'entorn de l'estudiant.
3. **Desplegament del namespace personalitzat**: Un cop la branca ha estat creada, el sistema desplega un nou namespace dins del clúster Kubernetes per a l'estudiant, utilitzant la configuració especificada al repositori extern. Això inclou la creació de recursos com API Flask, bases de dades MySQL, i PVC (Persistent Volume Claim) per a emmagatzematge de dades.
4. **Monitorització del sistema**: Un cop l'entorn ha estat desplegat, es comença a monitoritzar l'ús de recursos mitjançant Prometheus, que recull dades sobre el rendiment i les mètriques de l'entorn de l'estudiant. Grafana mostra aquestes mètriques als administradors per tal de poder actuar ràpidament en cas de detectar anomalies.
5. **Interacció de l'estudiant amb l'entorn**: L'estudiant pot començar a desenvolupar i interactuar amb la seva aplicació, utilitzant serveis com les bases de dades o APIs internes del seu entorn d'examen, amb total aïllament dels altres estudiants.
6. **Eliminació automàtica de l'entorn**: Un cop l'examen finalitza, el sistema elimina automàticament l'entorn de l'estudiant, tancant el namespace creat i alliberant els recursos utilitzats.

3. Infraestructura i escalabilitat

L'arquitectura proposada fa ús de la capacitat de Kubernetes per escalar automàticament els recursos en funció de la demanda, especialment durant els pics d'ús, com els moments en què diversos estudiants accedeixen simultàniament a la plataforma. Gràcies al desplegament de subentorns per namespace, permet eliminar els recursos quan l'usuari ha finalitzat l'examen només eliminant el namespace, pel que permet estalviar costos i recursos innecessaris.

A més, la integració amb CI/CD permet gestionar de manera senzilla i automàtica el desplegament de noves versions del sistema i l'actualització de les configuracions sense necessitat d'intervenció manual.

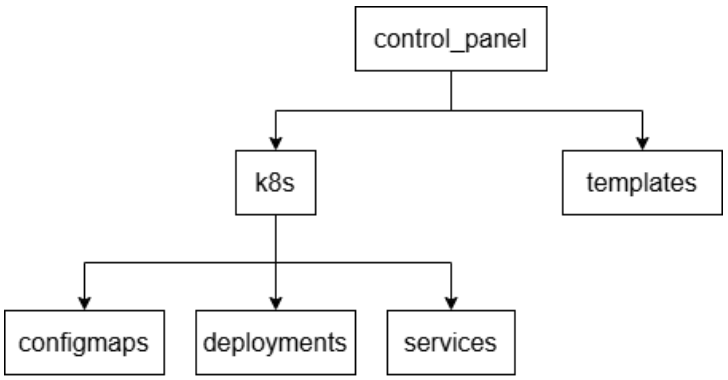
L'arquitectura proposada per al cas d'ús "Exàmens finals de desenvolupament amb entorns aïllats" permet crear una solució escalable, eficient i segura que compleix amb els requisits de seguretat i eficiència operativa. Mitjançant Kubernetes i CI/CD, es pot automatitzar el procés de creació i eliminació d'entorns, mentre que les eines de monitorització com Prometheus i Grafana asseguren que el sistema estigui constantment controlat i que els recursos es gestionin de manera òptima. Aquesta arquitectura permet proporcionar una experiència d'examen fluida i sense interrupcions per a tots els estudiants.

4.3 Desenvolupament

4.3.1 Desplegament de la plataforma

La plataforma s'ha desenvolupat mitjançant una arquitectura de contenidors Kubernetes que permet gestionar i desplegar entorns aïllats per a cada estudiant, proporcionant una solució escalable i eficient. L'estructura de carpetes del primer cas d'ús (CU1) s'ha organitzat tenint en compte la necessitat de separar les funcionalitats del sistema en diferents components per garantir la modularitat i la facilitat de gestió. Aquesta estructura també facilita l'escalabilitat i permet un control centralitzat a l'hora de gestionar els serveis que es despleguen al llarg de l'arquitectura.

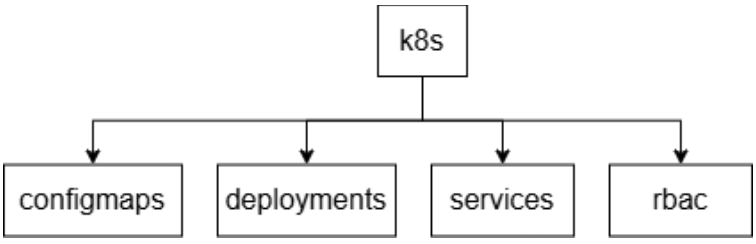
control_panel



Aquesta carpeta conté el codi i les configuracions relacionades amb el panell de control de la plataforma. S'hi inclou la configuració dels serveis de monitoratge i els templates necessaris per configurar els diferents entorns d'examen.

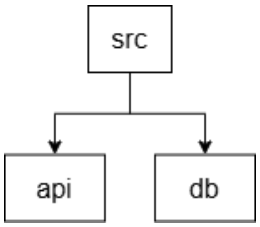
- **k8s**: Conté els fitxers de configuració per als recursos de Kubernetes, com configmaps, deployments i services. Aquests fitxers defineixen com s'han de configurar els recursos per al panell de control dins de l'entorn Kubernetes.

k8s



Aquesta carpeta conté la configuració Kubernetes global per al projecte, incloent-hi les definicions de configmaps, deployments, rbac (per la gestió de permisos) i services. Aquests fitxers són responsables de la creació i gestió dels recursos dins del clúster Kubernetes que gestiona les aplicacions desplegades a la plataforma.

src



Aquesta carpeta conté el codi font base de l'aplicació que ha de modificar i seguir desenvolupant l'estudiant durant l'examen, amb subcarpetes per als diferents components de l'aplicació.

- **api**: Conté el codi relacionat amb l'API que realitza les operacions amb la base de dades i l'usuari.
- **db**: Conté els fitxers de configuració i codi per gestionar la base de dades de l'aplicació de l'estudiant.

Per desplegar la plataforma de forma inicial, he desenvolupat un script que desplega els diferents serveis als clúster de Kubernetes un cop creat. Aquest codi realitza els següents passos:

1. Creació del namespace de monitoratge

```
# Create the monitoring namespace

kubectl create namespace monitoring
```

Aquest primer pas crea un namespace dedicat anomenat monitoring, el qual serà utilitzat per allotjar tots els serveis relacionats amb el monitoratge (com Prometheus i Grafana). Els namespaces permeten separar de manera eficient els diferents serveis dins del mateix clúster de Kubernetes, evitant que entrin en conflicte entre ells i mantenint-los aïllats.

2. Desplegament de Prometheus amb Helm

```
#Deploy Prometheus stack with helm

# Add the prometheus-community helm repo

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

# Install the values.yaml file

helm upgrade --install -f k8s/deployments/prometheus-values.yaml kube-prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring
```

El desplegament de Prometheus es fa mitjançant Helm, que és una eina per gestionar paquets de Kubernetes. Aquí es fa el següent:

- S'afegeix el repositori de Prometheus per poder utilitzar els charts de Helm.
- S'actualitzen els repositoris disponibles.

- Es desplega el stack de Kube-Prometheus-Stack (que inclou Prometheus i Grafana) utilitzant un fitxer prometheus-values.yaml, on es defineixen les configuracions necessàries per al desplegament dins del namespace monitoring.

3. Creació del namespace principal

```
# Create main namespace

kubectl create namespace main
```

Aquesta instrucció crea un namespace principal anomenat main, on es desplegaran tots els serveis relacionats amb el panell de control i altres recursos associats.

4. Creació de la imatge Docker i desplegament del panell de control

```
# Build and push control panel image

docker build -t arnaucirera98/control-panel ./control_panel

docker push arnaucirera98/control-panel:latest
```

Aquest pas construeix i puja la imatge del panell de control al repostiri públic de Docker Hub. Això permetrà crear un recurs de Kubernetes de tipus Deployment, que descarregui aquesta imatge per desplegar-la en un pod.

També es pot pujar en un repositori de contenidors propi o privat i configurar les credencials d'accés a Kubernetes per tal de que pugui accedir-hi, però en aquest projecte ho he volgut fer amb el repositori de Docker públic per facilitar el desplegament i configuració d'aquestes imatges i mantenir el projecte públic.

5. Creació de permisos i desplegament del Control Panel

```
# Create ServiceAccount, ClusterRole and ClusterRoleBinding for control panel

kubectl apply -f k8s/rbac/control-panel-rbac.yaml -n main
```

Es crea un ServiceAccount, ClusterRole i ClusterRoleBinding per al panell de control, que defineix els permisos d'accés i la seguretat dins del sistema Kubernetes. Això assegura que els serveis del panell de control tinguin els permisos adequats per interactuar amb altres recursos dins del clúster.

```
# Deploy control panel

kubectl apply -f k8s/deployments/control-panel-deployment.yaml -n main

kubectl apply -f k8s/services/control-panel-service.yaml -n main
```

Finalment, es desplega el panell de control a través dels fitxers de deployment i service. El deployment gestiona l'instància del panell de control, mentre que el service expose el panell perquè sigui accessible a altres components del sistema o usuaris. La imatge utilitzada per desplegar el pod és la imatge pujada a Docker Hub en el pas anterior.

4.3.2 Panell de control

En aquesta secció s'explica la funció general i l'estructura del Control Panel dins de la plataforma. El Control Panel és l'eina que permet la creació i eliminació d'un entorn d'un estudiant. Aquest, mitjançant una aplicació web, pot introduir el seu NIUB i rebre les instruccions per accedir al seu entorn, així com eliminar el seu entorn un cop finalitzat l'examen.

El panell de control està gestionat per una API, que amb la llibreria de kubernetes per python, crea el subentorn de l'estudiant en el mateix clúster on es troba desplegat, gràcies a que li hem donat accés al pod de panell de control per crear i modificar recursos de tot el clúster a través del ClusterRole.

Components del panell de control:

Kubernetes

Es troben els diferents manifests YAML que despleguen i configuren un subentorn per un estudiant amb les aplicacions base definides. Estructurat en subcarpetes per recursos, ens trobem els següents elements:

A. **Configmaps**
   - **mysql-configmap.yaml:** Permet configurar la base de dades MYSQL dins del contenidor permetent referenciar aquests valors a altres manifests YAML:
     - **MYSQL_ROOT_PASSWORD:** La contrasenya de l'usuari root de MySQL.
     - **MYSQL_DATABASE:** El nom de la base de dades que es crearà (en aquest cas, student_db).
     - **MYSQL_USER:** L'usuari de la base de dades (student_user).
     - **MYSQL_PASSWORD:** La contrasenya associada a l'usuari (student_password).
     - **INIT_SQL:** Un script SQL que inicialitza la base de dades, creant una taula students si no existeix, amb les columnes id, name, i score.
   - **mysql-secret.yaml:** Defineix les credencials d'accés a la base de dades codificades en base64 perquè els recursos necessaris puguin utilitzar-les de forma segura:
     - **DATABASE_NAME:** El nom de la base de dades (en aquest cas, el valor codificat és student_db).
     - **DATABASE_HOST:** El nom del servidor de la base de dades (en aquest cas, mysql-service).
     - **DATABASE_USER:** L'usuari de la base de dades (en aquest cas, student_user).
     - **DATABASE_PASSWORD:** La contrasenya associada a l'usuari de la base de dades (en aquest cas, student_password).

B. **Deployments**
   - **api-deployment.yaml:** Defineix el desplegament del contenidor de la API utilizant la imatge base del codi font de l'estudiant publicada al repositori de Docker Hub. S'utilitzen valors d'entorn per configurar la connexió a la base de dades MySQL mitjançant el secret definit a configmaps. Les variables d'entorn com **DATABASE_HOST**, **DATABASE_USER**, **DATABASE_PASSWORD**, i

DATABASE_NAME es defineixen referenciant al secret per garantir que les credencials no es mostrin en text clar.

- **api-pvc.yaml:** Defineix un PersistentVolumeClaim (PVC) que proporciona un emmagatzematge persistent per els arxius de l'API, garantint que les dades es mantinguin persistents, fins i tot si el contenidor es reinicia o es crea una nova instància. Això permet mantenir el codi font de l'aplicació de l'estudiant (api i frontend) actualitzat en un volum, de forma que si s'actualitza el codi externament, l'API s'actualitzarà en temps real i es mantindrà de forma persistent, sense necessitat de reiniciar el contenidor o tornar a descarregar el codi de forma manual. També es utilitzar per crear un volum per MYSQL per tal de mantenir les dades de la base de dades, persistents.
- **mysql-deployment.yaml:** Defineix el desplegament de la base de dades MySQL, utilitzant la imatge oficial de Docker (mysql:5.7). Es defineixen les variables d'entorn per a la configuració de MySQL referenciant a les variables definides al configmap de MYSQL. També s'assigna un volum persistent per emmagatzemar les dades de MSQL.

C. **Services**

- **api-service.yaml:** Defineix un servei per exposar l'API Python, permetent que altres components i usuaris es comuniquin amb ell a través del port 80 que redirigeix al 5000 dins dels contenidor.
- **mysql-service.yaml:** Defineix un servei per exposar la base de dades MYSQL dins del clúster (ja que només volem que accedeixi la propi API que estarà al mateix namespace) utilitzant el port per defecte de MYSQL 3306.

Templates

Els fitxers de plantilles HTML (index.html i instructions.html) formen part de la interfície web que utilitzen els usuaris per interactuar amb el Control Panel. Aquestes plantilles permeten als estudiants crear i eliminar els seus entorns d'examen i proporcionar-los instruccions detallades sobre com connectar-se al seu entorn. Els fitxers utilitzen Flask per generar dinàmicament el contingut basat en les interaccions dels usuaris.

A. **Index.html**

Proporciona una interfície simple per als usuaris per crear o eliminar un entorn d'examen. Aquest fitxer conté un formulari HTML per introduir el NUIB (Núm. Únic d'Identificació de l'Estudiant) i realitzar les accions.

B. **Instruction.html**

Es genera després que s'hagi creat un entorn d'examen per a un estudiant. Aquest fitxer mostra informació sobre l'entorn creat, com ara el namespace assignat a l'estudiant i les instruccions per connectar-se a la base de dades MySQL i modificar el codi de l'API. A través de la plantilla, es mostren les dades dinàmiques associades amb l'entorn de l'estudiant, com el namespace.

App.py

És el component central de l'aplicació que gestiona les rutes i operacions per crear i eliminar entorns d'examen dins de l'arquitectura Kubernetes. S'utilitza el framework Flask

per a gestionar les peticions HTTP i Kubernetes Python Client per interactuar amb el clúster Kubernetes. Es realitza els següents processos:

1. **Inicialització de l'aplicació Flask i càrrega de la configuració de Kubernetes:** Flask s'inicia per gestionar les peticions web i *config.load_incluster_config()* carrega la configuració de Kubernetes per interactuar amb el clúster des de dins del mateix.
2. **Ruta per crear un entorn d'examen:** És cridat pel formulari HTML definit a la interfície web i rep el NIUB de l'estudiant per tal de crear un namespace amb el nom "studentNIUB". Crea els diferents recursos necessaris, descrits anteriorment, al namespace creat de l'estudiant. També clona el repositori base del codi font i crea una branca amb el niub de l'estudiant.
3. **Ruta per eliminar un entorn:** És cridat pel formulari HTML definit a la interfície web i rep el NIUB de l'estudiant per tal d'eliminar el namespace amb el nom "studentNIUB", eliminant en conseqüència, els recursos desplegats en aquest namespace.

4.3.3 Codi font base per l'examen de l'estudiant

El codi base de l'examen de l'usuari és una aplicació senzilla desenvolupada amb Flask, un framework per a Python, que permet gestionar peticions HTTP per a la gestió de dades a través d'una base de dades. Aquest codi inclou tres funcionalitats bàsiques:

1. **Extreure dades de la base de dades** (/api/data GET): Aquesta ruta permet obtenir dades d'una base de dades MySQL.
2. **Ingresar dades a la base de dades** (/api/data POST): Aquesta ruta permet inserir noves dades en la base de dades MySQL a partir d'un cos de la petició en format JSON.
3. **Actualitzar el repositori** (/api/update_repo POST): Aquesta ruta permet actualitzar el repositori d'on s'extreu el codi a la branca de l'usuari. Aquesta funció s'activa automàticament quan l'estudiant fa un push a la seva branca per tal de que el codi del pod de l'estudiant s'actualitzi automàticament perquè l'usuari pugui veure el resultat de la seva modificació en quan realitzi el push.

Dockerfile

El **Dockerfile** defineix la configuració de la imatge Docker per a l'aplicació. La imatge base utilitzada és un contenidor amb python 3.9, i es realitzen les següents accions:

- S'instal·la git per permetre l'actualització del repositori Git del codi base.
- S'instal·len les dependències del projecte mitjançant el fitxer **requirements.txt**, que inclou les biblioteques Flask, mysql-connector-python, gunicorn i gitpython.
- Finalment, es copia tot el codi font del repositori dins del contenidor i s'executa l'aplicació Flask.

requirements.txt

Aquest fitxer conté les dependències que s'instal·len en el contenidor mitjançant pip. Inclou:

- **Flask** per crear l'aplicació web.
- **mysql-connector-python** per la connexió a la base de dades MySQL.
- **gunicorn**, un servidor WSGI per executar l'aplicació Flask de manera eficient en producció.
- **gitpython**, per interactuar amb repositoris Git de manera programàtica.

4.3.4 Monitoratge

Introducció a Prometheus i Grafana

En aquest context, un clúster de Kubernetes on es creen i eliminen dinàmicament múltiples subentorns d'aplicacions, és essencial disposar d'un sistema de monitoratge robust, flexible i capaç de recollir dades en temps real sobre l'estat i rendiment dels recursos. En aquest cas utilitzo Prometheus i Grafana per monitoritzar el sistema.

Prometheus és una eina de monitoratge i emmagatzematge de mètriques que destaca per la seva capacitat d'obtenir informació detallada i actualitzada constantment dels components del clúster. Prometheus està dissenyat especialment per a entorns dinàmics i distribuïts com Kubernetes, on els recursos es creen, modifiquen i eliminen de manera freqüent. Gràcies a la seva arquitectura basada en l'extracció periòdica de mètriques i la seva sintaxi flexible per a consultes (PromQL), Prometheus permet controlar de manera precisa l'ús de CPU, memòria, xarxa i altres paràmetres clau dels pods i nodes.

