Estudi de la viabilitat d'usar Apache Kafka com a base de dades distribuïda

---

Treball Final de Màster

Autor: Claudi Cervelló Nogués
Director: Joan Costa Sintes
Ponent: Marc Alier Forment (Dep. ESSI)
Titulació: Màster en Enginyeria Informàtica
Especialització: Computer Networks and Distributed Systems
Data: 27 de juny del 2018
Centre: Facultat d'Informàtica de Barcelona(FIB)
Universitat: Universitat Politècnica de Barcelona (UPC) - Barcelona Tech

Vull agrair a tots els qui, en algun moment o altre, han estat implicats en aquest projecte, el seu esforç, la seva paciència i la seva dedicació.

En primer lloc al director i a la ponent pels consells i orientacions al llarg del projecte.

En segon lloc als membres de l'equip NATOTRITON per l'ajuda i la bona feina.

I per últim, i molt especialment, al meu entorn més proper pel suport incondicional i la confiança.

A tots, moltes gràcies!

Abstract – Català

En aquest Treball de Final de Màster es presenta el disseny d'un sistema distribuït d'una complexitat elevada, on durant l'anàlisi de requisits es fa evident que un disseny tradicional no és una solució viable.

La necessitat de modificar la proposta de disseny tradicional implica la creació d'un nou disseny basat en la idea d'usar un bus de missatges, Apache Kafka, com a base de dades distribuïda, d'aquesta forma garantint: la reducció dels colls d'ampolla, la minimització de la latència de les dades i la simplificació de la replicació de la informació.

Seguint una metodologia de desenvolupament àgil s'estudia la viabilitat del nou disseny i s'implementa en dos entorns no productius. Les lliçons apreses i els tests realitzats són positius aportant la confiança necessària, com per a continuar amb la implementació del nou disseny en un entorn productiu.

Abstract - Castellano

En este Trabajo de Final de Máster se presenta el diseño de un sistema distribuido de complejidad elevada, durante el análisis de requisitos se hizo evidente que un diseño tradicional no es una solución viable.

La necesidad de modificar la propuesta de diseño tradicional, implica la creación de un nuevo diseño basado en la idea de usar un bus de mensajes, Apache Kafka, como base de datos distribuida, de esta forma garantizando: la reducción de los cuellos de botella, la minimización de la latencia de datos y la simplificación de la replicación de la información.

Siguiendo una metodología de desarrollo ágil se estudia la viabilidad del nuevo diseño y se implementa en dos entornos no productivos. Las lecciones aprendidas i los test realizados son positivos aportando la confianza necesaria, para continuar con la implementación del nuevo diseño en un entorno productivo.

Abstract - English

In this Final Master Thesis presents a distributed system design quite complicated, while analyzing the requirements it was clear that a traditional system design was not a feasible solution.

Having to modify the traditional design meant creating a new design based on an idea to use a message bus, Apache Kafka, as a distributed data base, ensuring: the minimization of bottle necks, the minimization of latency in data and the simplification in data replication.

Using an iterative agile methodology for development it is assessed the new design's feasibility and it was implemented in two no productive environments. All the lessons learned and positive testing brought enough trust to continue implementation of the new design in a productive environment.

Índex de continguts

Índex d'il·lustracions

1. Introducció

El document que teniu a les mans és un Treball Final de Màster (TFM) modalitat empresa del Màster en Enginyeria Informàtica, impartit a la Facultat d'Informàtica de la Universitat Politècnica de Catalunya.

El TFM descriu el desenvolupament del projecte TRITON, iniciat al gener del 2018, per part de l'empresa Everis[1]. Durant el disseny d'aquest projecte, es va posar de manifest la necessitat de modificar la proposta de disseny tradicional, plantejant un disseny on Apache Kafka s'usés com a base de dades distribuïda.

L'objectiu principal d'aquest treball és exposar l'estudi de la viabilitat d'usar Apache Kafka com a base de dades distribuïda. Els objectius secundaris són explicar la situació que va motivar la proposta, el desenvolupament i la implementació del nou disseny.

La lectura d'aquest document s'ha de dur a terme tenint present que el projecte TRITON es troba en una fase molt inicial del seu desenvolupament. L'abast del TFM va ser descrit en menys d'un mes, donat que el projecte TRITON es va començar a principis de gener de 2018 i el TFM va ser inscrit al registre de la FIB el dia 30 de gener de 2018. La redacció del TFM s'ha realitzat simultàniament al projecte implicant una incertesa que ha anat disminuint a mesura que el projecte prenia forma.

Aquest document presenta informació, decisions i passos que s'han pres des del gener de 2018 fins al maig de 2018, data de presentació d'aquest TFM. El document s'estructura en el context del document, el projecte TRITON i els requisits més importants. Seguidament, l'anàlisi i raonament que hi ha darrera l'idea d'usar el bus de missatges com a base de dades, formalitzant el disseny en base a aquesta. Després es tracta la implementació, la part més tècnica, identificant com s'ha materialitzat cada component que es descriu al disseny. Finalment, la metodologia i la planificació, seguides de la meva implicació al projecte. Per últim les conclusions extretes i el treball futur.

2. Context

En aquesta secció s'explica el context en el qual aquest TFM s'ha desenvolupat. Durant els dos últims anys, he format part de l'equip d'arquitectura del software d'Everis com a enginyer del software a mitja jornada, al mateix temps que realitzava aquest màster.

A finals del 2017, va sorgir la possibilitat de canviar de projecte i passar a formar part d'un projecte europeu anomenat TRITON. Al conèixer el nou projecte vaig efectuar el canvi donat que era una bona oportunitat tant a nivell professional com estudiantil. Amb la vista posada al TFM, vaig considerar que l'envergadura del projecte i els reptes tecnològics el feien molt atractiu.