Grafana complementa Prometheus oferint una plataforma visual potent i intuïtiva per crear panells de control (dashboards) personalitzats. Aquesta eina facilita la interpretació de les dades recopilades, permetent visualitzar gràficament l'estat del clúster i dels subentorns, així com configurar alertes que avisin quan es detecten anomalies o problemes de rendiment. Això és fonamental per a la detecció precoç d'incidències i per mantenir la fiabilitat del sistema durant períodes de càrrega elevada.

La combinació de Prometheus i Grafana és ideal per a aquest cas d'ús perquè ofereix:

- Monitoratge en temps real d'un entorn altament dinàmic, on els subentorns es creen i s'eliminen constantment a mesura que els estudiants comencen i acaben els exàmens.
- Visibilitat granular per namespaces i components, permetent monitoritzar de manera individualitzada cada entorn d'examen.
- Escalabilitat i automatització, ja que s'adapta automàticament als canvis de l'entorn sense necessitat de configuracions manuals constants.
- Alertes i notificacions que ajuden a prevenir problemes i a garantir que el servei es mantingui disponible i amb un rendiment òptim durant tot el procés d'examen.

Desplegament de Prometheus i Grafana al cúster

Per desplegar les eines de monitoratge, utilitzo Helm, una eina de gestió de paquets per Kubernetes que facilita la instal·lació, configuració i actualització d'aplicacions complexes dins del clúster mitjançant paquets anomenats charts.

El paquet utilitzat és el kube-prometheus-stack, que és un Helm chart mantingut per la comunitat de Prometheus. Aquest stack inclou una instal·lació completa i integrada de Prometheus, Grafana, Alertmanager, exporters i les configuracions bàsiques per monitoritzar un clúster Kubernetes. Així, ofereix una solució completa de monitoratge i visualització en un únic paquet, preparada per a ser desplegada de forma senzilla i eficient.

Aquest desplegament es configura a l'script de desplegament d'aplicacions inicials al clúster, definit en el punt *4.3.1. Desplegament de la Plataforma*.

Al no configurar directament els manifestos YAML del desplegament de les tecnologies del Helm Chart, s'ha de configurar les diferents tecnologies a través d'un arxiu values.yaml on s'indica les diferents variables configurables dels recursos que es desplegaran. Entre les configuracions més destacades d'aquests vàlues es troben:

- **Configuració d'exporters:** S'activen o desactiven exporters específics per recopilar mètriques detallades dels nodes, pods, serveis i altres components de Kubernetes.
- **Exposició dels serveis**: S'especifica com s'exposen Prometheus i Grafana per permetre-hi accés segur i controlat des de fora del clúster o des d'altres components.
- **Configuració d'aplicacions a desplegar**: S'activen o desactiven les diferents aplicacions de l'Stack, així com configurar cada una.
- **Configuració d'alertes**: Es defineixen regles per a alertes automàtiques a Prometheus, permetent la notificació immediata en cas d'anomalies o problemes detectats en el clúster o en els subentorns.

En aquest cas, he utilitzat pràcticament tots els valors per defecte del Helm Chard, excepte les següents variables:

- namespaceOverride: "monitoring" # Permet definir el namespace per a tots els recursos de l'stack. En el meu cas el namespace ja creat de monitoring
- upgradeJob/enabled: "false" # Permet actualitzar la versió d'imatges de forma automatitzada. Ho desactivo al tractar-se del desenvolupament d'un projecte on no m'és necessari tenir un recurs més que m'actualitzi les imatges.
- global/windowsMonitoring/enabled: "false" # Desactiva el desplegament de l'agent extractor de dades que permet monitoritzar sistemes Windows. En aquest cas no s'útilitza dispositius Windows, per tant, no és necessari.
- prometheus/monitor/enabled: "false" # Desactiva la monitorització de prometheus cap a aquests agents extractors de dades de sistemes Windows.

4.4 Anàlisi de costos i escalabilitat i escalabilitat

4.4.1 Escenaris d'ús

En aquest cas d'ús, hem estipulat dos escenaris per tal de subdividir el desplegament en un entorn de desplegament i un entorn de producció d'examen:

Entorn de proves intern

Aquest entorn és una infraestructura de desenvolupament i test que s'executa localment, utilitzant un clúster Kubernetes desplegat mitjançant Docker Desktop.

- **Característiques principals**:
    - No està exposat a internet, garantint un entorn segur i controlat per a proves i desenvolupament.
    - Els recursos (CPU, memòria, emmagatzematge) depenen directament del hardware local on s'executa Docker Desktop, sense costos associats a serveis cloud.
    - Permet provar funcionalitats, desplegar i eliminar entorns ràpidament, i validar el funcionament abans d'arribar a producció.
    - L'escalabilitat està limitada pels recursos del maquinari local i la capacitat de Docker Desktop per gestionar contenidors i pods.
- **Costos**:
    - No hi ha costos addicionals d'infraestructura, ja que s'utilitzen recursos propis.
    - El cost principal és el temps i esforç de manteniment i validació, així com possibles limitacions en la capacitat per proves massives.

Entorn de producció al Cloud

Aquest escenari correspon a la infraestructura real on la plataforma es desplega per gestionar els exàmens amb múltiples estudiants simultanis.

- **Característiques principals**:
    - El clúster Kubernetes s'executa en un entorn cloud, amb recursos provisionats sota demanda.
    - Permet desplegar i gestionar múltiples entorns d'examen de manera escalable, creant i eliminant namespaces i serveis dinàmicament.
    - La infraestructura està exposada i disponible per als usuaris finals durant el període de l'examen.
    - Permet escalar automàticament els recursos (CPU, memòria, nodes) segons la càrrega, optimitzant l'ús i el cost.
    - Després de la finalització de l'examen, es pot eliminar o reduir l'escala dels entorns per estalviar costos.

- **Costos**:
    - Inclou costos d'ús de computació (VMs o nodes del clúster), emmagatzematge persistent, tràfic de xarxa i altres serveis associats.
    - Els costos varien segons el nombre d'estudiants, la durada de l'examen i la càrrega dels serveis.
    - L'ús de mecanismes d'autoescalat (Horizontal Pod Autoscaler, Cluster Autoscaler) ajuda a minimitzar costos mantenint només els recursos estrictament necessaris.
    - És necessari planificar un pressupost flexible que permeti ajustar la capacitat en funció de la demanda real durant períodes puntuals.

4.4.2 Infraestructura i serveis necessaris

Entorn de testing

- **Nombre d'usuaris simultanis**: de 5 a 10 estudiants.
- **Nodes**:
    - **1 node** de mida mitjana (4 CPU i 8 GB de RAM).

  Aquest node pot allotjar tots els pods d'API, base de dades i serveis necessaris per a aquests usuaris de prova.
- **Emmagatzematge**:
    - Al voltant de 30-50 GB de volum persistent, suficient per a dades temporals, les còpies del codi per a cada estudiant de prova i les configuracions dels serveis.
- **Recursos per namespace**:
    - CPU: 0.25 - 0.5 CPU
    - Memòria: 256 - 512 MB

Entorn de producció

- **Nombre d'usuaris simultanis**: aproximadament 300 estudiants actius.
- **Nodes**:

  Caldran diversos nodes per distribuir la càrrega:

  Si considerem 0.5 CPU i 512 MB de RAM per namespace, necessitem un total aproximat de:

    - CPU: 300 * 0.5 = 150 CPUs
    - Memòria: 300 * 0.5 GB = 150 GB RAM

  Com que cap node sol oferir aquesta capacitat, es recomana un clúster amb uns 20-30 nodes de mida mitjana (8 CPU i 16 GB RAM), que permeti distribuir i balancejar la càrrega.

- **Emmagatzematge**:
    - 2-3 TB per dades persistents, repositoris i bases de dades, tenint en compte espai per còpies, logs i backups.

4.4.3. Anàlisi de costos cloud

Tenint en compte que l'entorn de testing només requereix d'un servidor mitjà de 4 CPU i 8 GB de RAM i un emmagatzematge de 30 a 50GB, no s'han considerat costos directes, ja que es pot utilitzar un servidor físic o virtual ja propietat del centre o inclòs un ordinador mitjà, tal com he desenvolupat en el meu cas el projecte.

Per analitzar els costos en l'entorn de producció, s'ha utilitzat el tarifari de preus oficial dels principals serveis de cloud actuals: Amazon Web Services, Azure Cloud, Google Cloud.

Premisses i dades per al càlcul

- **Nombre màxim de namespaces simultanis**: 300
- **Temps mitjà actiu per estudiant**: 2 hores (mínim 1h, màxim 3h)
- **Nombre d'IP públiques necessàries**:
    - 1 IP per monitoratge (Grafana)
    - 1 IP pel panell de control del sistema
    - 1 IP pels frontends dels estudiants. S'utilitza el recurs Ingress de Kubernetes, permetent especificar un subdomini per cada usuari que és rutejat a un sol servidor Proxy exposat a internet.
- **Autoscaling**:
    - La màxima càrrega es dona només en períodes puntuals (3h).
    - Els serveis cloud actuals, permeten configurar un autoescalatge de nodes que té en compte els requeriments del component Scheduler de Kubernetes, que indiquen quan es requereix un nou node per desplegar pods sol·licitats o quan un node no té pods desplegats ni es requereixen, per tal de modificar el nombre del conjunt de nodes configurats com a workers, podent així escalar el nombre de workers depenent de si es s'està realitzant l'exàmen o si no.
- **Nodes**:
    - Els nodes funcionen escalant dinàmicament segons la càrrega, assumint que per a 25 usuaris actius el sistema pot funcionar amb uns 3-4 nodes i per a la màxima càrrega amb 25-30 nodes.
- **Emmagatzematge**:
    - Emmagatzematge constant (3 TB) per a dades persistents, còpies i backups.

Costos per serveis Cloud (sense autoescalatge)

AWS (Amazon Web Services)

- **Nodes**
  - **Tipus recomanat**: m5.xlarge (4 vCPU, 16 GB RAM)
  - **Cost per node (On-Demand)**: 0,166 USD/hora aprox.
  - **Nodes necessaris**: 25 (mitjana entre 20 i 30)
  - **Cost mensual nodes**: 0,166 x 24h x 30d x 25 nodes **≈ 2.988 USD / mes**
- **Emmagatzematge**
  - **Tipus recomanat**: EBS General Purpose SSD (gp3)
  - **Cost per GB/mes**: 0,08 USD aprox.
  - **Capacitat**: 3 TB
  - **Cost mensual emmagatzematge**: 3.000 GB * 0,08 USD **≈ 240 USD / mes**
- **IPs públiques**
  - **Tipus:** Elastic IP
  - **Cost per hora per IP associada:** $0,005
  - **Cost mensual IPs:** 3 IPs x 24h x 30d x $0,005 = $10,8 / mes

Cost total aproximat sense autoscaling AWS: 3.238,8 USD / mes

Microsoft Azure

- **Nodes**
  - **Tipus recomanat**: D4s v3 (4 vCPU, 16 GB RAM)
  - **Cost per node (Pay-as-you-go)**: 0,19 USD/hora aprox.
  - **Nodes necessaris**: 25
  - **Cost mensual nodes**: 0,19 x 24h x 30d x 25 nodes ≈ **3.420 USD / mes**
- **Emmagatzematge**
  - **Tipus recomanat**: Premium SSD (P10)
  - **Cost per GB/mes**: 0,12 USD aprox.
  - **Capacitat**: 3 TB
  - **Cost mensual emmagatzematge**: 3.000 GB x 0,12 USD = **360 USD / mes**
- **IPs Públiques**
  - **Tipus:** IP pública estàtica
  - **Cost per mes d'ús per IP:** $3,65

- **Cost mensual IPs:** 3 IPs x $3,65 = $10,95 / mes

Cost total aproximat sense autoscaling Azure: 3.790,95 USD / mes

Google Cloud Platform (GCP)

- **Nodes**
  - **Tipus recomanat**: n1-standard-4 (4 vCPU, 15 GB RAM)
  - **Cost per node (On-Demand)**: 0,15 USD/hora aprox.
  - **Nodes necessaris**: 25
  - **Cost mensual nodes**: 0,15 x 24h x 30d x 25 nodes ≈ **2.700 USD / mes**
- **Emmagatzematge**
  - **Tipus recomanat**: SSD persistent disk
  - **Cost per GB/mes**: 0,10 USD aprox
  - **Capacitat**: 3 TB
  - **Cost mensual emmagatzematge**: 3.000 GB x 0,10 USD = **300 USD / mes**
- **IPs Públiques**
  - **Tipus:** IP estàtica reservada
  - **Cost per hora per IP associada:** $0,004
  - **Cost mensual IPs:** 3 IPs x 24h x 30d x $0,004 = $8,64 / mes

Cost total aproximat sense autoscaling GCP: 3.008,64 USD / mes

Costos de serveis Cloud amb Autoscaling

Per realitzar l'anàlisi de costos dels serveis Cloud amb la funcionalitat d'Autoscaling, s'ha tingut en compte un període total de 6 mesos (un semestre lectiu) on es realitzen 4 exàmens (1r parcial, 2n parcial, Final i Recuperació). Durant les hores restants del semestre, el sistema es manté actiu amb dos servidors, per mantenir emmagatzemat copies del codi a avaluar dels estudiants i el sistema de monitoratge i emmagatzematge de logs.

Núvol	Cost actiu (12 h)	Cost inactiu (4308 h)	Cost total nodes semestre
AWS	59,76 USD	1.429,66 USD	1.489,42 USD
Azure	68,4 USD	1.638,96 USD	1.707,36 USD
GCP	54 USD	1.292,4 USD	1.346,4 USD

Núvol	Cost nodes semestre	Cost fix semestre (IPs i emmagatzematge)	Cost total semestre
AWS	1.489,42 USD	1.504,8 USD	**2.994,22 USD**
Azure	1.707,36 USD	2.225,7 USD	**3.933,06 USD**
GCP	1.346,4 USD	1.851,84 USD	**3.198,24 USD**

Núvol	Cost total semestre (sense Autoscaling)	Cost total semestre (Amb autoscaling)
AWS	**19.432,8 USD**	**2.994,22 USD**
Azure	**22.745,7 USD**	**3.933,06 USD**
GCP	**18.051,84 USD**	**3.198,24 USD**

Com es pot analitzar, els costos dels serveis cloud varien però sense una diferència significativa, amb un mínim de 2.994,22 USD (AWS) i un màxim de 3.933,06 USD (Azure). L'elecció del servei Cloud dependrà també de la facilitat d'ús, servei tècnic i disponibilitat de la infraestructura.

El gran impacte diferencial es troba en l'ús de la tecnologia d'autoescalatge que permet Kubernetes, oferint una reducció de costos de més de 80% independentment del proveïdor Cloud utilitzat, reduint fins a 18.800 USD per semestre. Mostrant el potencial d'aquesta arquitectura basada en microserveis en contenidors.

4.5 Resultats i validació

Conclusions del Cas d'Ús

L'objectiu principal d'aquest cas d'ús era crear una plataforma capaç de generar subentorns dinàmics per a cada estudiant durant els seus exàmens pràctics, utilitzant Kubernetes com a eina central per gestionar aquests entorns. La solució ha demostrat ser extremadament eficaç per complir els requisits del sistema, i les conclusions obtingudes són les següents:

1. **Kubernetes com a Eina Central per la Gestió de Subentorns Dinàmics**:

   Kubernetes ha demostrat ser una eina ideal per a la creació i eliminació automàtica de subentorns. Gràcies a la seva capacitat de gestionar recursos de manera eficient, Kubernetes permet crear un namespace dedicat per a cada estudiant que s'elimina automàticament un cop finalitza l'examen. La creació de namespaces independents per a cada usuari facilita el aïllament de recursos i garanteix que cada estudiant tingui un entorn de treball net, sense interferències amb altres estudiants. A més, Kubernetes permet escalar dinàmicament els recursos (com els pods i els nodes) segons la demanda, assegurant que el sistema pot gestionar eficaçment un gran nombre d'usuaris simultanis sense sobrecarregar els recursos disponibles.

2. **Eficiència en el Desplegament i Gestió dels Subentorns**:

   L'ús de templates i automatisme en Kubernetes (mitjançant l'ús de manifestos YAML i eines com Helm i ArgoCD) ha permès crear subentorns d'estudiants de manera molt eficient. Els templates reutilitzables faciliten la replicació ràpida de configuracions per a cada examen, minimitzant els errors humans i reduint el temps necessari per desplegar els entorns. L'ús de Kustomize permet aplicar configuracions personalitzades per cada estudiant o examen, mentre que els manifests base asseguren que els entorns tinguin una configuració comuna i consistent.

3. **Automatització per Reduir la Intervenció Manual**:

   La automatització del procés a través d'API central i l'integració amb Kubernetes ha fet possible el desplegament de subentorns sense necessitat d'intervenció manual. Això no només millora l'eficiència operativa, sinó que també garanteix que els entorns estiguin configurats de manera consistent i amb mínimes errades. La creació i eliminació automàtica dels namespaces ha permès optimitzar l'ús dels recursos, ja que els entorns només s'activen durant l'examen i es destrueixen un cop finalitzat, estalviant recursos i reduint els costos operatius.

Resultats Obtinguts

El sistema ha funcionat correctament per a la gestió d'entorns dinàmics per als exàmens dels estudiants, amb els següents resultats destacats:

1. **Escalabilitat i eficiència**:
   - El sistema és capaç de gestionar una gran quantitat d'estudiants simultanis sense cap problema significatiu de rendiment, gràcies a la capacitat d'autoescalatge de Kubernetes. Durant les hores de màxima càrrega (per exemple, durant els exàmens), el sistema ha pogut adaptar-se de manera automàtica per garantir que els recursos fossin suficients per a tots els estudiants.

2. **Aïllament i Seguretat**:
   - Els entorns creats per cada estudiant estan aïllats els uns dels altres, cosa que garanteix que no hi hagi interacció no desitjada entre estudiants ni problemes de seguretat derivats de compartir recursos. Això també facilita el compliment de normatives de seguretat en entorns educatius.

3. **Costos optimitzats**:
   - Els costos operatius s'han reduït gràcies a la capacitat d'autoescalatge i a la creació d'entorns només quan és necessari. La infraestructura s'ha adaptat a la demanda en temps real, minimitzant els recursos inactius i permetent un estalvi significatiu.

4. **Temps de preparació reduït**:
   - El temps necessari per preparar els entorns d'examen ha estat excessiu per configurar o mantenir els entorns.