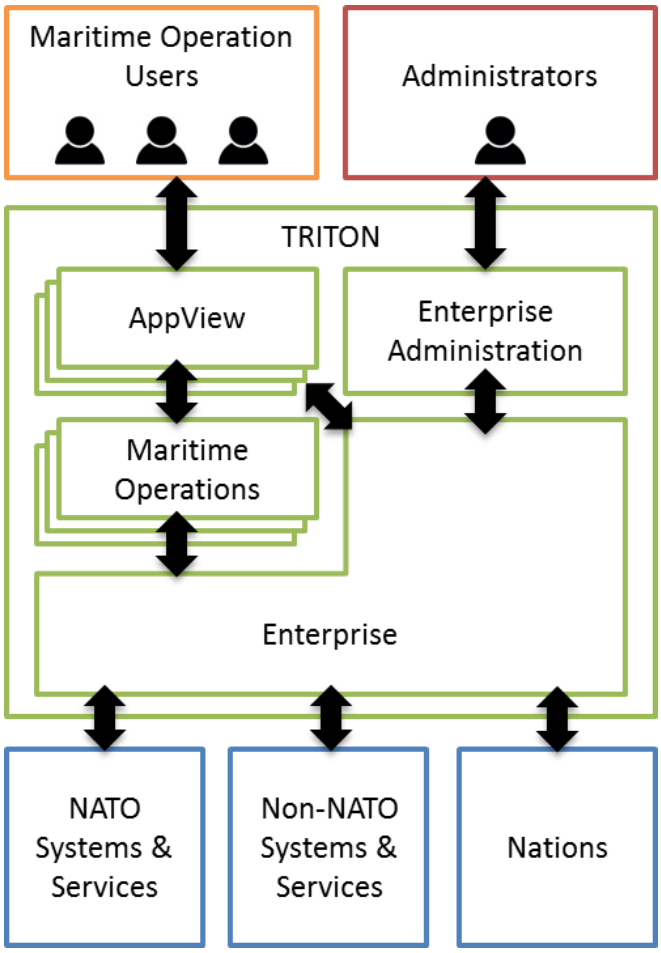
A principis del 2018 començava formalment el projecte TRITON on vaig assumir la responsabilitat d'una de les parts i d'on vaig extreure el tema d'aquest TFM.

3. El projecte TRITON

El projecte TRITON consisteix en un component lògic el qual s'integra dins els entorns de l'Organització del Tractat de l'Atlàntic Nord (abreviat a OTAN o en anglès NATO). Un cop estigui en funcionament, TRITON proveirà funcions de command and control (abreviat a C2) per operacions marítimes. La finalitat de C2 és el seguiment, la coordinació i el comandament d'embarcacions des de terra o des d'una altra embarcació.

TRITON és un projecte desenvolupat per múltiples empreses on Everis s'encarrega de definir i implementar l'arquitectura sobre la qual TRITON serà operatiu. En unes altres paraules, Everis té l'objectiu de crear els fonaments sobre els quals es basarà TRITON. A més a més, Everis té l'encàrrec d'elaborar una arquitectura que permetrà als altres actors, que participen en aquest projecte, construir sobre ella de forma senzilla i el més transparent possible.

La descripció proporcionada pel client sobre què és TRITON queda resumida en el següent diagrama. Aquest permet fer-se una idea sobre què ha de ser TRITON i permet visualitzar els usuaris principals, així com els altres sistemes amb els quals TRITON ha de ser capaç de comunicar-se.



Il·lustració 1: Diagrama d'alt nivell de TRITON

El requadre taronja (usuaris) i el requadre vermell (administradors) representen els usuaris habilitats per interaccionar directament amb TRITON. Els requadres blaus representen tres sistemes externs amb els quals cal comunicar-se: sistemes existents propietat de NATO, sistemes existents externs a NATO i sistemes existents propis de membres pertanyents a NATO. Per últim el requadre verd més extern, per Everis el requadre més important, és l'arquitectura TRITON. Des del punt de vista d'arquitectura podem interpretar els components de la següent forma:

-AppView i Maritime Operations: serveis funcionals.

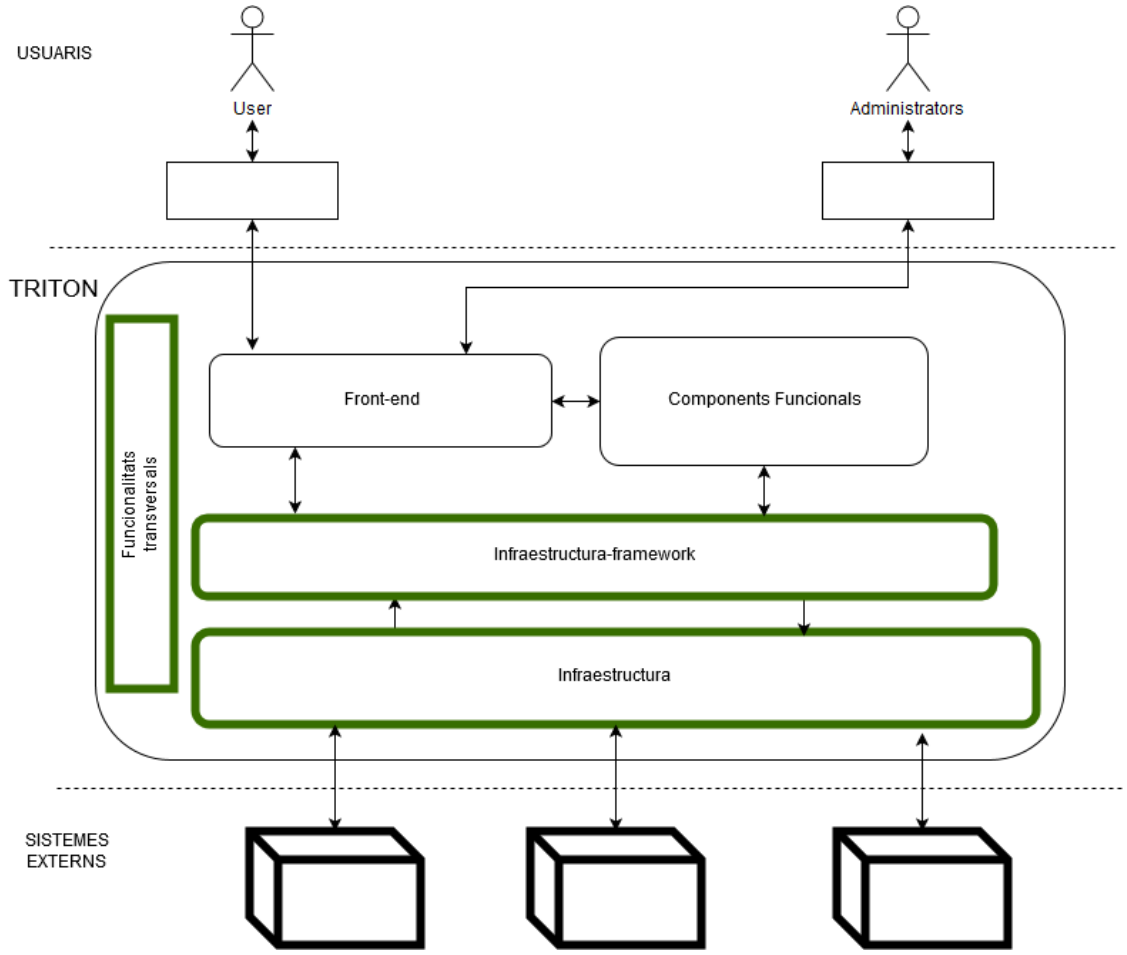
-Enterprise Administration: administració del sistema.

-Enterprese: bus de missatges i base de dades.

3.1. L'arquitectura

L'arquitectura de TRITON, part a desenvolupar per Everis, es divideix en la infraestructura, el infraestructura-framework i les funcionalitats transversals.

Tenint en compte aquest informació, es pot redibuixar el diagrama d'alt nivell de TRITON presentat anteriorment. A continuació, es pot observar quina hauria de ser la taxonomia del sistema.



Il·lustració 2:Taxonomia del sistema TRITON

Començant de dalt cap a baix del diagrama, s'observa la zona d'USUARIS on es veuen els possibles usuaris del sistema. Seguidament, es troba TRITON i, per últim, els sistemes externs amb els quals TRITON interactua.

Observant amb més detall la secció de TRITON:

- Front-end i Components Funcionals: capa de presentació i lògica del sistema.
- Infraestructura-framework: llibreria d'utilitats pel Front-end i els Components Funcionals. Permet comunicar i accedir a la informació.
- Funcionalitats transversals: serveis presents per a tots els components de TRITON (per exemple: mètriques).
- Infraestructura: plataforma on la resta de punts anteriors s'executen.

Els requadres representats en verd corresponen a l'arquitectura i el seu desenvolupament és responsabilitat d'Everis. La resta de components representats en el diagrama utilitzen l'arquitectura per assolir l'objectiu del sistema que és proveir funcions de command and control (abreviat a C2) per operacions marítimes.

4. Requisits

Els requisits presentats pel client demanaven una arquitectura amb capacitat de sostenir un sistema C2 d'embarcacions. Després de la lectura en profunditat d'aquests requisits, la complexitat i l'envergadura del sistema va augmentar i va obligar a replantejar alguns aspectes del disseny, com veurem en apartats posteriors.

Tots els requisits de TRITON s'adjunten al document annex "07-IFB-CO-13859-TRITON-Book II-Part IV-SOW-Annex A-SRS_1.0". La lectura d'aquest document és densa i no és necessària.

Els requisits a complir per part d'Everis es presenten simplificats en quatre mapes mentals, que permeten agrupar els requisits en ordre d'assoliment. La seva lectura és més recomanable que la del document annex anteriorment citat, tot i així, les seccions 4.1 i 4.2 resumeixen el seu contingut.

Tot seguit es presenten les referències als fitxers annex dels quatre mapes mentals: "Requisits-Everis-MapaMental-1", "Requisits-Everis-MapaMental-2", "Requisits-Everis-MapaMental-3" i "Requisits-Everis-MapaMental-4".

La secció 4.1 presenta els requisits de la infraestructura i la secció 4.2 presenta els requisits del infraestructura-framework. Cada apartat consta d'una taula que presenta: la subcategoria a la que pertany el requisit, l'identificador del requisit i la descripció del requisit (en anglès). Aquests dos apartats transmeten al lector informació útil sobre els requisits que TRITON ha de complir i són rellevants de cara a la comprensió posterior del TFM.

4.1. Requisits de la infraestructura

Els requisits de la infraestructura corresponen a requisits de rendiment del hardware, rendiment del sistema operatiu, mecanismes i plataformes de virtualització, tecnologies de tercers, escalabilitat, disponibilitat i TRITON Deployable Kit (TDK).

Els requisits d'escalabilitat, disponibilitat i TDK són especialment importants. L'escalabilitat i la disponibilitat són requisits relativament comuns en infraestructures d'aquesta taxonomia i sempre cal tenir-los en ment durant el disseny de la solució. Ara bé, els requisits TDK no són tan comuns perquè requereixen la capacitat de desplegar la infraestructura en qualsevol moment en una embarcació i, per tant, aporten una complexitat superior a la solució.

Subcategoria	Identificador	Descripció
HARDWARE	[T1-R1745]	The TRITON infrastructure requirements for each instance shall be identified in terms of Service Parameters as defined in the Description.
HARDWARE	[T1-R1746]	The Virtual Machine Performance Parameters shall be scaled not to exceed fifty percent (50%) load on average in twenty-four (24) hours.