Millores i Recomanacions

Tot i que el sistema ha demostrat ser eficaç, sempre hi ha espai per a la millora:

- **Automatització de la neteja de recursos**: Es podria afegir més automatització per garantir que tots els recursos associats als subentorns d'estudiants es netegin correctament després de cada examen, per evitar acumulacions d'arxius o volum de dades innecessàries que podrien augmentar el cost.
- **Suport per escenaris més complexos**: Es podrien afegir més configuracions per adaptar-se a escenaris més complexos, com exàmens amb requeriments més elevats de recursos (per exemple, exàmens amb aplicacions que consumeixin més CPU o memòria) o que necessitin un accés més gran a serveis compartits.
- **Integració amb altres eines de gestió educativa**: Integrar el sistema amb altres eines educatives o d'aula virtual podria permetre una millor gestió dels exàmens i una integració més fluida de l'infraestructura amb altres serveis de l'entorn educatiu.

5. Cas d'ús 2: Organització sanitària: Aïllament de dades entre hospitals

5.1 Objectius, requisits funcionals i no funcionals

5.1.1 Objectius específics

O1: Crear una infraestructura que permeti a cada hospital tenir un entorn dedicat i aïllat per a les seves aplicacions, bases de dades i serveis, utilitzant Kubernetes per garantir l'aïllament total dels recursos de cada hospital, amb la capacitat de gestionar fins a 10 hospitals simultanis sense interferències en el rendiment ni en l'accés a dades.
O2: Garantir que els recursos assignats a cada hospital siguin utilitzats exclusivament pel seu propi entorn, mitjançant l'ús de Kubernetes per establir límits de recursos (CPU, memòria, etc.) a nivell de namespace, evitant la competència entre hospitals i assegurant un ús eficient dels recursos, amb una assignació que no permeti la sobrecàrrega d'un hospital per un altre.
O3: Optimitzar els costos operatius de la infraestructura, mitjançant l'ús de Kubernetes per ajustar dinàmicament els recursos en funció de la demanda de cada hospital, i aconseguir una reducció dels costos operatius del 20% durant el primer any d'operació.
O4: Permetre als administradors dels hospitals monitoritzar els sistemes dels seus respectius entorns a través de dashboards personalitzats a Grafana, mentre que els administradors de sistema centrals podran veure una visió global de tots els hospitals, amb la capacitat de monitoritzar el rendiment, l'ús de recursos i l'estat de les aplicacions en temps real, garantint un temps de resposta inferior a 2 minuts en l'actualització de les mètriques.

5.1.2 Requisits funcionals

Requisits Funcionals	**Objectius**
El sistema ha de permetre la creació d'entorns independents i aïllats per cada hospital, amb la capacitat de crear una nova instància de namespace per a cada hospital i aïllar les seves aplicacions, bases de dades i serveis. Els recursos no poden ser compartits entre hospitals.	O1
El sistema ha de garantir que els recursos (CPU, memòria, etc.) assignats a cada hospital siguin únicament utilitzats per aquell hospital, establint límits de recursos per cada hospital de manera independent.	O2
El sistema ha de permetre als administradors de cada hospital monitoritzar els seus propis recursos en temps real a través de dashboards personalitzats a Grafana, mentre que els administradors centrals podran veure una visió global de tots els hospitals. El monitoratge inclourà mètriques com l'ús de CPU, memòria, trànsit de xarxa i l'estat de les aplicacions.	04
El sistema ha de permetre que els recursos es gestionin automàticament en funció de la demanda del sistema. Això garantirà que els hospitals tinguin accés als recursos necessaris durant els pics de càrrega i no pateixin sobrecàrregues o pèrdues de rendiment.	O1

El sistema ha de permetre la creació, actualització i eliminació dels entorns dels hospitals de manera automatitzada mitjançant CI/CD i plantilles YAML. Aquesta automatització ha de ser flexible per a l'actualització de serveis i bases de dades sense afectar la resta de l'entorn de l'hospital.	O1
El sistema ha de garantir que tots els entorns dels hospitals compleixin amb les normatives legals relacionades amb la privacitat i seguretat de les dades, com el **Reglament General de Protecció de Dades (RGPD)**. Això inclou l'aïllament de dades per evitar contaminació entre hospitals i la gestió adequada de les dades personals dels pacients.	O2

5.1.3 Requisits no funcionals

Requisits No Funcionals	**Objectius**
El sistema ha de garantir que les dades de cada hospital estiguin aïllades i no siguin accessibles per altres hospitals, utilitzant mesures de seguretat com la xifratge de les comunicacions i l'aïllament de dades a través de namespaces de Kubernetes. Ha de complir amb les normatives legals, com el **Reglament General de Protecció de Dades (RGPD)**, per protegir la privacitat de les dades dels pacients.	O1
El sistema ha de ser capaç d'escalar automàticament per gestionar l'augment del nombre de hospitals, aplicacions i dades sense afectar el rendiment. Els recursos han de poder ser ajustats dinàmicament, garantint que el sistema pugui gestionar almenys 10 hospitals simultàniament.	O2
El sistema ha de garantir una disponibilitat mínima del 99,9% durant les hores de funcionament per evitar interrupcions en els serveis crítics. Ha de ser capaç de recuperar-se automàticament de fallades, sense impactar l'accés a les dades ni als serveis dels hospitals, per assegurar la continuïtat dels processos sanitaris.	O2
El sistema ha de garantir que les operacions de lectura i escriptura a les bases de dades i serveis dels hospitals es realitzin amb una latència inferior a 1 segon, fins i tot en moments de càrrega elevada. Les actualitzacions de recursos i els canvis en la configuració dels hospitals han de ser processats de manera eficient sense afectar el rendiment general del sistema.	O4
El sistema ha de complir amb les normatives legals i de seguretat aplicables (com el **RGPD**), garantint que les dades dels pacients es tractin de forma segura i conforme als estàndards internacionals. A més, ha de proporcionar auditories i registres d'accés i modificació de dades, per garantir la traçabilitat i la seguretat de les dades emmagatzemades.	O2

5.2 Anàlisi i disseny

5.2.1 Idea i plantejament inicial

Aquest cas d'ús s'ha concebut per donar resposta a la necessitat d'una organització sanitària que gestiona diversos hospitals a través d'una plataforma de gestió centralitzada, però que, per motius legals, de seguretat i de compliment normatiu (com el RGPD), ha de garantir que cada hospital disposi d'un entorn aïllat i independent. Aquesta separació és clau per evitar la contaminació de dades, interferències operatives entre hospitals i garantir la confidencialitat i integritat de la informació sanitària.

La idea principal ha estat dissenyar una infraestructura que permeti desplegar, per a cada hospital, un entorn dedicat que inclogui l'aplicació de gestió de l'hospital (hospital-manager), bases de dades independents (Mysql) i serveis auxiliars (Mysql Workbench), assegurant que els recursos assignats a cada hospital siguin exclusius i no puguin ser reutilitzats o interferits per altres hospitals. Aquest aïllament es tradueix en la creació de namespaces independents a Kubernetes per a cada hospital, que proporcionen una separació efectiva a nivell de recursos i seguretat.

A més, s'ha contemplat la necessitat d'un espai comú, un namespace general de **serveis centrals**, que aculli serveis compartits i que permeti consultar totes les bases de dades dels hospitals a través d'un únic servei. Aquest model centralitzat però segregat ofereix flexibilitat i facilita la gestió global, tot mantenint el control i la privacitat dels entorns individuals.

5.2.2 Estudi i anàlisi de tecnologies

Per aquest cas d'ús, s'utilitzen les mateixes tecnologies que en el cas d'ús anterior, amb adaptacions específiques per garantir l'aïllament dels hospitals i la flexibilitat del sistema.

Kubernetes

Kubernetes continua sent la tecnologia central per a la gestió de la infraestructura, ja que permet la creació de namespaces independents per a cada hospital, amb els seus propis serveis i bases de dades aïllades. Cada hospital té el seu propi namespace dins del clúster de Kubernetes, el que permet separar de manera eficient els recursos i garantir la seguretat de les dades, especialment per complir amb normatives com el RGPD (Reglament General de Protecció de Dades). A més, l'ús de Kubernetes permet una gestió automàtica de la creació i eliminació d'aquests entorns dinàmics, així com l'escalabilitat automàtica per ajustar els recursos segons la demanda de cada hospital.

Un dels usos principals dels recursos de Kubernetes en aquest cas d'ús, ha sigut la tecnologia de control d'accès basada en rols de Kubernetes (RBAC) per la gestió de permisos dels serveis documentals a les bases de dades corresponents.

RBAC (Role-Based Access Control)

RBAC (Role-Based Access Control) és una tecnologia de seguretat de Kubernetes dissenyada per gestionar i controlar l'accés als recursos dins d'un clúster. En un sistema distribuit com Kubernetes, on diversos usuaris, serveis i components poden tenir accés a una gran varietat de recursos, RBAC proporciona una manera

estructurada i flexible de controlar qui pot fer què amb els recursos del clúster. Aquesta tecnologia és fonamental per garantir la seguretat, la privacitat i la correcta segregació de les dades i serveis dins del clúster.

RBAC s'organitza en tres components principals:

- **Subjectes**: Són els usuaris o serveis (Service Account) que interactuen amb el clúster. En un sistema Kubernetes, els subjectes poden ser usuaris humans, serveis o altres entitats que utilitzen l'API del clúster per realitzar operacions.
- **Rols**: Un *Role* defineix un conjunt de permisos o accions que es poden realitzar sobre els recursos del clúster. Per exemple, un *Role* pot definir permisos per llegir o modificar secrets, crear o esborrar pods, o accedir a bases de dades. Els *Roles* poden estar limitats a un namespace o aplicables a tot el clúster.
- **RoleBindings i ClusterRoleBindings**: Un *RoleBinding* associa un *Role* amb un subjecte dins d'un namespace específic, mentre que un *ClusterRoleBinding* associa un *ClusterRole* amb un subjecte a nivell de clúster. Això permet assignar rols a usuaris o serveis específics, i així definir qui pot realitzar què en un determinat espai o a nivell global.

Funcionalitats

- **Granularitat**: RBAC ofereix un control detallat dels permisos. Pots definir qui pot accedir a quins recursos i quines accions poden realitzar (llegir, escriure, esborrar, etc.). Això ajuda a garantir que només els usuaris o serveis autoritzats tinguin accés a determinats recursos.

- **Flexibilitat**: Pots crear *roles* i assignar-los a subjectes de manera precisa. Un rol pot contenir permisos per accedir a certs recursos dins d'un namespace específic, o pot ser més ampli, permetent l'accés a recursos de tot el clúster.

- **Control d'accés basat en principis de mínim privilegi**: RBAC permet establir permisos per garantir que els usuaris i serveis només tinguin accés a allò que realment necessiten per fer la seva feina, reduint així la superfície d'atac i millorant la seguretat.

- **Facilitat d'auditoria i compliment de normatives**: Els permisos i les configuracions d'accés estan explícitament definits i són fàcilment auditables. Això facilita la revisió i la gestió de les polítiques de seguretat al llarg del temps. En entorns regulats, com el sector sanitari, on el compliment de normatives com el RGPD és essencial, RBAC permet garantir que només els usuaris o serveis autoritzats tinguin accés a les dades sensibles. Això ajuda a protegir la privacitat dels usuaris i a mantenir la conformitat amb les lleis de protecció de dades.

- **Seguretat millorada**: RBAC ajuda a evitar que usuaris no autoritzats accedeixin a recursos sensibles, com les bases de dades o els serveis

crítics, establint restriccions detallades basades en les necessitats concretes dels usuaris o serveis. Per exemple, un servei documental pot tenir permís per accedir a una base de dades de documents interns d'un hospital, però no a les bases de dades d'altres hospitals.

- **Integració amb altres tecnologies de seguretat**: RBAC es pot integrar amb altres mecanismes de seguretat, com la gestió d'identitats (IAM) i el control d'accés a nivell de xarxa, oferint una capa de seguretat addicional per garantir la seguretat global del clúster.

Prometheus i Grafana

Es segueixen utilitzant per garantir el monitoratge dels recursos del clúster i els subentorns, proporcionant una visió clara del rendiment de la infraestructura en temps real.

Amb l'ús de Grafana i Prometheus, els administradors poden tenir un control proactiu de l'estat de tots els hospitals i serveis compartits, facilitant la detecció de problemes i l'optimització dels recursos per a garantir que el sistema funcioni de manera òptima en tot moment.

Gitlab

GitLab és la plataforma escollida per al control de versions del codi i recursos de Kubernetes. Els manifests YAML i altres fitxers de configuració de Kubernetes s'emmagatzemen en un repositori GitLab, on els desenvolupadors poden gestionar les versions dels recursos, fer canvis i realitzar integració contínua (CI) i desplegament continu (CD). GitLab també facilita l'accés al codi compartit i la gestió de modificacions de recursos per garantir que tots els hospitals tinguin accés a les versions més recents dels serveis i aplicacions, mantenint la coherència de la plataforma a mesura que creix.

Bases de dades SQL

En aquest cas d'ús, la base de dades SQL s'utilitza per emmagatzemar les dades relacionals dels hospitals i els arxius compartits. Els hospitals independents tenen les seves pròpies bases de dades SQL dins dels seus namespaces, permetent un accés controlat i aïllat a les seves dades. Així, cada hospital pot tenir el seu conjunt de dades relacionades amb pacients, trànsit mèdic, etc., mentre que el namespace de serveis-centrals conté una base de dades centralitzada per a arxius comuns o dades compartides entre els hospitals. Les bases de dades SQL proporcionen una gestió eficient i fiable de les dades relacionals, amb la capacitat de realitzar consultes eficients, mantenir la coherència de les dades i garantir la seguretat. Els serveis de gestió documental també poden accedir a aquestes bases de dades per emmagatzemar i consultar arxius relacionats amb els hospitals de manera centralitzada o individual.

En aquest cas d'ús, degut al repte i dimensions que planteja, s'ha afegit una nova tecnologia per la gestió del CI/CD:

ArgoCD

ArgoCD és una eina de desplegament continu basada en el model **GitOps**, que permet gestionar i automatitzar el desplegament d'aplicacions en clústers Kubernetes a partir de repositoris Git. Aquesta tecnologia s'ha escollit per al cas d'ús de la organització sanitària per la seva capacitat de mantenir la configuració del clúster sincronitzada amb l'estat definit als repositoris, assegurant coherència, traçabilitat i automatització en la gestió dels entorns.

En aquest projecte, ArgoCD s'ha utilitzat per desplegar els diferents entorns dedicats a cada hospital com a Applications independents, controlades a la vegada per una Application genèrica que gestiona la infraestructura comuna i els recursos compartits. Aquest enfocament permet separar clarament la part estàtica de la infraestructura (serveis generals i comuns) de la part escalable i variable, que correspon als entorns específics dels hospitals.

Mitjançant l'ús de templates declaratius i reutilitzables, ArgoCD facilita la creació de manifestos Kubernetes personalitzats per a cada hospital, evitant la duplicació innecessària de codi i configuracions. Això permet desplegar ràpidament nous entorns amb les adaptacions específiques que cada hospital requereix, mantenint alhora una estructura clara i eficient.

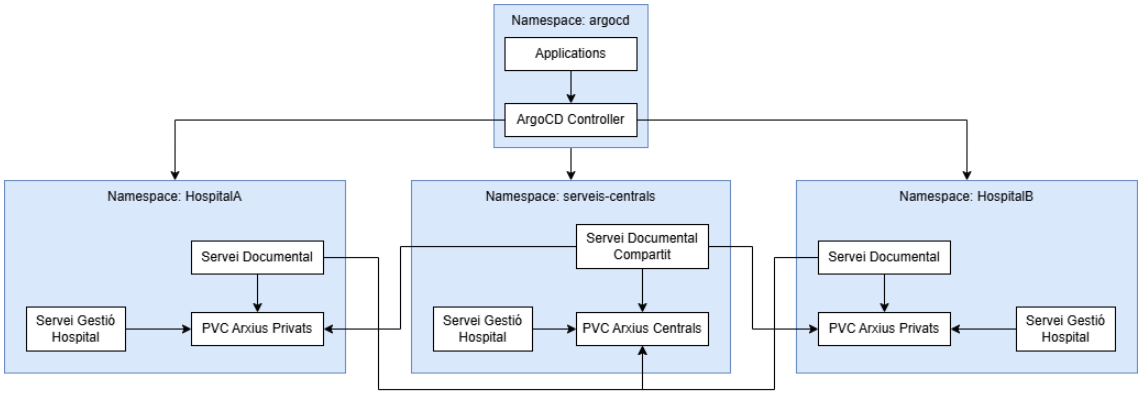
Funcionalitats:

- **Automatització i coherència**: garanteix que l'estat del clúster sigui sempre fidel al que es defineix en el repositori Git, reduint la possibilitat d'errors humans i facilitant les actualitzacions controlades.
- **Escalabilitat**: la gestió mitjançant Applications independents permet afegir o eliminar hospitals sense afectar la resta del sistema, adaptant-se a les necessitats creixents de l'organització sanitària.
- **Modularitat i reutilització**: les plantilles i manifests reutilitzables permeten personalitzar entorns de manera flexible, assegurant que les parts comunes no es repeteixin i facilitant el manteniment.
- **Control centralitzat**: mitjançant l'Application genèrica es coordinen i supervisen tots els entorns, facilitant la visió global i el control de la infraestructura compartida.

5.2.3 Disseny de l'arquitectura

1. **Arquitectura General**

L'arquitectura del cas d'ús es base en la creació de tres tipus de namespaces dins del clúster de Kubernetes: argocd, hospitalX i serveis-centrals, cadascun amb una funció específica dins de la plataforma.



- **Namespace argocd**: Aquest namespace és responsable de gestionar els recursos de ArgoCD, en aquest namespace s'hi troben les Applications per a cada servei dels hospitals, així com l'ArgoCD Controller, que s'encarrega de monitoritzar que els recursos desplegats estiguin sincronitzats amb el repositori Git. A més, aquest controlador gestiona els desplegaments automàtics dels recursos i la creació dels namespaces per a cada hospital, assegurant una gestió eficient i automatitzada de la infraestructura.
- **Namespace hospitalX**: Aquest namespace conté els recursos específics per a cada hospital (denotat aquí com a hospitalX, ja que pot haver-hi múltiples hospitals amb namespaces independents).