HARDWARE	[T1-R1747]	TRITON shall use Open Virtualisation Format, Option B. Mission Participant can create single, pre-packaged appliances and service providers can export and import virtual machines that can run across different virtualisation platforms.
HARDWARE	[T1-R1748]	TRITON shall be able to execute in the virtualisation hypervisor environments supported by NATO, namely "MS Hyper-V Server 12 or later" and "VMWare 5.5 or later".
HARDWARE	[T1-R1749]	The Server deployment package (virtual appliance or installation package) shall be tested against the used hypervisors, configured in accordance with NATO Security Settings “VMWare ESXi 5.5 or later” and “Microsoft Hyper-V 2012 or later".
OPERATING SYSTEM	[T1-R1750]	TRITON shall use a COTS Operating System on a Virtualised Environment. If the proposed solution does not use MS Windows, it shall be specified, documented, justified and the necessary licenses shall be provided.
OPERATING SYSTEM	[T1-R1751]	TRITON shall comply with the latest versions of the operating system available and supported by the Bi-SC AIS Servers and Workstations. This should include all versions of the operating systems planned to become available for the Bi-SC AIS prior to the TRITON System Integration Tests.
OPERATING SYSTEM	[T1-R1752]	The Operating System selected for TRITON shall have an operating lifetime of at least five (5) years. Any deviations shall be described and justified in the Obsolescence Management Plan (OMP). Standard documentation shall be provided.
VIRTUALITZATION	[T1-R1602]	TRITON Operational Software shall be adaptable to newer versions of the virtualised environment.
VIRTUALITZATION	[T1-R1753]	TRITON shall be able to utilise Virtualised Server provided by the Virtualised Environment.
VIRTUALITZATION	[T1-R1754]	TRITON Virtualised Server shall be designed considering performance, scalability and load balancing factors.
VIRTUALITZATION	[T1-R1755]	TRITON Virtualised Server shall use the Server Components given in the Description. Any necessary third-party software and licenses shall be provided for each instance of TRITON.

VIRTUALITZATION	[T1-R1756]	TRITON Virtualised Server shall be compatible with the native file system used on NATO Servers. Any deviation shall be documented in detail with justification and be subject to the Purchaser's evaluation and approval.
VIRTUALITZATION	[T1-R1757]	TRITON shall be able to utilise Virtualised Data Storage provided by the Virtualised Environment.
VIRTUALITZATION	[T1-R1758]	The TRITON Operational Software shall not have any direct dependency to the physical parameters of the storage environment (such as disk type, connection type, Storage Area Network and protocol) provided by the Virtualised Environment.
VIRTUALITZATION	[T1-R1759]	All UNC, File Path, Drive Letter or similar storage location settings used in TRITON Virtualised Data Storage shall be parametric, configurable, and possible to automate for unattended installation, backup, recovery. No hard-coded location setting shall be used.
THIRD-PARTY SOFTWARE	[T1-R1760]	TRITON may use third-party COTS Software or Open Source Software to support its Infrastructure Software. Samples are given in the Description.
THIRD-PARTY SOFTWARE	[T1-R1761]	The third-party COTS Software shall be the latest commercial version with the latest updates, unless agreed by the Purchaser.
THIRD-PARTY SOFTWARE	[T1-R1762]	The licensing for all third-party COTS Software components (including versions/editions, client access licenses, etc.) shall allow the component to take full advantage of the number of allocated processors and memory of the Virtualised Server to support the maximum number of users.
THIRD-PARTY SOFTWARE	[T1-R1763]	All third-party COTS Software shall have standard product documentation, a product support and validity time of at least five (5) years. Any deviations shall be described in the Obsolescence Management Plan.
THIRD-PARTY SOFTWARE	[T1-R1764]	TRITON may use COTS Software in its architecture in place of dedicated solutions when the functionalities of a COTS product matches the requirements for a service with no or minimal adaptation.
THIRD-PARTY SOFTWARE	[T1-R1765]	When COTS solutions are used, adaptations shall be delivered as additional services that complement the COTS Software native functionalities.

SCALABILITY	[T1-R1413]	TRITON shall be able to support at least twenty percent (20%) increase in users with no performance degradation during surge situations.
SCALABILITY	[T1-R1415]	TRITON shall have growth potential beyond the figures given to indicate data storage capacity, database or list sizes, and shall not deem any limitation on design.
AVAILABILITY	[T1-R1527]	TRITON shall ensure system availability to users so that they do not experience interruption of services as a result of intermittent connection, except as it impacts their direct access to Web Application Server for an in-progress action. Intermittent connection is defined as loss of connectivity that is less than thirty (30) seconds.
AVAILABILITY	[T1-R1528]	TRITON shall ensure system availability to users so that they do not experience interruption of services as a result of Limited Bandwidth. Limited Bandwidth is defined as a bandwidth of less than sixty-four (64) kbps for afloat sites and five-hundred-and-twelve (512) kbps for static sites.
AVAILABILITY	[T1-R1529]	TRITON shall ensure system availability to users so that they do not experience interruption of services as a result of high latency. High latency is defined as latency exceeding one-thousand-and-one-hundred (1100) milliseconds.
AVAILABILITY	[T1-R1549]	TRITON operation shall not be interrupted during incorporation of in-service equipment updates or maintenance software updates.
AVAILABILITY	[T1-R1550]	TRITON shall support fail-over in its architecture. Clients shall be able to survive failure of the Application Server without crashing or affecting the stability of the overall system.
TDK	[T1-R013]	TRITON Deployable Kit shall utilise its own infrastructure to support the operational software in Standalone Mode.
TDK	[T1-R1376]	TDK shall host the TRITON-NS Operational Software configured for ACP on the NS-Unit.
TDK	[T1-R1377]	TDK shall host the TRITON-NU Operational Software configured for ACP on the NU-Unit.
TDK	[T1-R1378]	TDK shall have its own installation tools and configuration procedures to install and operate the TRITON Operational Software.

4.2. Requisits del infraestructura-framework

Els requisits del infraestructura-framework corresponen a requisits d'emmagatzematge, sistema PUB/SUB, redundància, recuperació del sistema, replicació de dades i sincronització de dades

Els requisits del sistema PUB/SUB fan referència al funcionament del bus de missatges, mentre que els requisits de recuperació del sistema descriuen com intentar recuperar o passar a una segona infraestructura en cas de caiguda del sistema.

El requisit de sincronització de dades és comú però, a l'incorporar els requisits TDK, la sincronització no és tan senzilla. El mecanisme de sincronització descrit als requisits consisteix a sincronitzar tota la informació independentment d'on resideixi la infraestructura, sigui a terra o a una embarcació.

Subcategoria	Identificador	Descripció
STORAGE	[T1-R026]	TRITON shall allow the authorised user to manage (create, backup, restore, delete, import, export) the Operational-Specific Store. The user shall be able to import whole or part of previously exported Maritime Operational Object Databases into the internal databases of a Maritime Operation.
STORAGE	[T1-R026]	TRITON shall allow the authorised user to manage (create, backup, restore, delete, import, export) the Operational-Specific Store. The user shall be able to import whole or part of previously exported Maritime Operational Object Databases into the internal databases of a Maritime Operation.
PUB/SUB SYSTEM	[T1-R773]	TRITON Middleware should provide for any TRITON service to subscribe to hierarchical topics and receive publications over the Middleware.
PUB/SUB SYSTEM	[T1-R774]	TRITON Middleware should provide for consumer services to subscribe to Maritime Information Entities using the topic syntax.

PUB/SUB SYSTEM	[T1-R775]	TRITON Middleware should use event-driven mechanisms compliant with OASIS WS-Notifications protocols to consume event driven, time sensitive and critical Web Services of other systems.
PUB/SUB SYSTEM	[T1-R776]	TRITON Middleware should allow consumers to initiate and manage subscriptions.
PUB/SUB SYSTEM	[T1-R777]	TRITON Middleware should provide for a publication manager to manage all publications from its services.
PUB/SUB SYSTEM	[T1-R778]	TRITON Middleware should allow subscribers to manage their subscription.
PUB/SUB SYSTEM	[T1-R779]	TRITON Middleware should publish each element of the Maritime Information Entities by creating a hierarchical topics structure.
PUB/SUB SYSTEM	[T1-R780]	TRITON Middleware should publish each element of the Maritime Information Entities by initiating a message delivery corresponding to the topic.
PUB/SUB SYSTEM	[T1-R781]	TRITON Middleware should publish each element of the Maritime Information Entities with an appropriate filtering syntax to allow consumers to subscribe to a subset of those Entities.
PUB/SUB SYSTEM	[T1-R782]	TRITON Middleware should allow consumers to subscribe to Maritime Information Entities using the topic and filtering syntax.
PUB/SUB SYSTEM	[T1-R783]	TRITON Middleware should provide for subscriptions to be either infinite (i.e. a subscription remains in force until it is cancelled) or subscriptions with predefined termination time, which automatically expire (i.e. the consumer is only a subscriber for a certain amount of time).
PUB/SUB SYSTEM	[T1-R784]	TRITON Middleware should provide synchronisation capability to consumers with Core Data Store for a given time period using a synchronisation interface.

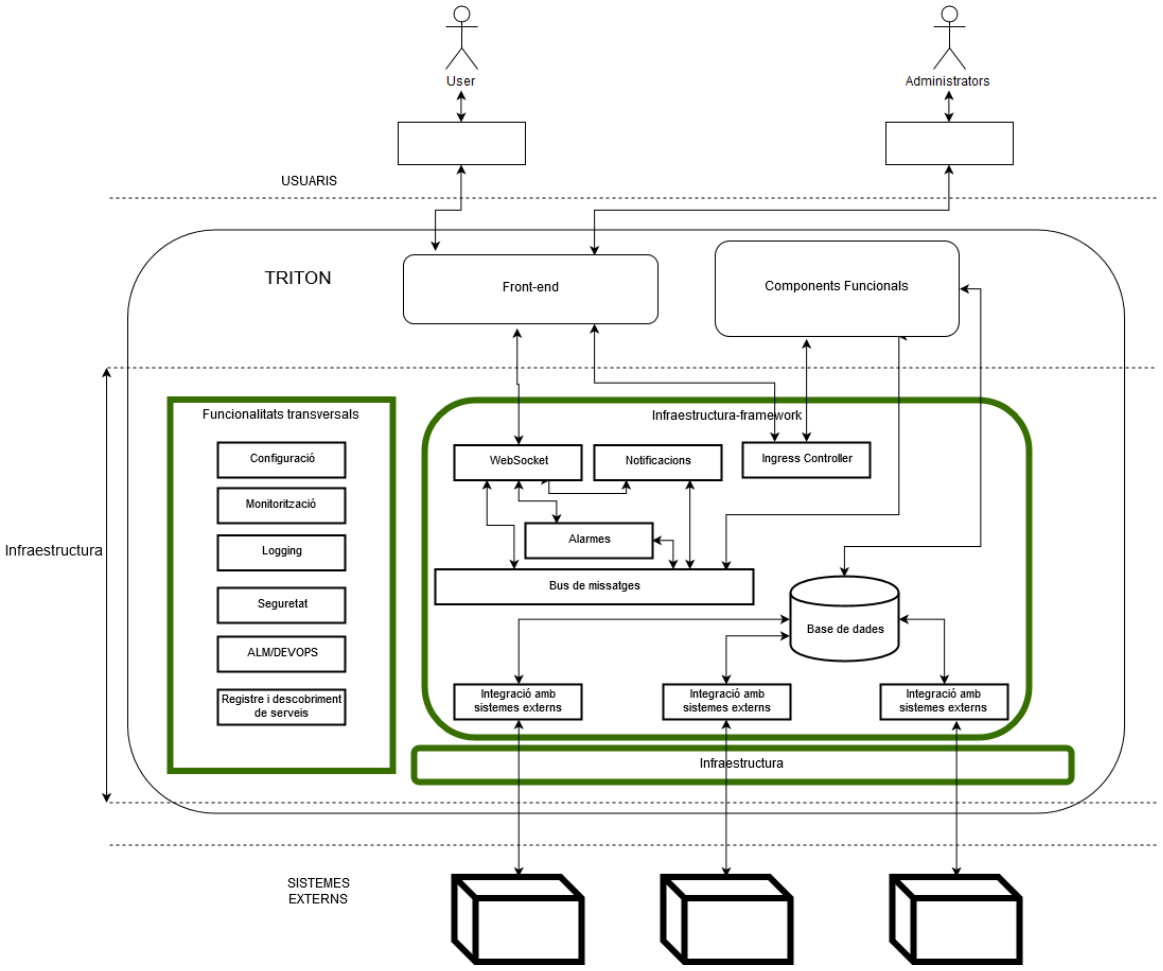
REDUNDANCY	[T1-R670]	TRITON shall implement a Redundancy Management using master-slave mechanism and redundancy methods as defined in the Description. COTS solutions may be used upon Purchaser approval.
REDUNDANCY	[T1-R671]	TRITON shall allow the authorised user to configure instances of TRITON for Redundancy Management.
REDUNDANCY	[T1-R672]	TRITON shall allow the authorised user to control and monitor the Redundancy Management.
TAKEOVER	[T1-R673]	In case the Active TRITON Instance fails, the Hot Standby Instance shall automatically take over and become operational within sixty (60) seconds.
TAKEOVER	[T1-R674]	In case the Active TRITON Instance fails and the Hot Standby Instance is not available, the Warm Standby Instance shall become operational within fifteen (15) minutes after the manual initiation by the authorised user.
TAKEOVER	[T1-R675]	In case the Active, Hot Standby and Warm Standby TRITON Instances fail, the Cold Standby Instance shall become operational within two (2) hours after the manual initiation.
TAKEOVER	[T1-R1548]	In case of a TRITON Server failure, TRITON shall be capable of switching to a TRITON Backup Server within three (3) minutes.
DATA REPLICATION	[T1-R676]	TRITON shall support Data Replication to ensure complete, accurate, timely, confidential and consistent data coherence between instances. Data Centre Infrastructure shall be utilised to achieve resilience.
DATA REPLICATION	[T1-R677]	TRITON shall allow the authorised user to configure Data Replication rules over selected data (e.g. critical, non-critical).
DATA REPLICATION	[T1-R678]	The maximum allowed latency for a set of selected synchronised data shall not exceed one (1) minute for static instances and three (3) minutes for afloat instances.