  Aquí es despleguen tres serveis principals:

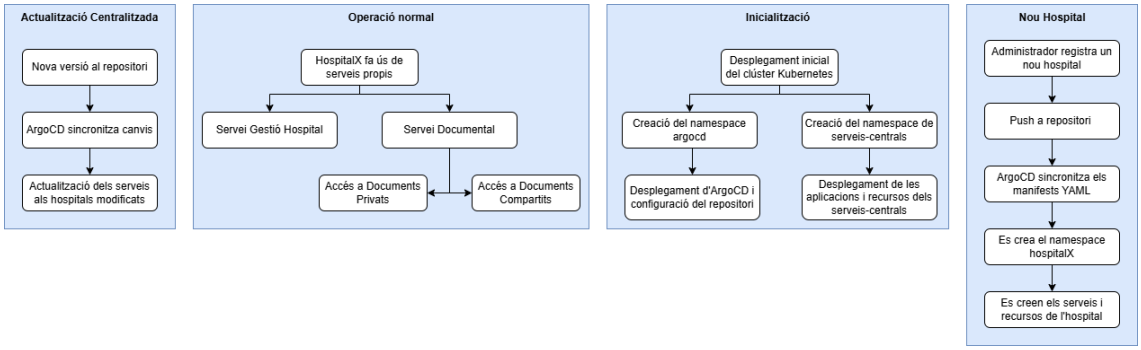
  - **Servei de gestió d'hospital**: Un sistema dedicat a gestionar les operacions internes de l'hospital, com l'administració de pacients, personal i recursos.
  - **Servei documental**: Un servei que permet gestionar documents interns, com arxius de pacients o informes. Aquest servei també pot connectar-se al PVC (Persistent Volume Claim) d'arxius centrals, permetent l'emmagatzematge i l'accés als documents de manera compartida i flexible.
  - **PVC amb arxius privats**: Cada hospital té el seu PVC per emmagatzemar arxius privats i dades específiques del hospital, com dossiers mèdics o altres documents confidencials. Aquest PVC assegura la persistència de les dades, tot mantenint l'aïllament entre hospitals.

- **Namespace serveis-centrals**: En aquest namespace es despleguen els serveis comuns i compartits entre tots els hospitals, proporcionant una capa centralitzada per la gestió dels serveis compartits.

Els serveis centrals inclouen:

- **Servei de gestió d'hospitals central**: Un sistema que permet la supervisió centralitzada de tots els hospitals, facilitant la gestió global i l'accés a dades de tots els hospitals que operen dins de la plataforma.
- **Servei documental central**: Aquest servei permet l'accés a documents compartits de manera centralitzada, podent accedir a arxius privats dels hospitals (emmagatzemats a través dels seus PVCs dedicats) o a arxius centrals que s'emmagatzemen en aquest mateix namespace. Això proporciona una única font d'accés per a arxius comunes que no requereixen un aïllament estricte.
- **PVC amb arxius compartits**: PVC on s'emmagatzema la informació que és comuna per tots els hospitals, evitant el replicat d'informació innecessari.

2. Flux de dades del sistema



En aquest cas d'ús es duu a terme quatre flux de dades diferents per les funcions que permet la plataforma:

- **Actualització Centralitzada:** Aquest flux es desencadena quan hi ha una nova versió de l'aplicació o canvis en el repositori central. Aquests canvis es poden produir, per exemple, quan es fa una actualització d'un servei o una modificació de configuració a nivell de Kubernetes.

  Passos:

  a. Quan es fa un canvi al repositori Git (per exemple, una nova versió d'un servei o una actualització de configuració), ArgoCD detecta aquest canvi.

  b. **ArgoCD sincronitza els canvis** i actualitza automàticament els recursos de Kubernetes.

  c. Els **recursos de Kubernetes es modifiquen o actualitzen**, com ara serveis, bases de dades o deployments, d'acord amb la configuració definida al repositori.

  d. La **plataforma es manté sincronitzada** en tot moment amb l'estat desitjat definit al repositori, garantint la coherència del sistema.

- **Operació Normal:** Durant l'operació normal, els hospitals utilitzen els seus serveis independents per gestionar les operacions internes, incloent l'accés als documents privats i compartits, així com l'ús dels serveis de gestió hospitalària.

Passos:

a. Els hospitals utilitzen el seu servei de gestió d'hospital per gestionar operacions internes, com l'administració de pacients, personal i recursos.

b. Els hospitals poden accedir als documents privats (emmagatzemats al PVC dins del seu propi namespace) per consultar dades sensibles de pacients.

c. També poden accedir als documents compartits (ubicats al PVC del namespace de serveis-centrals), que poden ser arxius generals que no estan restringits a un hospital específic.

d. Aquest flux permet als hospitals treballar de manera independent, però amb la capacitat de consultar recursos comuns quan sigui necessari, assegurant la seguretat de les dades i la integritat del sistema.

- **Inicialització de la Plataforma:** En aquest flux es descriu el procés de desplegament inicial de la infraestructura, des de la creació del clúster de Kubernetes fins al desplegament de l'*Application* principal d'ArgoCD, que gestiona els recursos i serveis de la plataforma.

  Passos:

  a. **Es crea el clúster de Kubernetes** i es desplega el **namespace argocd**, que contindrà tots els recursos necessaris per gestionar les aplicacions.

  b. A continuació, es desplega l'**Application principal** d'ArgoCD que sincronitza el repositori Git amb el clúster de Kubernetes.

  c. ArgoCD crea les **Applications derivades** per a cadascun dels serveis necessaris, com els serveis de gestió d'hospitals, bases de dades i serveis documentals.

  d. **Es crea el namespace serveis-centrals**, on es desplegaran les aplicacions i recursos comuns que s'utilitzaran per gestionar els serveis compartits entre els hospitals.

- **Nou Hospital:** Quan un nou hospital es vol afegir al sistema, s'ha de registrar afegint els manifests YAML corresponents al repositori Git. Aquests manifests defineixen els recursos que es desplegaran per aquest hospital.

  Passos:

  a. L'administrador del sistema afegeix els **manifests YAML** necessaris per a un nou hospital, incloent-hi configuracions per a serveis específics com la gestió hospitalària, la base de dades i el servei documental.

  b. Els manifests es **pugen al repositori Git**, i ArgoCD detecta automàticament el **push** al repositori.

  c. **ArgoCD sincronitza els recursos del clúster** amb els nous manifests YAML, creant el **namespace dedicat** per al nou hospital.

d. Es despleguen els **serveis i recursos** associats al nou hospital (com els serveis de gestió d'hospital i les bases de dades independents), i el sistema queda llest per ser utilitzat pel nou hospital.

5.3 Desenvolupament

5.3.1 Arquitectura de plantilles d'ArgoCD

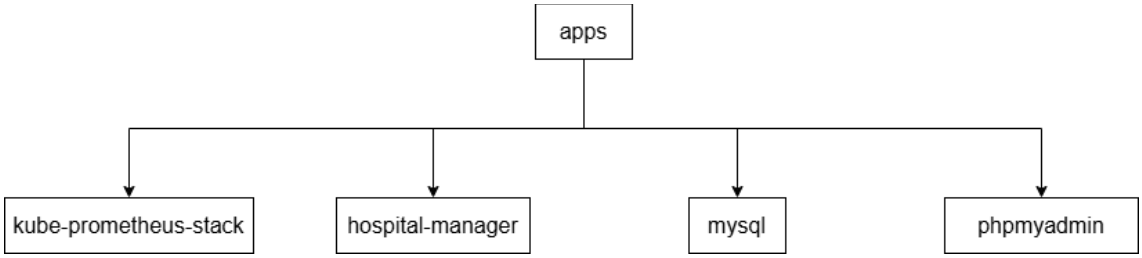
L'arquitectura de carpetes dissenyada per aquest cas d'ús està basada en un enfocament modular que permet gestionar múltiples hospitals amb entorns aïllats, garantint l'escalabilitat, la seguretat i la flexibilitat. Aquesta arquitectura es fonamenta en ArgoCD i Kustomization.yaml, que permeten gestionar i desplegar aplicacions de manera eficient i automatitzada, amb un model de gestió centralitzada però amb alta flexibilitat per personalitzar els entorns per a cada hospital.

Estructura de Carpetes

Per aquest cas d'ús s'ha generat una estructura de carpetes basades en capes utilitzant les capacitats de la tecnologia d'ArgoCD. L'estructura és la següent:

L'arquitectura de carpetes es divideix principalment en dues parts:

Apps (Aplicacions per entorns específics):

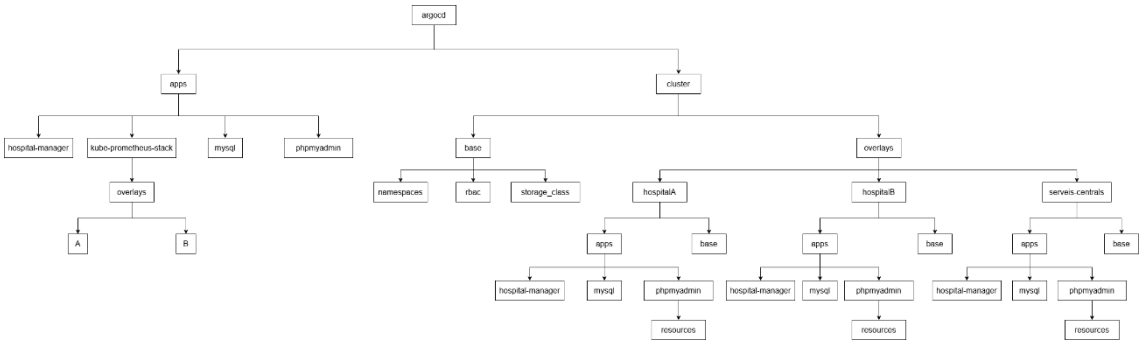


Cada aplicació està continguda dins de la carpeta apps, on s'organitzen les diferents aplicacions que es desplegaran al clúster, tant als hospitals, com el hospital-manager, la base de dades mysql, i el client phpmyadmin, com en els serveis centrals, com el sistema de monitoratge.

Cada aplicació té el seu propi kustomization.yaml que defineix la configuració específica de l'aplicació, així com els arxius associats com deployment.yaml, service.yaml i altres fitxers de configuració dels recursos de Kubernetes per l'aplicació.

Les aplicacions estan dissenyades per poder-se desplegar en qualsevol namespace que representi un hospital específic, garantint que cada hospital té el seu entorn aïllat i independent.

Argocd (Gestió centralitzada per ArgoCD):



La carpeta *argocd* conté la configuració necessària per gestionar els entorns a través d'ArgoCD. Aquí s'agrupen els kustomization.yaml i els fitxers app.yaml que defineixen les aplicacions a desplegar en els entorns específics (hospital A, hospital B) i en el namespace de serveis centrals.

El sistema es gestiona mitjançant capes per a cada hospital, permetent una personalització fàcil però mantenint les configuracions generals compartides entre tots els hospitals.

La carpeta *cluster* conté configuracions comunes per al clúster, com ara la definició de namespaces, rols d'accés (RBAC) i classes de almacenamiento, assegurant un model consistent i segur per tots els entorns.

Servei Comú de Serveis Centrals:

A part dels namespaces independents per a cada hospital, es crea un namespace comú per als serveis centrals (com la base de dades MySQL compartida i phpMyAdmin), que permet accedir a dades comunes entre hospitals de manera segura i controlada.

Aquest namespace de serveis centrals és on es centralitzen els recursos compartits, com les connexions a bases de dades, la gestió de serveis de consulta (phpMyAdmin), i altres serveis utilitzats per gestionar múltiples hospitals de manera centralitzada, evitant així duplicacions de recursos i facilitant la gestió.

El sistema proposat ofereix escalabilitat i aïllament de recursos mitjançant l'ús de namespaces independents per a cada hospital, permetent que cada entorn es pugui crear o eliminar automàticament sense afectar la resta del sistema. L'ús de Kustomization facilita la replicació i personalització ràpida dels entorns per a cada hospital, mentre que la configuració de RBAC garanteix que les dades i serveis de cada hospital estiguin totalment aïllats i accessibles només per usuaris autoritzats. A més, el model GitOps amb ArgoCD automatitza les actualitzacions i desplegaments, millorant l'eficiència operativa i reduint la complexitat de gestió centralitzada.

Aquesta arquitectura permet personalitzar fàcilment cada entorn mitjançant plantilles i capes, eliminant la necessitat de duplicar configuracions i facilitant el manteniment. A més, la creació automàtica i l'escalabilitat dels entorns asseguren una utilització òptima dels recursos, evitant el malbaratament i reduint els costos operatius a llarg termini.

5.3.2 Aplicacions desplegades

El sistema d'aplicacions d'ArgoCD permet automatitzar i gestionar el desplegament d'infraestructures en un clúster Kubernetes. Gràcies a l'ús d'ArgoCD Applications, és possible gestionar diversos entorns de manera modular i aïllada per cada hospital i els serveis centrals. En aquest cas d'ús, ArgoCD s'utilitza per desplegar els subentorns dedicats als hospitals i als serveis centrals de manera automatitzada a partir de repositoris Git. Això permet mantenir la coherència en tots els entorns, facilitar l'escalabilitat i la personalització dels serveis, i garantir que els recursos del clúster siguin gestionats de manera eficient.

ArgoCD Applications: Què són i Com Funciona?

ArgoCD és una plataforma de GitOps per a Kubernetes que gestiona els recursos i les aplicacions del clúster a partir de repositoris Git. El concepte bàsic d'ArgoCD Applications és permetre que cada servei o subentorn dins d'un clúster Kubernetes es gestioni de manera independent a través d'un manifest YAML. Aquests manifestos contenen informació sobre com s'ha de desplegar una aplicació, incloent la ubicació del codi, la configuració i el namespace de destinació.

Les ArgoCD Applications permeten la gestió descentralitzada de serveis dins d'un mateix clúster, proporcionant una capa d'abstracció per organitzar-los en subentorns independents, com és el cas dels hospitals i els serveis centrals en aquest projecte.

Desplegament d'Aplicacions a través d'ArgoCD: Exemple de YAML

A continuació es mostren tres exemples d'ArgoCD Application per als subentorns del sistema de gestió d'hospitals, amb les seves funcionalitats descrites.

a) Application per Hospital: hospitalA

Aquest primer YAML és per desplegar els recursos associats a un hospital específic. El name de l'aplicació correspon a hospitalA, i dins d'aquesta es defineixen tots els serveis específics per a aquest hospital, com el servei de gestió interna, la base de dades privada, i altres serveis relacionats.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: hospitalA
  namespace: argocd
spec:
  source:
    path: argocd/cluster/overlays/hospitalA/base
    repoURL: https://gitlab.com/ArnolSwet/tfg-arnau_cirera-
2024_2025.git
    targetRevision: testing
  destination:
    server: "https://kubernetes.default.svc"
    namespace: argocd
  project: default
```

- **name**: El nom de l'aplicació, en aquest cas hospitalA, identifica l'entorn específic d'aquest hospital.
- **source**: Indica la ubicació dels manifestos YAML al repositori Git (aquí ubicats a argocd/cluster/overlays/hospitalA/base), que conté tots els recursos necessaris per al desplegament d'aquest hospital. repoURL especifica l'URL del repositori Git, i targetRevision fa referència a la branca (testing en aquest cas) del repositori que conté els recursos.
- **destination**: Aquest camp especifica el clúster de Kubernetes on es desplegaran els recursos. En aquest cas, el servidor és https://kubernetes.default.svc, que fa referència al clúster local de Kubernetes.
- **namespace**: Es desplegarà a argocd, que és el namespace on ArgoCD gestiona les seves pròpies aplicacions i recursos.

b) Application per Serveis Centrals: serveis-centrals

Aquest segon YAML s'utilitza per gestionar el namespace dels serveis centrals, on es despleguen els recursos i serveis comuns a tots els hospitals, com la base de dades compartida i els serveis de gestió central.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: serveis-centrals
  namespace: argocd
spec:
  source:
    path: argocd/cluster/overlays/serveis-centrals/base
    repoURL:            https://gitlab.com/ArnolSwet/tfg-arnau_cirera-
2024_2025.git
    targetRevision: testing
  destination:
    server: "https://kubernetes.default.svc"
    namespace: argocd
  project: default
```

- **name**: El nom de l'aplicació és serveis-centrals, que és el namespace dedicat a tots els serveis que són compartits entre els hospitals.
- **source**: La ubicació del codi o dels recursos de serveis centrals al repositori Git. En aquest cas, els recursos per als serveis centrals estan localitzats a argocd/cluster/overlays/serveis-centrals/base.
- **destination**: Defineix on s'han de desplegar aquests recursos dins del clúster de Kubernetes. El clúster utilitzat és el local (amb el servidor https://kubernetes.default.svc).

c) Application per a un Servei Independent (MySQL per Serveis Centrals)
Aquest primer exemple és per un servei de base de dades MySQL en el namespace serveis-centrals. L'Application s'utilitza per desplegar els recursos de la base de dades MySQL, amb una configuració personalitzada mitjançant el fitxer kustomization.yaml.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: mysql-sc
  namespace: argocd
spec:
  source:
    path: argocd/cluster/overlays/serveis-centrals/apps/mysql
    repoURL:            https://gitlab.com/ArnolSwet/tfg-arnau_cirera-
2024_2025.git
    targetRevision: testing
  destination:
    server: "https://kubernetes.default.svc"
    namespace: serveis-centrals
  project: default
```

- **name: mysql-sc**: Aquest és el nom de l'Application que gestionarà el desplegament del servei de base de dades MySQL per a serveis centrals.
- **source.path**: Aquí s'especifica la ubicació dels recursos de Kubernetes per al servei de base de dades MySQL dins del repositori Git. En aquest cas, els recursos es troben a argocd/cluster/overlays/serveis-centrals/apps/mysql.
- **destination.namespace: serveis-centrals**: L'Application es desplega en el namespace serveis-centrals, on s'hi troben els recursos compartits entre els hospitals.
- **project: default**: Assigna l'Application al projecte per defecte d'ArgoCD.

d) Application amb Helm per Desplegar un Stack de Monitoratge (Prometheus)
Aquest segon exemple és per desplegar el stack de Prometheus i Grafana mitjançant Helm. Aquí es fa servir un values.yaml per personalitzar la configuració del desplegament.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: prometheus-stack
  namespace: argocd
spec:
  source:
    path: CU2/apps/kube-prometheus-stack
    repoURL:            https://gitlab.com/ArnolSwet/tfg-arnau_cirera-
2024_2025.git
    targetRevision: testing
    helm:
      valueFiles:
        - values-default.yaml
  destination:
    server: "https://kubernetes.default.svc"
```

```
  namespace: monitoring
project: default
```

- **name: prometheus-stack**: Aquest és el nom de l'Application que gestiona el desplegament del stack de Prometheus i Grafana.
- **source.path**: Aquesta ruta especifica on es troba el Helm chart per al stack de Prometheus dins del repositori Git. Els manifestos es troben a CU2/apps/kube-prometheus-stack.
- **helm.valueFiles**: Aquí es fa servir el fitxer values-default.yaml, que conté la configuració personalitzada per a Prometheus i Grafana (per exemple, límits de recursos, configuració de logs, alertes, etc.).
- **destination.namespace: monitoring**: Es desplega el servei al namespace **monitoring**, dedicat a la monitorització del clúster.
- **project: default**: Assigna l'Application al projecte per defecte d'ArgoCD.