DATA REPLICATION	[T1-R679]	TRITON shall be able to replicate new data entry on the Active Instance database on the other Instances' databases based on the rules set by the authorised user.
DATA REPLICATION	[T1-R680]	TRITON shall be able to replicate new data instances that are marked as "Critical" no later than ten (10) seconds plus the average network latency of the infrastructure.
DATA REPLICATION	[T1-R681]	TRITON "will" be able to use Universally Unique Identifier (UUID) [ISO/IEC 9834] for Database Replication.
DATA REPLICATION	[T1-R682]	TRITON shall allow the authorised user to manage (configure, monitor, control) Data Replication Process for all TRITON instances.
DATA SYNCHRONISATION	[T1-R684]	TRITON shall support Data Synchronisation Process defined in the Description to synchronise its internal databases with the selected TRITON Server. The functionality over the data shall be preserved.
DATA SYNCHRONISATION	[T1-R685]	TRITON shall allow the authorised user manage (configure, monitor, control) the Data Synchronisation Process.
DATA SYNCHRONISATION	[T1-R686]	TRITOA54:C77the authorised user when any inconsistency is detected during synchronisation process.

5. Idea: Utilitzar el bus de missatges com a base de dades distribuïda

Després de descriure la infraestructura de TRITON formalment a partir de requisits, en aquesta secció es pretén visualitzar la infraestructura amb més detall. Al llarg d'aquest apartat, s'incorpora la informació dels requisits i es justifica el motiu d'usar Kafka com a base de dades distribuïda.

A continuació es mostra el diagrama sobre la taxonomia del sistema i s'incorporen els components descrits a l'apartat de requisits. La descripció del diagrama correspon a una visió d'alt nivell d'una sola instància de TRITON.



Il·lustració 3: Diagrama d'alt nivell complet

Començant per la part superior del diagrama, s'observa els usuaris del sistema, el Front-end i els Components Funcionals. No es mostren més detalls d'aquestes tres entitats del diagrama perquè no formen part de la infraestructura i, conseqüentment, queden fora de l'abast d'aquest document.

Seguidament, es mostren els components que són responsabilitat d'Everis.

- Funcionalitats transversals: serveis disponibles per a tots els components de TRITON. Les funcionalitats transversals es subdivideixen en configuració, monitorització, *logging*, seguretat, *Application Lifecycle Management* (abreviat a ALM), *Development Operations* (abreviat a DEVOPS) i registre i descobriment de serveis.
- Infraestructura-framework: llibreria d'utilitats pel Front-end i pels Components Funcionals. Permet comunicar i accedir a la informació proporcionant dos canals de comunicació: comunicació basada en WebSocket i comunicació basada en Ingress Controller.
    - WebSocket: canal bidireccional full-dúplex.
    - Ingress Controller: motor de regles que avalua cada petició contra les regles actives acceptant o denegant l'accés.

  Per últim, també es troben utilitats de notificació i alarmes.
- Infraestructura: infraestructura on la resta de punts anteriorment mencionats s'executen.

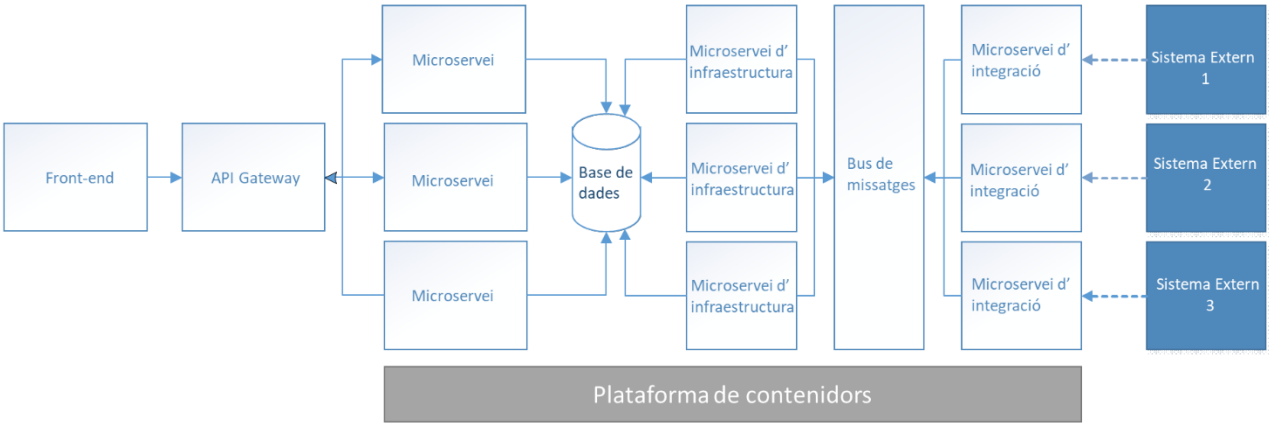
En la part inferior del diagrama, es troben els sistemes externs.