Organització dels Recursos a través d'ArgoCD

- **Aplicació Genèrica per a Cada Subentorn**: Cada hospital i serveis centrals tenen la seva pròpia Application que es desplega i es gestiona de forma independent. Cada aplicació gestionada per ArgoCD representa un subentorn dedicat dins de Kubernetes. A través d'ArgoCD, els recursos de cada subentorn es sincronitzen automàticament amb el repositori Git per garantir que els canvis en el codi o configuracions es despleguin de manera consistent.
- **Desplegament de Diverses Applications dins de Cada Subentorn**: Cada Application (com hospitalA o serveis-centrals) pot tenir altres Applications internes que gestionen recursos específics com serveis, bases de dades, o aplicacions específiques per a cada hospital. Això permet tenir un control detallat sobre quins recursos es despleguen i actualitzen dins de cada entorn.
- **Facilitació del Desplegament Modular i Escalable**: Mitjançant la utilització de Kustomize (integrat amb ArgoCD), es poden utilitzar overlays i templates per generar múltiples entorns de manera eficient. Això facilita la personalització de cada subentorn (hospital o serveis centrals) sense duplicar configuracions ni codi, millorant l'escalabilitat i simplificant el manteniment a llarg termini.

Ús de Kustomization i PatchStrategicMerge per a Aplicacions Personalitzades

L'ús de Kustomize i les opcions de patchStrategicMerge permet aplicar aquests manifestos YAML de manera personalitzada per a cada entorn, facilitant la modularització i personalització dels recursos de Kubernetes.

Kustomization és una eina que permet crear un conjunt de manifestos YAML reutilitzables per desplegar diferents recursos en Kubernetes. Els fitxers kustomization.yaml s'utilitzen per a afegir overlays o modificacions sobre els manifests base, facilitant la gestió de configuracions específiques per cada entorn (per exemple, per un hospital concret o per serveis centrals).

Com Funciona Kustomize?

- **Kustomization de Base**: Els manifests base contenen la configuració comuna a tots els entorns. Per exemple, pots tenir un manifest base per desplegar una base de dades MySQL, amb una configuració bàsica del servei.
- **Overlays i Personalització**: Mitjançant overlays, pots aplicar configuracions específiques per a cada entorn. Per exemple, podries afegir un overlay per personalitzar el namespace de cada hospital o per modificar la configuració de recursos com la memòria o el nombre de rèpliques de serveis. Aquest overlay s'aplica sobre la configuració base.
- **patchStrategicMerge**: Aquesta estratègia permet fusionar diferents configuracions YAML i aplicar canvis específics per cada entorn. Per exemple, si tens una configuració base de MySQL, pots aplicar un patch per modificar el nom del servei o els límits de recursos només per a un hospital específic, sense haver de duplicar tot el manifest.

```
# Example of patchStrategicMerge usage in a kustomization.yaml
patchesStrategicMerge:
  - patch-file.yaml
```

5.3.2 Gestió de rols i permisos (RBAC)

El sistema de control d'accés RBAC (Role-Based Access Control) és essencial per garantir la seguretat i la gestió eficient dels permisos dins de la plataforma que gestiona múltiples hospitals i serveis centrals. A través de la creació de service accounts, Roles i RoleBindings, es controla l'accés als recursos dins de cada namespace i als recursos compartits entre hospitals. A continuació, es descriu detalladament com es configura l'accés a través de RBAC tant per a hospitals com per a serveis-centrals.

1. Configuració de RBAC per Hospitals

Quan es crea un nou hospital (en el seu propi namespace dedicat), es necessiten configuracions de RBAC per permetre que els serveis i usuaris del hospital puguin interactuar de manera controlada amb els recursos de seu namespace i amb els recursos compartits al namespace de serveis-centrals.

- **Service Account per SQL**:
  - Quan es crea un hospital, es genera un Service Account (SA) dedicat per al client de SQL dins del namespace de l'hospital. Aquest SA serà utilitzat pels serveis que necessiten accedir a la base de dades del hospital per realitzar operacions de lectura.
  - Exemple: sql-client-hospital-sa, el qual serà utilitzat per accedir a la base de dades del hospital dins del seu namespace.
- **Role per llegir serveis del namespace de l'hospital**:
  - Es crea un Role dedicat per al servei de SQL de l'hospital que li permet llegir els serveis dins del mateix namespace de l'hospital. Aquest rol garanteix que el servei SQL només pugui interactuar amb els recursos locals del seu propi namespace (com la base de dades i altres serveis interns).
  - Exemple de permisos per llegir serveis: el rol anomenat hospital-role permet accedir a serveis dins del mateix namespace, com el servei de la base de dades de l'hospital.

- **Role per llegir serveis del namespace de serveis-centrals**:
    - Els hospitals necessiten també accedir als serveis centralitzats, com el servei de la base de dades de serveis-centrals. Per això, es crea un segon RoleBinding que permet al Service Account de l'hospital obtenir els permisos per accedir a aquests serveis externs (com la base de dades central).
    - Exemple de permisos per accedir als serveis de serveis-centrals: el rol anomenat sc-role permet llegir el servei de SQL de serveis-centrals per accedir a les dades compartides (si és necessari).
- **RoleBindings**:
    - Un RoleBinding relaciona el Service Account amb els Roles especificats. El primer RoleBinding permet que el Service Account de l'hospital pugui llegir els serveis dins del seu propi namespace, i el segon RoleBinding permet l'accés als serveis de serveis-centrals.

Beneficis:

- **Aïllament i Seguretat**: Els serveis de cada hospital tenen accés restringit a només aquells recursos dins del seu namespace, mentre que poden accedir als serveis comuns de serveis-centrals de manera controlada.
- **Control de Permisos Granular**: A través de l'ús de Roles i RoleBindings, els permisos són gestionats amb detall, permetent un accés a mesura i evitant l'accés no autoritzat als recursos d'altres hospitals o de serveis centrals.

2. Configuració de RBAC per els Serveis Centrals

Els serveis centrals tenen un conjunt diferent de permisos, ja que han de gestionar recursos comuns i compartits entre tots els hospitals, com les bases de dades centrals. Aquests serveis necessiten un accés ampli, tant als serveis del clúster com als serveis de les bases de dades compartides.

- **Service Account per SQL de Serveis-Centrals**:
    - Es crea un Service Account (SA) per al client de SQL dels serveis centrals. Aquest SA s'utilitza per gestionar les connexions a la base de dades compartida per a tots els hospitals i altres serveis que requereixin accedir a la base de dades central.
    - Exemple: sql-client-sc-sa, que permet als serveis centrals accedir a la base de dades comuna per a la gestió dels arxius compartits entre els hospitals.
- **ClusterRole per Accedir a Recursos del Clúster**:
    - Un ClusterRole és creat per atorgar permís a l'Service Account dels serveis centrals per accedir als recursos del clúster complet. Aquest rol permet realitzar operacions bàsiques com llegir serveis i interactuar amb recursos a nivell de clúster.
    - Exemple: cluster-role permet accedir a recursos de tot el clúster (com secrets, configmaps, services, etc.).

- **ClusterRoleBinding per Associar el ClusterRole al Service Account**:
    - Un ClusterRoleBinding es crea per associar el ClusterRole amb el Service Account dels serveis centrals, garantint que aquest servei tingui els permisos necessaris per gestionar els recursos de tot el clúster i interactuar amb els serveis de tots els hospitals, com la base de dades compartida.
    - **Exemple de binding**: sc-cluster-rolebinding, que connecta el service-account dels serveis centrals amb el ClusterRole creat per permetre l'accés als recursos del clúster.

Beneficis:

- **Accés Global i Centralitzat**: Els serveis centrals tenen accés a tota la infraestructura del clúster per gestionar serveis compartits, com la base de dades central, mentre que cada hospital manté els seus recursos aïllats.
- **Seguretat de les Dades**: Es garanteix que només els serveis centrals tinguin accés als recursos compartits, mentre que cada hospital té accés restringit als seus propis serveis i dades.
- **Gestió Centralitzada de Permisos**: L'ús de ClusterRole i ClusterRoleBinding permet una gestió més senzilla i centralitzada de permisos a nivell de clúster, sense necessitat de configurar rols individualment per a cada servei dins del clúster.

5.4 Anàlisi de costos i escalabilitat

5.4.1 Escenaris d'ús

En aquest cas d'ús, hem estipulat dos escenaris per tal de subdividir el desplegament en un entorn de desplegament i un entorn de producció:

Entorn de test

Aquest entorn és una infraestructura de desenvolupament i test que s'executa localment, utilitzant un clúster Kubernetes desplegat mitjançant Docker Desktop.

- **Característiques principals**:
    - No està exposat a internet, garantint un entorn segur i controlat per a proves i desenvolupament.
    - Els recursos (CPU, memòria, emmagatzematge) depenen directament del hardware local on s'executa Docker Desktop, sense costos associats a serveis cloud.
    - Permet provar funcionalitats, desplegar i eliminar entorns ràpidament, i validar el funcionament abans d'arribar a producció.
    - L'escalabilitat està limitada pels recursos del maquinari local i la capacitat de Docker Desktop per gestionar contenidors i pods.

- **Costos**:
    - No hi ha costos addicionals d'infraestructura, ja que s'utilitzen recursos propis.
    - El cost principal és el temps i esforç de manteniment i validació, així com possibles limitacions en la capacitat per proves massives.

Entorn de producció al Cloud

Aquest escenari correspon a la infraestructura real on la plataforma es desplega per gestionar els diferents hospitals tant de forma individual com centralitzada.

- **Característiques principals**:
    - El clúster Kubernetes s'executa en un entorn cloud, amb recursos provisionats sota demanda.
    - Permet desplegar i gestionar múltiples entorns de manera escalable, creant i eliminant namespaces i serveis dinàmicament.
    - Permet escalar automàticament els recursos (CPU, memòria, nodes) segons la càrrega, optimitzant l'ús i el cost.
    - Durant els períodes d'inactivitat o baixa activitat dels hospitals, es pot eliminar o reduir l'escala dels entorns per estalviar costos.
- **Costos**:
    - Inclou costos d'ús de computació (VMs o nodes del clúster), emmagatzematge persistent, tràfic de xarxa i altres serveis associats.
    - Els costos varien segons el nombre d'hospitals i la càrrega dels serveis.
    - L'ús de mecanismes d'autoescalat (Horizontal Pod Autoscaler, Cluster Autoscaler) ajuda a minimitzar costos mantenint només els recursos estrictament necessaris.
    - És necessari planificar un pressupost flexible que permeti ajustar la capacitat en funció de la demanda real durant períodes puntuals.

5.4.2 Infraestructura i serveis necessaris

Entorn de testing

- **Nombre d'usuaris simultanis**: de 5 a 10 hospitals.
- **Nodes**:
    - 1 node de mida mitjana (4 CPU i 8 GB de RAM).

    Aquest node pot allotjar tots els pods d'API, base de dades i serveis necessaris per a aquests usuaris de prova.

- **Emmagatzematge**:
    - Al voltant de 30-50 GB de volum persistent, suficient per a dades temporals, les còpies del codi per a cada estudiant de prova i les configuracions dels serveis.
- **Recursos per namespace**:
    - CPU: 0.25 - 0.5 CPU
    - Memòria: 256 - 512 MB

Entorn de producció

- **Nombre d'usuaris simultanis**: aproximadament 300 hospitals actius.
- **Nodes**:

  Caldran diversos nodes per distribuir la càrrega:

  Si considerem 0.5 CPU i 512 MB de RAM per namespace, necessitem un total aproximat de:

    - CPU: 300 * 0.5 = 150 CPUs
    - Memòria: 300 * 0.5 GB = 150 GB RAM

  Com que cap node sol oferir aquesta capacitat, es recomana un clúster amb uns 20-30 nodes de mida mitjana (8 CPU i 16 GB RAM), que permeti distribuir i balancejar la càrrega.
- **Emmagatzematge**:
    - 2-3 TB per dades persistents i configuracions d'aplicacions, tenint en compte espai per còpies, arxius i backups.

5.4.3 Anàlisis de costos

Entorn de Testing

El entorn de testing s'utilitza principalment per a proves de desenvolupament i es desplega amb un únic subentorn dedicat a un hospital. Aquest entorn requereix menys recursos, ja que només s'activa durant els períodes de desenvolupament o proves. El servidor utilitzat per aquest entorn pot ser un servidor físic o virtual propietat de l'empresa o un ordinador mitjà, similar al que es fa servir per al desenvolupament del projecte.

Premisses per al càlcul de costos del testing:

- **Servidor necessari**: Un servidor mitjà amb 4 CPU i 8 GB de RAM, amb un emmagatzematge de 30 a 50 GB.
- **Costos directes**: No es consideren costos directes, ja que es pot utilitzar un servidor existent dins de l'organització o un ordinador dedicat al desenvolupament.

Aquest entorn, en no requerir un desplegament constant, no genera costos operatius significatius més enllà de l'emmagatzematge per les dades de les proves.

Entorn de Producció

L'entorn de producció és el que gestionarà els serveis operatius dels hospitals, amb una infraestructura basada en Kubernetes que permet l'autoescalatge dinàmic segons la demanda. Aquest entorn serà principalment sempre actiu i gestionarà un nombre creixent de hospitals. Cada hospital es desplega com un subentorn dins de Kubernetes, amb namespaces independents per a cada hospital. A mesura que es creïn més hospitals, el sistema haurà d'escalar per gestionar la càrrega addicional.

Premisses per al càlcul de costos de producció:

- **Nombre màxim de hospitals (namespaces)**: Inicialment, es preveu un màxim de 300 hospitals (namespaces) en funcionament simultàniament.
- **Escalabilitat**: A mesura que el sistema creix, es preveu que cada any s'afegeixi 1 hospital nou.
- **Autoescalatge**: Durant les hores no actives (quan no s'estan realitzant proves o exàmens), el sistema podrà reduir la capacitat de nodes en funcionament i escalar automàticament quan sigui necessari.
- **Nodes**: El sistema necessitarà entre 20 i 30 nodes durant el seu funcionament habitual, i es poden reduir a 2 nodes durant les hores inactives (sense càrrega).

Costos Estimats per al Núvol (AWS, Azure i GCP)

Els costos es basen en un entorn de producció amb autoescalatge per nodes, de manera que la infraestructura es manté operativa a una capacitat mínima quan no hi ha usuaris actius, però pot escalar automàticament quan es necessita més capacitat.

AWS (Amazon Web Services)

- **Nodes**:
    - **Tipus de node**: m5.xlarge (4 vCPU, 16 GB RAM).
    - **Cost per node (On-Demand)**: 0,166 USD/hora.
    - **Nodes necessaris (mitjana)**: 25 nodes.
    - **Cost mensual nodes**: 0,166 USD x 24h x 30d x 25 nodes ≈ 2.988 USD/mes.
- **Emmagatzematge**:
    - **Tipus de disc**: EBS General Purpose SSD (gp3).
    - **Cost per GB/mes**: 0,08 USD/GB.
    - **Capacitat**: 3.000 GB.
    - **Cost mensual emmagatzematge**: 3.000 GB * 0,08 USD ≈ 240 USD/mes.
- **IPs Públiques**:
    - **Cost per hora per IP**: 0,005 USD.
    - **Cost mensual IPs**: 3 IPs x 24h x 30d x 0,005 USD = 10,8 USD/mes.

- **Cost Total Estimat (sense autoescalatge)**: 3.238,8 USD/mes.
- **Cost Total amb Autoescalatge (producció)**: 2.994,22 USD/mes.

Azure

- **Nodes**:
  - **Tipus de node**: D4s v3 (4 vCPU, 16 GB RAM).
  - **Cost per node (Pay-as-you-go)**: 0,19 USD/hora.
  - **Nodes necessaris (mitjana)**: 25 nodes.
  - **Cost mensual nodes**: 0,19 USD x 24h x 30d x 25 nodes ≈ 3.420 USD/mes.
- **Emmagatzematge**:
  - **Tipus de disc**: Premium SSD (P10).
  - **Cost per GB/mes**: 0,12 USD/GB.
  - **Capacitat**: 3.000 GB.
  - **Cost mensual emmagatzematge**: 3.000 GB x 0,12 USD = 360 USD/mes.
- **IPs Públiques**:
  - **Cost per mes per IP**: 3,65 USD.
  - **Cost mensual IPs**: 3 IPs x 3,65 USD = 10,95 USD/mes.
- **Cost Total Estimat (sense autoescalatge)**: 3.790,95 USD/mes.
- **Cost Total amb Autoescalatge (producció)**: 3.933,06 USD/mes.

Google Cloud Platform (GCP)

- **Nodes**:
  - **Tipus de node**: n1-standard-4 (4 vCPU, 15 GB RAM).
  - **Cost per node (On-Demand)**: 0,15 USD/hora.
  - **Nodes necessaris (mitjana)**: 25 nodes.
  - **Cost mensual nodes**: 0,15 USD x 24h x 30d x 25 nodes ≈ 2.700 USD/mes.
- **Emmagatzematge**:
  - **Tipus de disc**: SSD persistent disk.
  - **Cost per GB/mes**: 0,10 USD/GB.
  - **Capacitat**: 3.000 GB.
  - **Cost mensual emmagatzematge**: 3.000 GB x 0,10 USD = 300 USD/mes.
- **IPs Públiques**:
  - **Cost per hora per IP**: 0,004 USD.

- **Cost mensual IPs**: 3 IPs x 24h x 30d x 0,004 USD = 8,64 USD/mes.

- **Cost Total Estimat (sense autoescalatge)**: 3.008,64 USD/mes.
- **Cost Total amb Autoescalatge (producció)**: 3.198,24 USD/mes.

Anàlisi de Costos amb Autoescalatge (Producció)

Premisses:

- Durant **les hores inactives**, es mantindran **només 2 nodes** en funcionament per garantir la disponibilitat mínima per al sistema de monitoratge i emmagatzematge de logs.
- **Escalabilitat automàtica**: Quan el sistema estigui en ple ús, com durant l'examen, els recursos es poden escalar fins a un màxim de **25-30 nodes** per gestionar la càrrega addicional dels hospitals.
- **Escalabilitat en el temps**: A mesura que el sistema afegeix més hospitals cada any, el nombre de **nodes** i **namespaces** haurà d'escalar automàticament per gestionar la càrrega.