5.1. Proposta de disseny

En aquest apartat es presenta la proposta de disseny de la infraestructura de TRITON. En primer lloc, es presenta un disseny més tradicional, que incorpora un model de relació entre components més comunament utilitzat durant els últims anys. En segon lloc, es presenta una nova proposta que repensa el model de relació entre components i que pretén simplificar la infraestructura.

5.1.1. Model tradicional

La infraestructura proposada utilitza contenidors, per tant, cada component representat al següent diagrama és un contenidor. Els contenidors permeten desplegar els serveis de forma ràpida i escalable, al mateix temps que permeten a altres equips desplegar components funcionals.



Il·lustració 4:Proposta de disseny tradicional

A continuació, s'analitza el diagrama d'esquerra a dreta.

- Front-end: interfície d'usuaris.
- API Gateway: servidor que rep i gestiona les peticions que es fan a cada microservi, sent transparent per a l'usuari.

NOTA: Front-end i API Gateway s'executen fora de la infraestuctura. El Front-end s'executa a la màquina de l'usuari i l'API Gateway és un servei proporcionat per tercers.

- Microserveis: components que aporten lògica a TRITON desenvolupats per altres equips. Per comunicar-se amb la infraestructura i transmetre informació aquests fan ús del infraestructura-framework.

NOTA: Everis proporciona el infraestructura-framework i el mecanisme de contenidors per als microserveis, però no proporciona la lògica funcional.

- Base de dades: component que permet l'emmagatzematge i la consulta d'informació per part dels microserveis.
- Microserveis d'infraestructura: microserveis responsables de transformar els missatges provinents de sistemes externs i emmagatzemar-los a la base de dades.

NOTA: Everis sí que proporciona aquests microserveis.

- Bus de missatges: protocol de comunicació que permet l'enviament de missatges entre microserveis d'infraestructura i microserveis d'integració.
- Microserveis d'integració: microserveis encarregats de comunicar-se amb sistemes externs.

NOTA: Everis sí que proporciona aquests microserveis.

- Plataforma de contenidors: entorn que facilita el desplegament i la gestió de contenidors.
- Sistemes externs: sistemes amb els quals TRITON interactua.

NOTA: Els sistemes externes es gestionen i s'executen en plataformes de tercers.

El model tradicional presenta una sèrie de problemes que posen en risc el correcte funcionament del sistema.

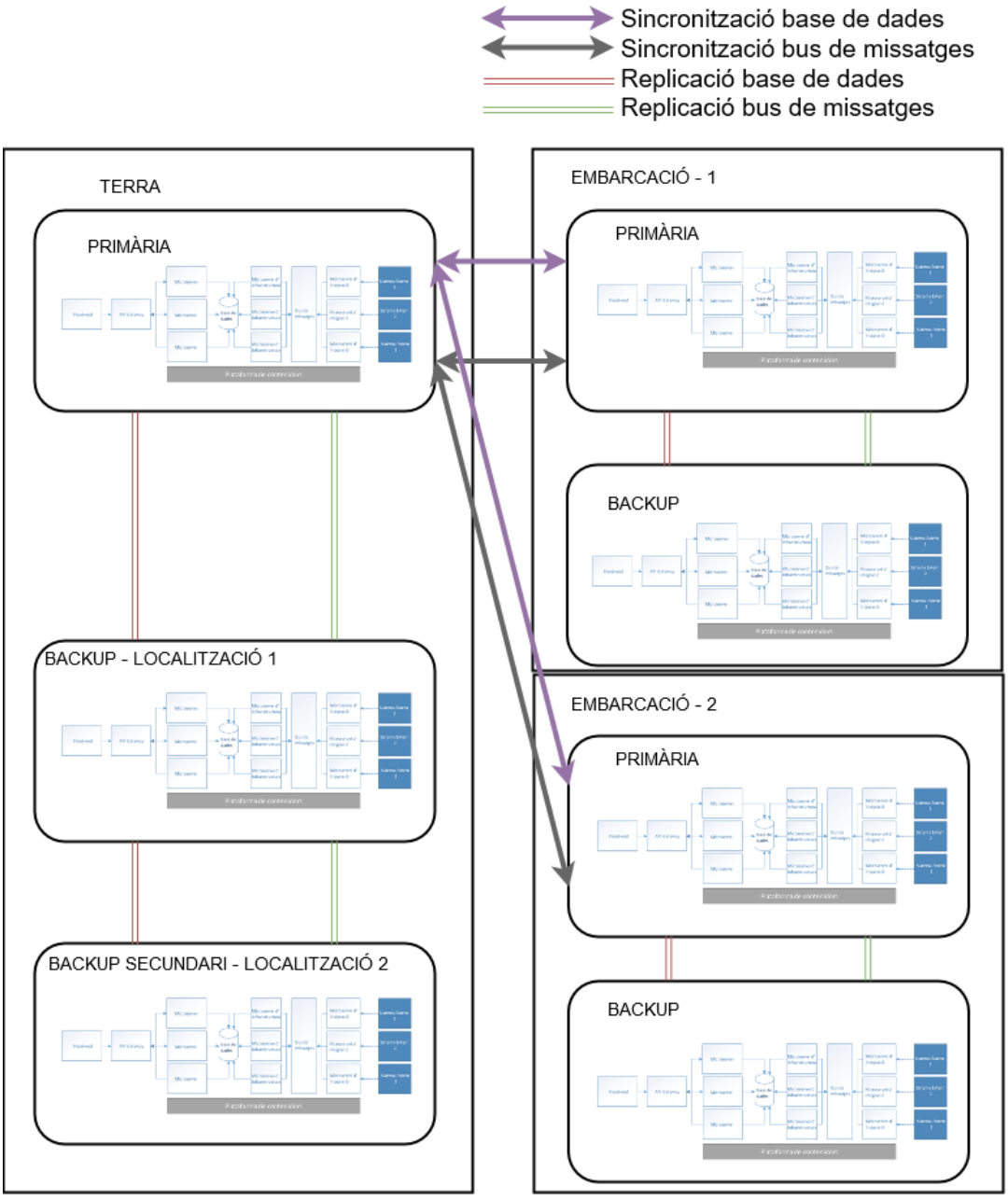
En primer lloc, el model tradicional presenta una alta probabilitat de generar colls d'ampolla quan el sistema escala. La base de dades representa un component crític del sistema, donat que és el punt principal d'accés a la informació i és un punt on es pot produir l'aparició d'un coll d'ampolla.