Càlculs:

1. **Cost nodes semestre** (12 hores d'ús actiu):
    - AWS: 1.489,42 USD
    - Azure: 1.707,36 USD
    - GCP: 1.346,4 USD
2. **Cost fix semestre (IPs i emmagatzematge)**:
    - AWS: 1.504,8 USD
    - Azure: 2.225,7 USD
    - GCP: 1.851,84 USD
3. **Cost total semestre amb autoscaling**:
    - AWS: 2.994,22 USD
    - Azure: 3.933,06 USD
    - GCP: 3.198,24 USD

5.5 Resultats i validació

Conclusions del Cas d'Ús

L'objectiu principal d'aquest cas d'ús era crear una plataforma que permetés gestionar una infraestructura multihospital on cada hospital tingués els seus serveis i dades aïllats per complir amb normatives com el RGPD i protegir-se davant de possibles filtracions de dades, però que al mateix temps, els serveis comuns (com la base de dades centralitzada o els serveis de gestió) fossin gestionats de manera centralitzada per facilitar la seva administració i optimització de recursos.

Les conclusions obtingudes en aquest cas d'ús són molt positives, ja que gràcies a l'ús de Kubernetes, ArgoCD, RBAC i altres eines de gestió de recursos, hem aconseguit un sistema que compleix amb els objectius de seguretat, aïllament, escalabilitat i eficiència operativa. A continuació, es descriuen els resultats obtinguts.

Aïllament i Seguretat dels Hospitals

- **Namespace Independent per a Cada Hospital**: Gràcies a Kubernetes, cada hospital es gestiona dins d'un namespace dedicat, el que garanteix que els serveis, dades i bases de dades d'un hospital estan aïllats de les dades i serveis d'altres hospitals. Aquesta separació ajuda a garantir que les dades de pacients i altres informació sensible es mantinguin protegides dins de cada hospital, complint amb les normatives de seguretat com el RGPD.
- **Gestió d'Accés amb RBAC:** Mitjançant RBAC (Role-Based Access Control), es defineixen rols i permisos específics per cada hospital i servei. Això garanteix que només els usuaris autoritzats tinguin accés als recursos del seu hospital i, en el cas dels serveis centrals, als recursos compartits de manera controlada. Això permet un control d'accés molt detallat, protegint les dades sensibles dels pacients i altres serveis crítics.

Gestió Centralitzada de Serveis Comuns

Tot i l'aïllament entre hospitals, la gestió centralitzada dels serveis és una part fonamental del sistema. Els serveis compartits com la base de dades comuna o els serveis de gestió centralitzada s'han gestionat al namespace de serveis-centrals. Això permet als hospitals utilitzar recursos comuns com la base de dades centralitzada de manera eficient, però mantenint alhora l'aïllament necessari per complir amb les normatives.

Kubernetes i ArgoCD permeten gestionar aquests serveis centrals de manera centralitzada, amb l'ús d'ArgoCD Applications que controlen els recursos que es despleguen a nivell de clúster. Gràcies a ArgoCD, és possible desplegar serveis com la base de dades SQL de manera centralitzada, però garantir que l'accés sigui només per a les parts autoritzades (com el servei de gestió d'hospitals).

A mesura que s'afegiran més hospitals, l'ús de Kustomize permetrà escalar i personalitzar fàcilment els recursos i serveis per cada hospital, sense la necessitat de duplicar tota la infraestructura. Això fa que el sistema sigui fàcil d'expandir, mantenint una gran eficiència operativa.

Desplegament Modular i Personalitzat Amb ArgoCD i Kustomize

L'ús de Kustomize i ArgoCD ha permès dissenyar un sistema de desplegament modular que facilita la replicació i personalització de l'entorn de cada hospital. Mitjançant templates i overlays, cada hospital pot tenir una infraestructura completament independent (amb els seus serveis i dades), però compartint configuracions comunes per a serveis comuns com la base de dades central o la gestió centralitzada. Cada Application de ArgoCD es personalitza mitjançant els fitxers de configuració a Kustomize, permetent que cada hospital tingui els seus serveis personalitzats (com la base de dades local o serveis de gestió interna). Això permet a cada hospital configurar els seus serveis i recursos d'una manera independent, mantenint la coherència a nivell global. Gràcies a ArgoCD, el procés de desplegament i actualització de recursos és totalment automatitzat. Quan es fa un canvi en el repositori Git, ArgoCD sincronitza automàticament els canvis amb el clúster de Kubernetes, assegurant que tots els entorns i serveis estiguin actualitzats sense necessitat d'intervenció manual.

Resultats Obtinguts

- **Compliment de Normatives**: La creació de namespaces independents per a cada hospital i l'ús de RBAC ha permès garantir que les dades dels pacients i altres serveis crítics siguin segurs i privats, complint amb normatives com el RGPD.
- **Gestió Eficaç de Serveis Comuns**: Els serveis centrals, com la base de dades SQL comuna, s'han gestionat de manera centralitzada, permetent un accés controlat per part dels hospitals sense comprometre la seguretat ni la segregació de dades.
- **Escalabilitat i Flexibilitat**: El sistema ha demostrat ser escalable, ja que s'ha pogut afegir nous hospitals i serveis sense dificultats, gràcies a l'ús de Kustomize i la creació d'overlays per a cada hospital o servei.
- **Eficiència Operativa**: El sistema ha demostrat gran eficiència operativa en la gestió dels serveis i en l'ús de recursos, especialment gràcies a l'ús d'ArgoCD i Kustomize, que automatitzen el desplegament i actualitzacions dels recursos.

Millores i Recomanacions

Tot i que el sistema ha complert amb els requisits inicials, sempre hi ha marge per a la millora:

1. **Automatització de la Netegera de Recursos**:
    - Implementar un sistema més automatitzat per eliminar recursos innecessaris o subentorns d'hospitals quan no siguin necessaris, per minimitzar el malbaratament de recursos i reduir els costos operatius.
2. **Més Integració amb altres Sistemes Sanitaris**:
    - La integració amb altres sistemes de gestió hospitalària (per exemple, sistemes d'històrics de pacients o de gestió de cites) pot millorar encara més la capacitat de gestionar la informació d'un hospital de manera eficient.

6. Cas d'ús 3: Start-up de contingut multimèdia online

6.1 Objectius, requisits funcionals i no funcionals

6.1.1 Objectius específics

O1: Desenvolupar una plataforma que proporcioni contingut multimèdia amb un temps de resposta inferior a 10 segons a milions d'usuaris simultàniament de diverses ubicacions.
O2: Desenvolupar un sistema de monitorització en temps real per permetre a l'empresa controlar el rendiment dels entorns de la plataforma, utilitzant eines com Prometheus i Grafana per recopilar mètriques de rendiment (ús de CPU, memòria, trànsit de xarxa) i generar alertes automàtiques en cas d'anomalies, garantint una resposta ràpida amb un temps de latència inferior a 1 minut en la detecció de problemes.
O3: Implementar una infraestructura autoescalable que permeti gestionar els pics d'ús durant el llançament de contingut popular sense interrupcions o pèrdues de qualitat, escalant automàticament els recursos segons la demanda, amb la capacitat de manejar un augment del tràfic de fins a 5 vegades més de la capacitat habitual.

6.1.2 Requisits funcionals

Requisits Funcionals	**Objectius**
El sistema ha de ser capaç de proporcionar contingut multimèdia (pel·lícules, sèries) amb un temps de resposta inferior a 10 segons per a milions d'usuaris simultàniament. Aquesta capacitat s'ha de garantir mitjançant una infraestructura distribuïda, amb diversos punts de presència (PoPs) per reduir la latència i garantir un rendiment òptim.	O1
El sistema ha de permetre als administradors monitoritzar el rendiment dels serveis de la plataforma, utilitzant eines per recopilar dades de rendiment (ús de CPU, memòria, trànsit de xarxa) i visualitzar-les en panells de control. A més, ha de generar alertes automàtiques en temps real si es detecten anomalies en els recursos, amb un temps de resposta a les alertes inferior a 1 minut.	02
El sistema ha de ser capaç d'escalar automàticament els recursos en funció de la demanda del trànsit, per gestionar els pics de trànsit sense interrupcions. El sistema ha de ser capaç de gestionar un augment de trànsit fins a 5 vegades la capacitat habitual de manera automàtica i transparent per als usuaris.	O3
El sistema ha de permetre distribuir el trànsit de manera eficient entre diversos **punts de presència (PoPs)** arreu del món, gestionant les sol·licituds d'entrada i equilibrant la càrrega entre els nodes disponibles, reduint la latència i millorant l'experiència d'usuari en tots els països.	O1

6.1.3 Requisits no funcionals

Requisits no Funcionals	Objectius
El sistema ha de garantir que el temps de resposta de la plataforma sigui inferior a 10 segons per a milions d'usuaris simultanis, fins i tot durant els pics de trànsit elevat. Això ha d'incloure la capacitat de gestionar de manera eficient les peticions de contingut multimèdia, utilitzant **caching** i la distribució de contingut per proximitat geogràfica mitjançant punts de presència (PoPs).	O1
El sistema ha de ser capaç de gestionar un augment del trànsit de fins a 5 vegades més la capacitat habitual sense provocar interrupcions ni pèrdues de qualitat. El sistema ha de ser autoescalable, activant nous recursos (pods, nodes) automàticament segons la demanda, amb un temps de resposta de l'escalat inferior a 1 minut.	O3
El sistema ha de garantir una disponibilitat del 99,9% o superior durant les hores de funcionament, per assegurar que els usuaris no experimentin interrupcions.	O3
El sistema ha de permetre la monitorització contínua dels serveis de la plataforma en temps real, amb l'ús de **Prometheus** i **Grafana** per obtenir mètriques de rendiment com l'ús de CPU, memòria i trànsit de xarxa. Les alertes automàtiques han d'estar configurades per identificar qualsevol degradació del servei o anomalies, amb un temps de resposta a les alertes inferior a 1 minut.	O2
El sistema ha de ser capaç d'optimitzar els costos operatius mitjançant la distribució eficient de recursos i l'escalat d'infraestructura per reduir els costos operatius en almenys un 25% respecte a les configuracions tradicionals.	O3

6.2 Anàlisi i disseny

6.2.1 Idea i plantejament inicial

Aquest cas d'ús es focalitza en una start-up que gestiona una plataforma global de contingut multimèdia on milions d'usuaris poden accedir simultàniament a pel·lícules i sèries. La característica principal i repte de la plataforma és gestionar eficientment els pics de tràfic, especialment durant llançaments de contingut popular que provoquen una demanda molt elevada i puntual, amb la necessitat de garantir un servei fluït, amb temps de resposta mínims i sense pèrdues de qualitat o interrupcions.

La idea de l'arquitectura s'ha basat en separar clarament la part comuna i estàtica de la plataforma de la part dinàmica i personalitzada per a cada usuari. Així, s'ha dissenyat un namespace centralitzat que conté el frontend on els usuaris poden navegar i seleccionar contingut, la base de dades centralitzada de pel·lícules i sèries, i una API que actua com a motor principal, gestionant les peticions i el desplegament dels entorns personalitzats.

Els namespaces independents es creen dinàmicament per a cada usuari que inicia la visualització d'una pel·lícula o capítol, allotjant un visualitzador multimèdia dedicat i una base de dades local on s'emmagatzemen dades específiques d'aquesta sessió (puntuació,

temps de visualització, preferències, la pel·lícula a visualitzar, etc.). Aquest model assegura aïllament, permet personalitzar l'experiència de l'usuari i optimitza el rendiment.

A diferència del cas d'ús anterior, aquests subentorns no es despleguen mitjançant ArgoCD, sinó que són gestionats directament per l'API central que crea i elimina aquests espais de treball de forma eficaç segons l'ús, assegurant una escalabilitat immediata i una gestió dinàmica dels recursos.

El flux de dades es desenvolupa de la següent manera: quan un usuari selecciona un contingut al frontend, l'API central crea el subentorn personalitzat, clona la pel·lícula o capítol des de la base de dades centralitzada al namespace de l'usuari i posa en marxa el visualitzador. Quan l'usuari acaba la sessió, el subentorn es destrueix automàticament per alliberar recursos.

6.2.2 Estudi i anàlisi de tecnologies

Per al cas d'ús de la start-up de contingut multimèdia, s'han utilitzat les mateixes tecnologies ja descrites en els casos d'ús anteriors, amb algunes adaptacions per adaptar-se als requisits específics de la plataforma d'aquest cas d'ús.

Kubernetes

Continua sent l'eina central per a la gestió dels múltiples entorns d'aplicacions (en aquest cas, les instàncies de visualització de contingut per a cada usuari). Kubernetes permet la creació d'entorns aïllats per a cada usuari (namespace per cada sessió de visualització), la qual cosa és essencial per garantir que cada sessió de visualització sigui independent i segura, evitant interferències entre usuaris i gestionant els recursos de manera eficaç. A més, Kubernetes facilita l'automatització del desplegament i la creació d'aquests entorns segons la demanda, així com l'escalabilitat dinàmica, adaptant els recursos a mesura que el tràfic augmenta o disminueix durant els llançaments de contingut popular.

Prometheus i Grafana

Es mantenen com a eines per al monitoratge dels recursos del clúster, oferint visibilitat en temps real sobre l'ús de CPU, memòria i trànsit de xarxa. Aquestes eines permeten monitoritzar l'eficiència del sistema durant els pics de tràfic, identificar possibles anomalies i ajustar els recursos segons sigui necessari, millorant l'optimització de costos. A més, Grafana proporciona visualitzacions clares i útils que ajuden als administradors a comprendre millor l'estat de la infraestructura i detectar qualsevol problema en les seves aplicacions.

Gitlab

És utilitzat per al control de versions, gestionant el codi font i els recursos de Kubernetes per a les aplicacions que es despleguen en els entorns dels usuaris. Amb GitLab, el codi de l'API central i els templates de Kubernetes es poden gestionar i actualitzar de manera senzilla, garantint que els canvis es puguin aplicar automàticament en el sistema de manera segura i coherent. Això és fonamental per mantenir la coherència entre el codi i els recursos de la plataforma, assegurant que qualsevol nova actualització es desplegui ràpidament i sense errors.

Python i Flask

És l'elecció per al desenvolupament de l'API central de la plataforma, que gestiona la creació i l'eliminació dels entorns de reproducció de contingut per a cada usuari. L'API de Flask es fa càrrec de gestionar les peticions per crear un entorn d'usuari (namespace), clonar el contingut multimèdia des de la base de dades central i proporcionar els recursos necessaris per reproduir el contingut seleccionat per l'usuari. Python és el llenguatge utilitzat per desenvolupar aquesta API, aprofitant la seva simplicitat i potència per integrar-se amb altres serveis i recursos del sistema.

Kubernetes plugin i pyyaml

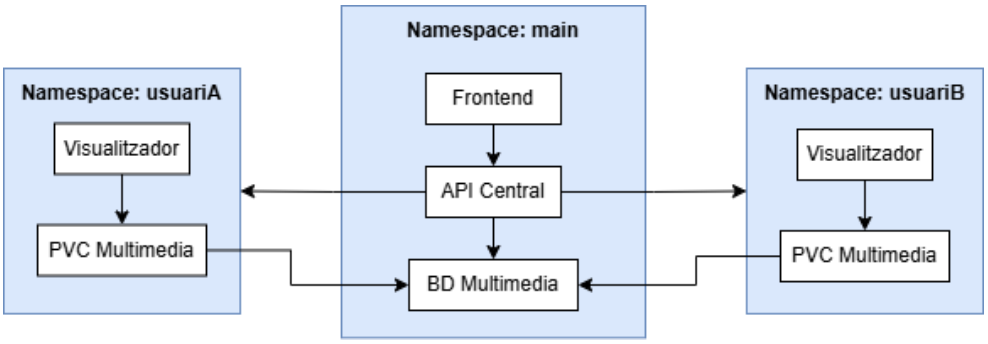
El plugin de Kubernetes de Python permet automatitzar el desplegament dels entorns de reproducció a partir de l'API central. Aquest plugin interactua amb el clúster de Kubernetes per crear els namespaces necessaris, gestionar els PVCs i controlar els recursos assignats a cada entorn. Aquesta integració fa possible la creació automàtica d'entorns independents per a cada usuari, amb la configuració adequada de l'emmagatzematge necessari per a les dades de visualització.

SQL

És utilitzat per gestionar la base de dades multimèdia central, que emmagatzema tot el catàleg de contingut (pel·lícules, sèries, episodis) i les metadades associades. Aquesta base de dades permet a l'API consultar les dades multimèdia i transferir-les als entorns de reproducció dels usuaris quan seleccionen un contingut per veure. La base de dades SQL és essencial per mantenir una estructura organitzada i eficient per a l'accés a grans volums de dades.

6.2.3 Disseny de l'arquitectura

1. **Arquitectura General**



En aquest cas d'ús, la plataforma de contingut multimèdia està dissenyada per gestionar l'accés a pel·lícules i sèries per part dels usuaris a través dels subentorns dins de Kubernetes. Aquests subentorns, permeten aïllar i gestionar els recursos de manera eficient, tant per a la visualització del contingut com per al servei centralitzat que gestiona el catàleg i els recursos globals. A continuació, es descriuen les dues subarquitectures principals de la plataforma.

1. Namespace: usuariX

Aquest namespace conté els recursos específics per a un usuari individual que està visualitzant un contingut multimèdia (per exemple, una pel·lícula o un capítol d'una sèrie). Cada usuariX té el seu propi entorn aïllat on es gestiona l'experiència de visualització del contingut.

Components:

- **Reproductor/Visualitzador**: Una aplicació dedicada a reproduir el contingut multimèdia seleccionat per l'usuari. Aquest component és el que interactua directament amb l'usuari per mostrar el contingut i permetre la interacció (play, pausa, etc.).
- **PVC (Persistent Volume Claim)**: Un volum persistent on s'emmagatzema el contingut multimèdia que l'usuari està visualitzant. Els arxius es copien des de la base de dades central al PVC del namespace usuariX, el que garanteix que cada usuari tinguin accés només al contingut que ha seleccionat per visualitzar.

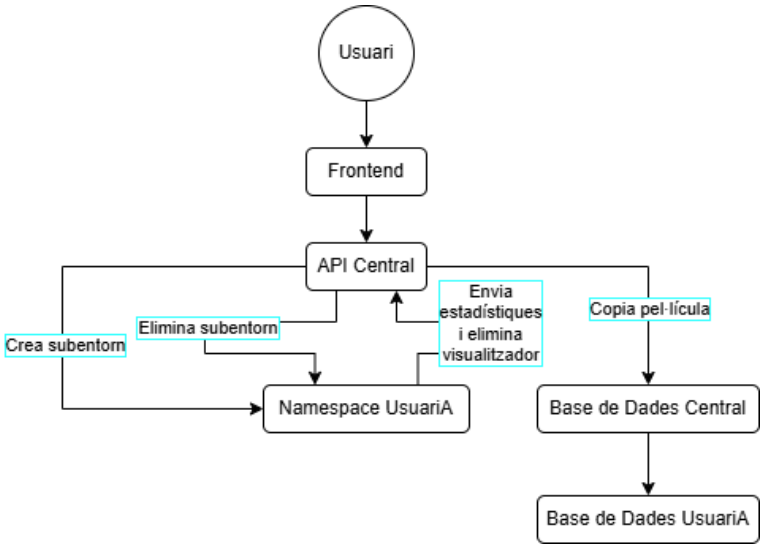
2. Namespace: main

El namespace main conté els recursos centrals de la plataforma que gestionen el catàleg multimèdia, les operacions dels usuaris i la generació d'entorns per la visualització del contingut. Aquest és el punt d'entrada per als usuaris, on poden navegar pel catàleg, seleccionar el contingut a visualitzar i interactuar amb l'API central per crear els subentorns específics per a cada usuari.

Components:

- **Frontend**: Aplicació web on els usuaris poden veure el catàleg de pel·lícules i sèries disponibles, seleccionar un títol i iniciar la visualització. Aquest frontend està estretament integrat amb l'API per mostrar el contingut i gestionar la selecció.
- **API Central**: API que gestiona les consultes a la base de dades de contingut multimèdia i l'intercanvi d'informació entre els usuaris i els serveis de la plataforma. Quan un usuari selecciona un contingut, l'API crea un subentorn dedicat a l'usuari, amb el reproductor i l'emmagatzematge associat. L'API també s'encarrega de copiar l'arxiu de la base de dades central al PVC del subentorn de l'usuari.
- **Base de dades SQL**: Base de dades relacional on es guarden els detalls de tot el contingut multimèdia (pel·lícules, sèries, capítols). La base de dades també emmagatzema metadades associades a cada arxiu (per exemple, títols, descripcions, puntuacions, etc.). Aquesta base de dades és centralitzada i es consulta a través de l'API per proporcionar informació al frontend i per gestionar la selecció de contingut.

2. Flux de dades del sistema



El flux de dades en aquest sistema segueix una seqüència ben definida que garanteix una experiència d'usuari fluïda i un ús eficient dels recursos, tot mantenint la seguretat i l'aïllament de les dades de cada usuari.

1. Entrada de l'usuari al Frontend: L'usuari accedeix a la pàgina principal de la plataforma, on es mostra el catàleg de pel·lícules i sèries disponibles. Aquest catàleg és gestionat per una API central, que consulta la base de dades SQL on es guarden totes les metadades del contingut multimèdia (títols, descripcions, classificacions, etc.).

2. Selecció de Contingut i Creació de Subentorn: L'usuari selecciona una pel·lícula o capítol de sèrie que vol visualitzar. Quan l'usuari selecciona un contingut, es realitza una petició a l'API central, la qual rep la identificació del contingut seleccionat i l'usuari que el visualitzarà. L'API central crea un namespace dedicat per a l'usuari (usuariX), amb el nom de l'usuari o una identificació única. Aquest subentorn allotja tot el que es necessita per a la visualització: el reproductor, els recursos del sistema i la base de dades associada. A continuació, l'API central utilitza Kubernetes per desplegar els serveis necessaris en aquest namespace, incloent el reproductor multimèdia i el Persistent Volume Claim (PVC) per allotjar l'arxiu multimèdia. El contingut seleccionat es copia des de la base de dades SQL central al PVC del namespace de l'usuari (per exemple, es clona la pel·lícula o capítol al PVC específic de l'usuari).

3. Visualització del Contingut: Un cop creat el subentorn, el reproductor multimèdia es carrega dins del seu namespace i comença a reproduir el contingut seleccionat. El reproductor, allotjat dins del namespace de l'usuari, accedeix al contingut emmagatzemat al PVC, i el mostra a l'usuari a través de l'aplicació frontend del subentorn.

4. Finalització de la Reproducció i Eliminació de l'Entorn: Quan l'usuari acaba de veure la pel·lícula o capítol, el reproductor multimèdia genera una senyal per finalitzar la sessió i envia aquesta informació a l'API central. L'API central rep la sol·licitud d'eliminació del subentorn, i procedeix a eliminar el namespace de l'usuari. Això inclou la destrucció del PVC associat, el tancament del reproductor multimèdia i la lliberació de recursos. Així, els recursos es destrueixen automàticament per optimitzar l'ús de la infraestructura i evitar el malbaratament de recursos.

6.3 Desenvolupament

6.3.1 Desplegament inicial de la Plataforma

El procés de desplegament inicial de la plataforma multimèdia es realitza mitjançant un script automatitzat que s'executa en el clúster Kubernetes. Aquest script gestiona la creació dels recursos essencials per a la infraestructura de la plataforma, com el monitoratge, els serveis del catàleg multimèdia i la base de dades de contingut. A continuació, es descriu detalladament el que fa aquest script.

1. Creació del namespace de monitoratge

```
# Create the monitoring namespace

kubectl create namespace monitoring
```

Aquest primer pas crea un namespace dedicat anomenat monitoring, el qual serà utilitzat per allotjar tots els serveis relacionats amb el monitoratge (com Prometheus i Grafana). Els namespaces permeten separar de manera eficient els diferents serveis dins del mateix clúster de Kubernetes, evitant que entrin en conflicte entre ells i mantenint-los aïllats.

2. Desplegament de Prometheus amb Helm

```
#Deploy Prometheus stack with helm

# Add the prometheus-community helm repo

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

# Install the values.yaml file

helm upgrade --install -f k8s/deployments/prometheus-values.yaml kube-prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring
```

El desplegament de Prometheus es fa mitjançant Helm, que és una eina per gestionar paquets de Kubernetes. Aquí es fa el següent:

- S'afegeix el repositori de Prometheus per poder utilitzar els charts de Helm.
- S'actualitzen els repositoris disponibles.
- Es desplega el stack de Kube-Prometheus-Stack (que inclou Prometheus i Grafana) utilitzant un fitxer prometheus-values.yaml, on es defineixen les configuracions necessàries per al desplegament dins del namespace monitoring.

3. Creació del namespace principal

```
# Create main namespace

kubectl create namespace main
```

Aquesta instrucció crea un namespace principal anomenat main, on es desplegaran tots els serveis relacionats amb el catàleg.

4. Creació de la imatge Docker i desplegament del panell de control

```
# Build and push control panel image

docker build -t arnaucirera98/cataleg ./cataleg

docker push arnaucirera98/cataleg:latest
```

Aquest pas construeix i puja la imatge del panell de control al repostiri públic de Docker Hub. Això permetrà crear un recurs de Kubernetes de tipus Deployment, que descarregui aquesta imatge per desplegar-la en un pod.

També es pot pujar en un repositori de contenidors propi o privat i configurar les credencials d'accés a Kubernetes per tal de que pugui accedir-hi, però en aquest projecte ho he volgut fer amb el repositori de Docker públic per facilitar el desplegament i configuració d'aquestes imatges i mantenir el projecte públic.

4. Desplegament de la Base de Dades MySQL

A continuació, l'script desplega la base de dades MySQL, que és utilitzada per emmagatzemar les pel·lícules i sèries disponibles al catàleg. Aquesta base de dades conté una taula on s'emmagatzemen les metadades de cada pel·lícula o sèrie, incloent el títol, descripció, durada i l'arxiu multimèdia associat.

`kubectl apply -f k8s/configs/mysql-config.yaml -n main`

Aplica la configuració de la base de dades MySQL, establint els paràmetres generals de connexió i configuració.

`kubectl apply -f k8s/configs/mysql-secret.yaml -n main`

Aplica el secret de MySQL, que emmagatzema les credencials d'accés a la base de dades de manera segura.

`kubectl apply -f k8s/deployments/mysql-deployment.yaml -n main`

Desplega el contenidor de la base de dades MySQL al namespace main. Aquest contenidor allotja la base de dades amb la taula que conté tota la informació sobre el catàleg de pel·lícules i sèries.

`kubectl apply -f k8s/services/mysql-service.yaml -n main`

Crea un servei de tipus ClusterIP per exposar la base de dades MySQL als altres serveis dins del namespace main.

Taula de Base de Dades MySQL:

La taula de la base de dades MySQL contindrà les següents columnes:

- **id**: Clau primària, un identificador únic per a cada pel·lícula o episodi.
- **títol**: El nom de la pel·lícula o capítol de sèrie.
- **descripció**: Una breu descripció del contingut.
- **durada**: La durada de la pel·lícula o episodi.
- **arxiu**: Fitxer multimèdia que conté la pel·lícula o episodi.

5. Desplegament del Catàleg Multimèdia

El següent pas és el desplegament del servei catàleg multimèdia. Aquest servei és el que permet als usuaris veure el catàleg, seleccionar contingut i iniciar la visualització. El catàleg es desplega en un contenidor que allotja l'API Flask i el frontend de la plataforma.

```
docker build -t arnaucirera98/catalog ./catalog
```

Aquesta línia construeix la imatge Docker del catàleg multimèdia, que inclou l'API Flask i el frontend per gestionar la selecció de contingut per part de l'usuari.

```
docker push arnaucirera98/catalog:latest
```

S'envia la imatge Docker al registre de contenidors per a la seva utilització en el desplegament del servei.

```
kubectl apply -f k8s/deployments/catalog-deployment.yaml -n main
```

Desplega l'API Flask i el frontend dins del namespace main per gestionar les consultes del catàleg.

```
kubectl apply -f k8s/services/catalog-service.yaml -n main
```

Crea un servei per exposar l'API Flask i el frontend del catàleg a través d'un ClusterIP.

6.3.2 Catàleg central

L'API desenvolupada en Flask per a la plataforma multimèdia gestiona el procés de visualització de contingut per part dels usuaris, així com la creació i la gestió dels entorns d'usuari per visualitzar pel·lícules i sèries. Aquesta API fa ús de diverses tecnologies, com la connexió amb una base de dades MySQL per recuperar informació de les pel·lícules, la creació de namespaces dinàmics dins de Kubernetes per allotjar els entorns de reproducció, i la manipulació d'arxius multimèdia per a la seva visualització. A continuació, es descriuen les funcions i operacions que realitza l'API.

1. Ruta Principal: /

La ruta principal de l'API és la pàgina index, que serveix el catàleg multimèdia. Quan un usuari accedeix a aquesta ruta, es mostra una interfície on pot veure el catàleg de pel·lícules i sèries disponibles per ser visualitzades.

- **Funció**: Retorna el frontend del catàleg on l'usuari pot explorar el contingut multimèdia (pel·lícules, sèries, episodis, etc.).

2. Ruta per Consultar el Catàleg: /api/catalog

Aquesta ruta connecta l'API amb la base de dades **MySQL**, recuperant el catàleg de pel·lícules i sèries per mostrar-los a l'usuari. Les pel·lícules i sèries són recuperades mitjançant una consulta SQL que selecciona el títol, descripció i durada del contingut.

Funció:

- Recupera els detalls de les pel·lícules i sèries del catàleg des de la base de dades SQL.
- Exposa la informació en format **JSON** per ser consumida per l'frontend.

3. Ruta per Crear l'Entorn de Visualització: /api/visualization

Quan un usuari selecciona una pel·lícula o capítol de sèrie per visualitzar, aquesta ruta crea un namespace dedicat per a l'usuari i desplega un entorn de reproducció. Es realitza una sèrie de passos:

Funció:

- **Creació del namespace**: Un namespace dedicat és creat per allotjar l'entorn de visualització (reproductor multimèdia) i la base de dades associada.
- **Còpia del contingut**: El contingut multimèdia seleccionat es recupera de la base de dades MySQL i es copia al PVC associat al namespace creat. L'API també s'encarrega de crear el pod per copiar el contingut multimèdia dins del PVC de l'usuari.
- **Desplegament del reproductor multimèdia**: Un cop el contingut és copiat, es desplega el reproductor multimèdia per a l'usuari dins del subentorn creat.

6.3.3 Visualitzador

El visualitzador és un component clau en aquesta arquitectura, ja que permet a cada usuari visualitzar un contingut multimèdia en el seu propi subentorn creat a Kubernetes. Aquest procés es gestiona a través de Kubernetes amb un namespace dedicat per a cada usuari i diversos recursos associats per garantir una experiència de visualització aïllada, eficient i de qualitat. El visualitzador fa ús d'un contenidor VLC Media Player dins d'un pod Kubernetes, que s'encarrega de reproduir el contingut multimèdia copiat des de la base de dades central al PVC associat al namespace de l'usuari. A continuació, es detalla com funciona i què recursos es despleguen per a fer-ho possible.

Cada usuari que accedeix a la plataforma de contingut multimèdia genera un namespace dedicat dins de Kubernetes. Aquest namespace és aïllat per cada usuari i conté els recursos necessaris per visualitzar el contingut multimèdia seleccionat. L'ús d'aquesta separació garanteix que cada usuari tingui el seu propi entorn de reproducció, evitant qualsevol interferència amb altres usuaris.

Components del namespace:

- **Persistent Volume Claim (PVC)**: El PVC és el volum persistent que emmagatzema l'arxiu multimèdia que l'usuari ha seleccionat per veure. Quan un usuari selecciona una pel·lícula o capítol de sèrie, l'API central copia l'arxiu de la base de dades central a aquest PVC dins del namespace dedicat.
- **Reproductor multimèdia (Visualitzador)**: El reproductor és un contenidor dins del pod de Kubernetes que s'encarrega de reproduir el contingut multimèdia utilitzant VLC Media Player.

El visualitzador és un servei que utilitza el contenidor de VLC Media Player per reproduir el contingut multimèdia emmagatzemat al PVC associat al namespace de l'usuari. Aquest component és gestionat mitjançant un Deployment de Kubernetes, que s'encarrega de desplegar i mantenir el contenidor de reproducció de manera eficient. El LoadBalancer s'utilitza per exposar aquest servei a l'usuari.

Flux de dades:

- Quan l'usuari selecciona un contingut al catàleg, l'API central crea el namespace dedicat, genera el PVC per allotjar el contingut i, posteriorment, desplega el pod que conté el visualitzador.
- VLC Media Player en el contenidor accedeix al PVC per obtenir l'arxiu multimèdia i iniciar la reproducció.
- El pod de reproducció pot estar configurat perquè VLC llegeixi el fitxer des del PVC i el mostri en temps real a l'usuari mitjançant una interfície de reproducció adequada.

6.4 Anàlisi de costos i escalabilitat

6.4.1 Escenaris d'ús

En aquest cas d'ús, hem estipulat dos escenaris per tal de subdividir el desplegament en un entorn de desplegament i un entorn de producció:

Entorn de test

Aquest entorn és una infraestructura de desenvolupament i test que s'executa localment, utilitzant un clúster Kubernetes desplegat mitjançant Docker Desktop.

- **Característiques principals**:
    - No està exposat a internet, garantint un entorn segur i controlat per a proves i desenvolupament.
    - Els recursos (CPU, memòria, emmagatzematge) depenen directament del hardware local on s'executa Docker Desktop, sense costos associats a serveis cloud.
    - Permet provar funcionalitats, desplegar i eliminar entorns ràpidament, i validar el funcionament abans d'arribar a producció.
    - L'escalabilitat està limitada pels recursos del maquinari local i la capacitat de Docker Desktop per gestionar contenidors i pods.
- **Costos**:
    - No hi ha costos addicionals d'infraestructura, ja que s'utilitzen recursos propis.
    - El cost principal és el temps i esforç de manteniment i validació, així com possibles limitacions en la capacitat per proves massives.

Entorn de producció al Cloud

Aquest escenari correspon a la infraestructura real on la plataforma es desplega per gestionar les peticions dels usuaris i generar els diferents subentorns de visualització.

- **Característiques principals**:
    - El clúster Kubernetes s'executa en un entorn cloud, amb recursos provisionats sota demanda.
    - Permet desplegar i gestionar múltiples entorns de manera escalable, creant i eliminant namespaces i serveis dinàmicament.
    - Permet escalar automàticament els recursos (CPU, memòria, nodes) segons la càrrega, optimitzant l'ús i el cost.
    - Durant els períodes d'inactivitat o baixa activitat dels hospitals, es pot eliminar o reduir l'escala dels entorns per estalviar costos.
- **Costos**:
    - Inclou costos d'ús de computació (VMs o nodes del clúster), emmagatzematge persistent, tràfic de xarxa i altres serveis associats.
    - Els costos varien segons el nombre d'usuaris i la càrrega dels serveis.
    - L'ús de mecanismes d'autoescalat (Horizontal Pod Autoscaler, Cluster Autoscaler) ajuda a minimitzar costos mantenint només els recursos estrictament necessaris.
    - És necessari planificar un pressupost flexible que permeti ajustar la capacitat en funció de la demanda real durant períodes puntuals.

6.4.2 Infraestructura i serveis necessaris

Entorn de testing

- **Nombre d'usuaris simultanis**: de 5 a 10 usuaris.
- **Nodes**:
    - 1 node de mida mitjana (4 CPU i 8 GB de RAM).

    Aquest node pot allotjar tots els pods d'API, base de dades i serveis necessaris per a aquests usuaris de prova.
- **Emmagatzematge**:
    - Al voltant de 30-50 GB de volum persistent, suficient per a dades temporals, les còpies del codi per a cada estudiant de prova i les configuracions dels serveis.
- **Recursos per namespace**:
    - CPU: 0.25 - 0.5 CPU
    - Memòria: 256 - 512 MB

Entorn de producció

- **Nombre d'usuaris simultanis**: aproximadament 300 usuaris actius.
- **Nodes**:

  Caldran diversos nodes per distribuir la càrrega:

  Si considerem 0.5 CPU i 512 MB de RAM per namespace, necessitem un total aproximat de:

  - CPU: 300 * 0.5 = 150 CPUs
  - Memòria: 300 * 0.5 GB = 150 GB RAM

  Com que cap node sol oferir aquesta capacitat, es recomana un clúster amb uns 20-30 nodes de mida mitjana (8 CPU i 16 GB RAM), que permeti distribuir i balancejar la càrrega.
- **Emmagatzematge**:
  - 2-3 TB per dades persistents i contingut multimedia, tenint en compte espai per còpies, arxius i backups.

6.4.3 Anàlisi de costos

En aquest cas d'ús, la startup de contingut multimèdia té un model basat en la visualització de pel·lícules i sèries. Els subentorns es generen de manera dinàmica quan un usuari selecciona un contingut per visualitzar, i aquests es poden eliminar un cop finalitza la visualització. Això implica que el cost de recursos serà més variable, amb una baixa càrrega durant les hores de baixa activitat (on no hi ha visualitzacions en curs), i una alta càrrega en períodes de màxima demanda, com durant els llançaments de contingut popular o estrenes.

A continuació es realitza una anàlisi detallada dels costos per a la infraestructura en un escenari normal i en un escenari de màxima demanda.

Premisses i Dades per al Càlcul

- **Subentorns i Escalabilitat**:
  - Els subentorns es generen dinàmicament només quan es visualitza contingut. Això vol dir que només es consumiran recursos durant les hores de visualització (quan un usuari està veient una pel·lícula o sèrie).
  - En hores de baixa activitat (quan no s'està visualitzant contingut), el sistema no consumeix gaire recursos, ja que es redueix la necessitat de nodes.
- **Usuari Mitjà**:
  - Una mitjana de 100 usuaris diaris visualitzant contingut en hores punta.
- **Moment d'Estrenes**:
  - En moments d'estrenes o llançaments de contingut popular, la demanda pot augmentar fins a 1000 usuaris en un dia.

- **Temps mitjà de visualització**: Aproximadament 2 hores per usuari.
- **Nodes i Autoscaling**:
    - En la configuració normal, el sistema utilitza entre 3 i 4 nodes per gestionar fins a 100 usuaris simultanis.
    - En moment d'estrenes o en períodes de màxima demanda, es pot arribar a utilitzar entre 25-30 nodes per suportar l'augment de la càrrega d'usuaris.
- **Emmagatzematge**:
    - 3 TB d'emmagatzematge per a dades persistents, còpies i backups.
- **IPs Públiques**:
    - Una IP per al panell de control.
    - Una IP per al monitoratge (Grafana).
    - Una IP per al frontend (Ingress a Kubernetes) que gestiona els subdominis dels usuaris.

Costos per als Serveis Cloud (AWS, Azure, GCP)

Els costos es calcularan per als nodes necessaris en diferents escenaris (actiu vs inactiu) i tenint en compte els recursos d'emmagatzematge i IPs públiques. A continuació es detalla l'anàlisi de costos per als principals proveïdors de serveis cloud.

1. AWS (Amazon Web Services)

- **Nodes**:
    - **Tipus de node**: m5.xlarge (4 vCPU, 16 GB RAM).
    - **Cost per node (On-Demand)**: 0,166 USD/hora.
    - **Nodes necessaris**:
        - **Escenari normal** (100 usuaris): 3-4 nodes.
        - **Escenari màxim d'estrenes** (1000 usuaris): 25-30 nodes.
    - **Cost mensual nodes (normal)**: 0,166 USD * 24h * 30d * 3 nodes ≈ **358,56 USD/mes**.
    - **Cost mensual nodes (estrenes)**: 0,166 USD * 24h * 30d * 25 nodes ≈ **2.988 USD/mes**.
- **Emmagatzematge**:
    - **Tipus de disc**: EBS General Purpose SSD (gp3).
    - **Cost per GB/mes**: 0,08 USD/GB.
    - **Capacitat**: 3.000 GB.
    - **Cost mensual emmagatzematge**: 3.000 GB * 0,08 USD = **240 USD/mes**.

- **IPs Públiques**:
    - **Cost per hora per IP**: 0,005 USD.
    - **Cost mensual IPs**: 3 IPs x 24h x 30d x 0,005 USD = **10,8 USD/mes**.
- **Cost Total Estimat (sense autoescalatge)**: **3.238,8 USD/mes**.
- **Cost Total amb Autoescalatge (producció)**:
    - Escenari normal (100 usuaris actius): **2.994,22 USD/mes**.
    - Escenari màxim d'estrenes (1000 usuaris actius): **2.994,22 USD/mes**.

2. Microsoft Azure

- **Nodes**:
    - **Tipus de node**: D4s v3 (4 vCPU, 16 GB RAM).
    - **Cost per node (Pay-as-you-go)**: 0,19 USD/hora.
    - **Nodes necessaris**:
        - **Escenari normal** (100 usuaris): 3-4 nodes.
        - **Escenari màxim d'estrenes** (1000 usuaris): 25-30 nodes.
    - **Cost mensual nodes (normal)**: 0,19 USD * 24h * 30d * 3 nodes ≈ **410,4 USD/mes**.
    - **Cost mensual nodes (estrenes)**: 0,19 USD * 24h * 30d * 25 nodes ≈ **3.420 USD/mes**.
- **Emmagatzematge**:
    - **Tipus de disc**: Premium SSD (P10).
    - **Cost per GB/mes**: 0,12 USD/GB.
    - **Capacitat**: 3.000 GB.
    - **Cost mensual emmagatzematge**: 3.000 GB * 0,12 USD = **360 USD/mes**.
- **IPs Públiques**:
    - **Cost per mes per IP**: 3,65 USD.
    - **Cost mensual IPs**: 3 IPs * 3,65 USD = **10,95 USD/mes**.
- **Cost Total Estimat (sense autoescalatge)**: **3.790,95 USD/mes**.
- **Cost Total amb Autoescalatge (producció)**:
    - Escenari normal (100 usuaris actius): **3.933,06 USD/mes**.
    - Escenari màxim d'estrenes (1000 usuaris actius): **3.933,06 USD/mes**.

3. Google Cloud Platform (GCP)

- **Nodes**:
  - **Tipus de node**: n1-standard-4 (4 vCPU, 15 GB RAM).
  - **Cost per node (On-Demand)**: 0,15 USD/hora.
  - **Nodes necessaris**:
    - **Escenari normal** (100 usuaris): 3-4 nodes.
    - **Escenari màxim d'estrenes** (1000 usuaris): 25-30 nodes.
  - **Cost mensual nodes (normal)**: 0,15 USD * 24h * 30d * 3 nodes ≈ **324 USD/mes**.
  - **Cost mensual nodes (estrenes)**: 0,15 USD * 24h * 30d * 25 nodes ≈ **2.700 USD/mes**.
- **Emmagatzematge**:
  - **Tipus de disc**: SSD persistent disk.
  - **Cost per GB/mes**: 0,10 USD/GB.
  - **Capacitat**: 3.000 GB.
  - **Cost mensual emmagatzematge**: 3.000 GB * 0,10 USD = **300 USD/mes**.
- **IPs Públiques**:
  - **Cost per hora per IP**: 0,004 USD.
  - **Cost mensual IPs**: 3 IPs x 24h x 30d x 0,004 USD = **8,64 USD/mes**.
- **Cost Total Estimat (sense autoescalatge)**: **3.008,64 USD/mes**.
- **Cost Total amb Autoescalatge (producció)**:
  - Escenari normal (100 usuaris actius): **3.198,24 USD/mes**.
  - Escenari màxim d'estrenes (1000 usuaris actius): **3.198,24 USD/mes**.

Càlcul de Costos per Tot un Semestre

Per calcular els costos de la infraestructura durant un semestre complet, considerarem que la plataforma es mantindrà operativa durant 4 exàmens, amb un total de 12 hores d'ús intens i el sistema funcionarà durant 4308 hores restants (quan el sistema estigui en baixa càrrega).

Cost de Nodes durant el Semestre (Autoescalatge):

Escenari normal (100 usuaris actius):

- **AWS**: 1.489,42 USD (nodes) + 1.504,8 USD (emmagatzematge i IPs) = **2.994,22 USD**.
- **Azure**: 1.707,36 USD (nodes) + 2.225,7 USD (emmagatzematge i IPs) = **3.933,06 USD**.

- **GCP**: 1.346,4 USD (nodes) + 1.851,84 USD (emmagatzematge i IPs) = **3.198,24 USD**.

Escenari d'estrenes (1000 usuaris actius):

- **AWS**: 2.988 USD (nodes) + 1.504,8 USD (emmagatzematge i IPs) = **4.492,8 USD**.
- **Azure**: 3.420 USD (nodes) + 2.225,7 USD (emmagatzematge i IPs) = **5.645,7 USD**.
- **GCP**: 2.700 USD (nodes) + 1.851,84 USD (emmagatzematge i IPs) = **4.551,84 USD**.

6.5 Resultats i validació

Conclusions del Cas d'Ús

L'objectiu d'aquest cas d'ús era desenvolupar una plataforma multimèdia capaç de generar de manera ràpida i eficient subentorns de visualització de contingut per als usuaris, de manera que cada usuari pogués visualitzar pel·lícules o sèries de manera independent, sense afectar la qualitat de la visualització per la resta d'usuaris, independentment de la quantitat de connexions simultànies.

Després de la implementació del sistema utilitzant Kubernetes, s'han obtingut les següents conclusions:

- **Kubernetes com a Solució Escalable**: Kubernetes ha demostrat ser **molt útil** per gestionar l'escalabilitat i la creació d'entorns dinàmics de visualització de contingut. A través de namespaces dedicats i l'ús de pods efímers, Kubernetes permet crear subentorns només quan són necessaris, minimitzant el malbaratament de recursos i mantenint l'escalabilitat del sistema.
- **Autoescalatge Eficaç**: L'ús de l'autoescalatge dins de Kubernetes ha estat un gran avantatge en el sistema, ja que ha permès gestionar l'augment de la càrrega de manera automàtica durant els moments de màxima demanda (com en els llançaments de contingut o en períodes de màxima visualització). Els entorns s'han creat i eliminat automàticament, reduint la necessitat de gestionar manualment els recursos i optimitzant l'ús de la infraestructura.
- **Aïllament i Seguretat**: Cada subentorn creat per cada usuari és totalment aïllat, garantint que les dades i l'experiència de visualització d'un usuari no interfereixin amb les d'altres usuaris. Aquest aïllament és fonamental per garantir que no hi hagi conflictes ni problemes de seguretat durant la visualització.
- **Optimització de Recursos**: Kubernetes ha permès optimitzar els recursos utilitzats durant els períodes d'inactivitat, ja que els subentorns es destrueixen automàticament un cop l'usuari acaba de veure el contingut. Aquesta funcionalitat ha ajudat a minimitzar els costos i ha evitat el malbaratament de recursos.

Resultats Obtinguts

Els resultats obtinguts en la implementació de la plataforma multimèdia han estat molt positius, amb els següents punts destacats:

1. **Escalabilitat Dinàmica**: El sistema ha estat capaç de gestionar un gran nombre d'usuaris simultanis de manera eficaç. Durant els períodes de màxima càrrega (per exemple, durant l'estrena de contingut popular), Kubernetes ha pogut escalar automàticament per garantir que els recursos fossin suficients per a tots els usuaris. La capacitat d'escalar de manera dinàmica ha demostrat ser un gran avantatge en aquest cas.

2. **Aïllament de les Sessions d'Usuari**: Cada usuari té el seu propi subentorn de visualització totalment aïllat, garantint la seguretat i la privacitat de les dades de cada usuari durant la seva sessió. Això ha millorat l'experiència d'usuari, evitant possibles conflictes entre sessions o interferències en el contingut visualitzat.

3. **Optimització en l'ús de la infraestructura**: Gràcies a l'autoescalatge i la creació i eliminació automàtica dels subentorns, els recursos s'han utilitzat de manera eficient, només consumint els recursos necessaris durant les hores de visualització del contingut. Això ha ajudat a reduir els costos operatius i a millorar la gestió de recursos.

4. **Experiència d'Usuari Fluïda**: L'usuari ha pogut accedir al contingut de manera àgil i sense interrupcions, gràcies a l'ús de Kubernetes per gestionar el cicle de vida dels subentorns i garantir que els recursos estiguessin disponibles quan fos necessari.

Millores i Recomanacions

Tot i que la solució basada en Kubernetes ha sigut eficaç, es poden identificar algunes àrees de millora:

- **Eines més lleugeres per a subentorns efímers**: Encara que Kubernetes és una eina potent per gestionar entorns dinàmics i escalables, potser no és la solució més lleugera per a un sistema tan simple com el de visualització de contingut. Per entorns efímers tan senzills, tecnologies serverless com AWS Lambda poden resultar més òptimes i econòmiques. AWS Lambda permet executar codi de manera dinàmica i efímera sense necessitat de gestionar nodes ni contenidors, i és especialment útil per a processos com la creació i eliminació d'entorns sense necessitat d'una infraestructura de clúster com Kubernetes.
- **Millora en la gestió de recursos durant períodes inactius**: Tot i que Kubernetes pot desactivar nodes durant les hores inactives, AWS Lambda i altres serveis serverless podrien ser més eficaços per a un sistema de petites dimensions com aquest, on l'objectiu principal és l'estalvi de recursos quan no hi ha visualitzacions en curs. Lambda permet executar funcions només quan són necessàries, eliminant la necessitat de gestionar nodes durant els períodes d'inactivitat.
- **Optimització de costos**: Considerar l'ús de serveis de serverless per reemplaçar la infraestructura de Kubernetes en alguns casos pot ser una opció més rendible, ja que no es paga per recursos inactius. Lambda i altres serveis serverless poden ser més eficients per processos efímers com la creació de subentorns de reproducció, estalviant així costos associats amb nodes i altres recursos d'infraestructura.

7. Conclusions generals i línies futures

7.1 Conclusions del projecte

Després de desenvolupar i desplegar la plataforma SaaS basada en Kubernetes per gestionar múltiples entorns replicables per client, amb els tres casos d'ús descrits, podem concloure que els objectius inicials del projecte s'han complert satisfactòriament, i la plataforma ha demostrat ser eficaz, escalable i modular.

La arquitectura Kubernetes ha permès crear una plataforma modular i escalable, amb l'ús de namespaces per aïllar els entorns de cada client i garantir la independència dels recursos i dades. Aquest enfocament ha facilitat l'expansió de la plataforma, tant en el cas de la gestió d'hospitals, com en la startup de contingut multimèdia i els exàmens d'estudiants. La possibilitat de replicar entorns amb configuracions específiques per a cada client ha estat un dels punts forts de la solució.

La implementació de l'autoescalatge mitjançant Kubernetes (mitjançant Horizontal Pod Autoscaler i Cluster Autoscaler) ha permès optimitzar els recursos utilitzats en funció de la demanda de cada moment. La plataforma pot escalar automàticament tant a nivell de nodes com de pods, assegurant-se que els recursos siguin adequats durant els períodes de càrrega elevada (com durant els exàmens o l'estrena de contingut) i que es redueixin en moments de baixa activitat. Això ha ajudat a minimitzar els costos operatius, mantenint la plataforma cost-efectiva.

L'ús d'una infraestructura automatitzada amb la metodologia GitOps ha permès sincronitzar els canvis entre els repositoris Git i l'estat actual del clúster de manera automàtica. Els processos de desplegament continu i actualitzacions automàtiques han millorat l'eficiència operativa, reduint el temps i l'esforç manual requerit per gestionar les actualitzacions i canvis en els serveis. Aquesta integració de CI/CD ha millorat la disponibilitat i fiabilitat de la plataforma.

La monitorització de recursos amb Prometheus i Grafana ha permès obtenir una visió detallada i en temps real de la salut dels serveis desplegats. Els dashboards personalitzats a Grafana han ajudat a visualitzar l'estat dels serveis i identificar possibles colls d'ampolla o anomalies de manera ràpida. El sistema d'alertes automàtiques ha estat fonamental per detectar problemes de rendiment (com saturació de CPU o memòria) i permetre una resposta proactiva per evitar interrupcions.

7.2 Possibles millores i continuïtat futura

Tot i que la plataforma ha complert amb els objectius inicials, hi ha algunes millores potencials i recomanacions per a la continuïtat futura del projecte:

1. **Optimització dels Subentorns de Visualització (Startup de Contingut Multimèdia)**:

   En el cas de l'startup de contingut multimèdia, tot i que Kubernetes ha funcionat bé per l'autoescalatge de subentorns dinàmics, es podria optimitzar encara

més utilitzant tecnologies serverless com AWS Lambda. Aquestes tecnologies poden ser més lleugeres per a la gestió d'entorns efímers com el de visualització de contingut, ja que no requerixen la gestió de nodes dedicats, estalviant recursos i costos.

2. **Millora de la Gestió de Recursos durant les Baixes Activitats**:

   En casos com el de l'entorn de testing o quan no hi ha càrrega, la gestió de recursos pot ser encara més eficient mitjançant tecnologies serverless. L'ús de AWS Lambda o Google Cloud Functions permetria una gestió automàtica i extremadament eficaç dels recursos, evitant costos innecessaris quan no hi ha exàmens o visualitzacions en curs.

3. **Ampliació del Monitoratge i Anàlisi Predictiva**:

   Tot i que Prometheus i Grafana proporcionen un monitoratge en temps real, la implementació d'anàlisi predictiva mitjançant Intel·ligència Artificial podria ajudar a identificar tendències de càrrega i anticipar possibles sobrecàrregues abans que es produeixin. Això milloraria la proactivitat del sistema en la gestió de recursos i en la planificació de l'escalabilitat.

4. **Integració amb Nous Sistemes**:

   En el futur, la plataforma es podria integrar amb altres serveis externs per enriquir l'experiència dels usuaris o per millorar la gestió de dades. Per exemple, integrar sistemes de gestió d'usuaris o gestió de contingut multimèdia per crear una plataforma més robusta i escalable.

5. **Desplegament Multinúvol i Resiliència**:

   Per garantir alta disponibilitat i resiliència, una millora futura seria la implementació d'una estratègia multinúvol per allotjar els serveis en diversos proveïdors de cloud, com AWS, Azure i GCP. Això permetria assegurar la continuïtat del servei en cas d'incidències amb algun proveïdor, augmentant la fiabilitat de la plataforma.

Referències

1. Kubernetes. (n.d.). *Kubernetes* [Web]. Recuperat de https://kubernetes.io/es/
2. Argo CD. (n.d.). *Argo CD Documentation* [Web]. Recuperat de https://argo-cd.readthedocs.io/en/stable/
3. Aguilera, I. (2021, 21 de juliol). *Is Kubernetes a good choice for multitenant SaaS?* Reddit. Recuperat de https://www.reddit.com/r/kubernetes/comments/on0557/is_kubernetes_a_good_choice_for_multitenant_saas/?tl=es-es
4. Aguilera, I. (2020, 30 d'agost). *Monitorear clúster de Kubernetes con Prometheus, Loki y Grafana* [Post en blog]. Medium. Recuperat de https://medium.com/@ismaelaguilera_/monitorear-cluster-de-kubernetes-con-prometheus-loki-y-grafana-d6ffb620d265