En segon lloc, aquest model presenta una latència elevada, és a dir, des de que un sistema extern envia la informació a TRITON fins que la informació està disponible es consumeix molt temps. La latència elevada és perjudicial per al sistema implicant una ràpida degradació d'aquest.

En tercer lloc, la redundància i la replicació de la informació és complexa. Aquesta complexitat prové de la necessitat d'haver de garantir la redundància i la replicació en dos components del sistema: la base de dades i el bus de dades. La redundància ha de maximitzar el rendiment i minimitzar la pèrdua d'informació dels dos components a la mateixa instància. La replicació ha d'assegurar que les dades dels dos components han estat copiades en una altra instància.

A més a més dels tres problemes exposats anteriorment, el model tradicional presenta una elevada complexitat a l'hora de sincronitzar i replicar informació en un entorn multi-instància distribuït. El diagrama següent mostra aquesta complexitat. Per simplificar el diagrama es mostren tres instàncies a terra i només dues instàncies a embarcacions:

- Tres instàncies a terra:
    - una instància primària
        - una instància configurada com a backup primari
            - una instància configurada com a backup secundari
- Dos instàncies a embarcacions:
    - una instància primària a l'embarcació 1
        - una instància configurada com a backup primari
    - una instància primària a l'embarcació 2
        - una instància configurada com a backup primari



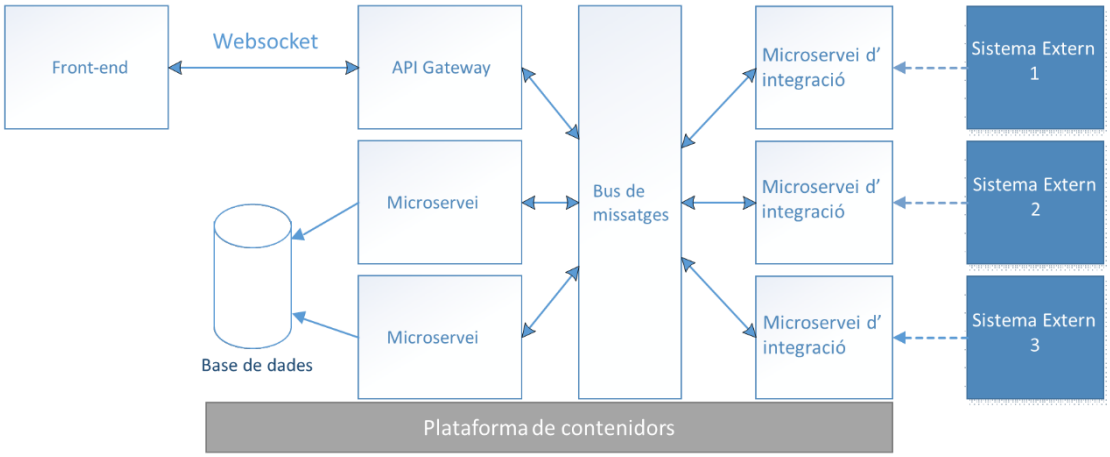
Il·lustració 5: Replicació i sincronització en la proposta tradicional

5.1.2. Nova proposta

La nova proposta té com a objectiu simplificar els potencials problemes i dificultats identificades al model tradicional, així com aprimar el disseny sense penalitzar a TRITON com a sistema.

Per assolir l'objectiu, un bon punt de partida és l'observació de la volumetria del sistema. Grans volums de dades procedeixen de sistemes externs i d'altres instàncies de TRITON, orientar el disseny cap a un paradigma de streaming de dades ajudarà a complir l'objectiu.

La nova proposta manté una infraestructura que utilitza contenidors, sent cada component representat al següent diagrama un contenidor.



Il·lustració 6: Nova proposta de disseny

A continuació, s'analitza el diagrama d'esquerra a dreta.

- Front-end: interfície d'usuaris.

NOTA: Front-end s'executa fora de la infraestructura, concretament a la màquina de l'usuari.

- API Gateway: servidor que rep i gestiona les peticions que es fan a cada microservi, sent transparent per a l'usuari. Al nou model, API Gateway passa a ser un contenidor desplegat per Everis, fet que proporciona més control sobre la infraestructura i la possibilitat de diversificar protocols de comunicacions.
- Microserveis: components que aporten lògica a TRITON desenvolupats per altres equips. Per comunicar-se amb la infraestructura i transmetre informació aquests fan ús del infraestructura-framework.

NOTA: Everis proporciona el infraestructura-framework i el mecanisme de contenidors per als microserveis, però no proporciona la lògica funcional.

- Base de dades: component que permet l'emmagatzematge i la consulta d'informació per part dels microserveis. Al nou model, l'accés a la base de dades queda limitada als microserveis.
- Bus de missatges: protocol de comunicació que permet l'enviament de missatges entre API Gateway, microserveis i microserveis d'integració. Al nou model, absolutament tots els missatges entre components s'envien pel bus de dades.
- Microserveis d'integració: microserveis encarregats de comunicar-se amb sistemes externs.

NOTA: Everis sí que proporciona aquests microserveis.

- Plataforma de contenidors: entorn que facilita el desplegament i gestió de contenidors.
- Sistemes externs: sistemes amb els quals TRITON interactua.

NOTA: Els sistemes externes es gestionen i s'executen en plataformes de tercers.

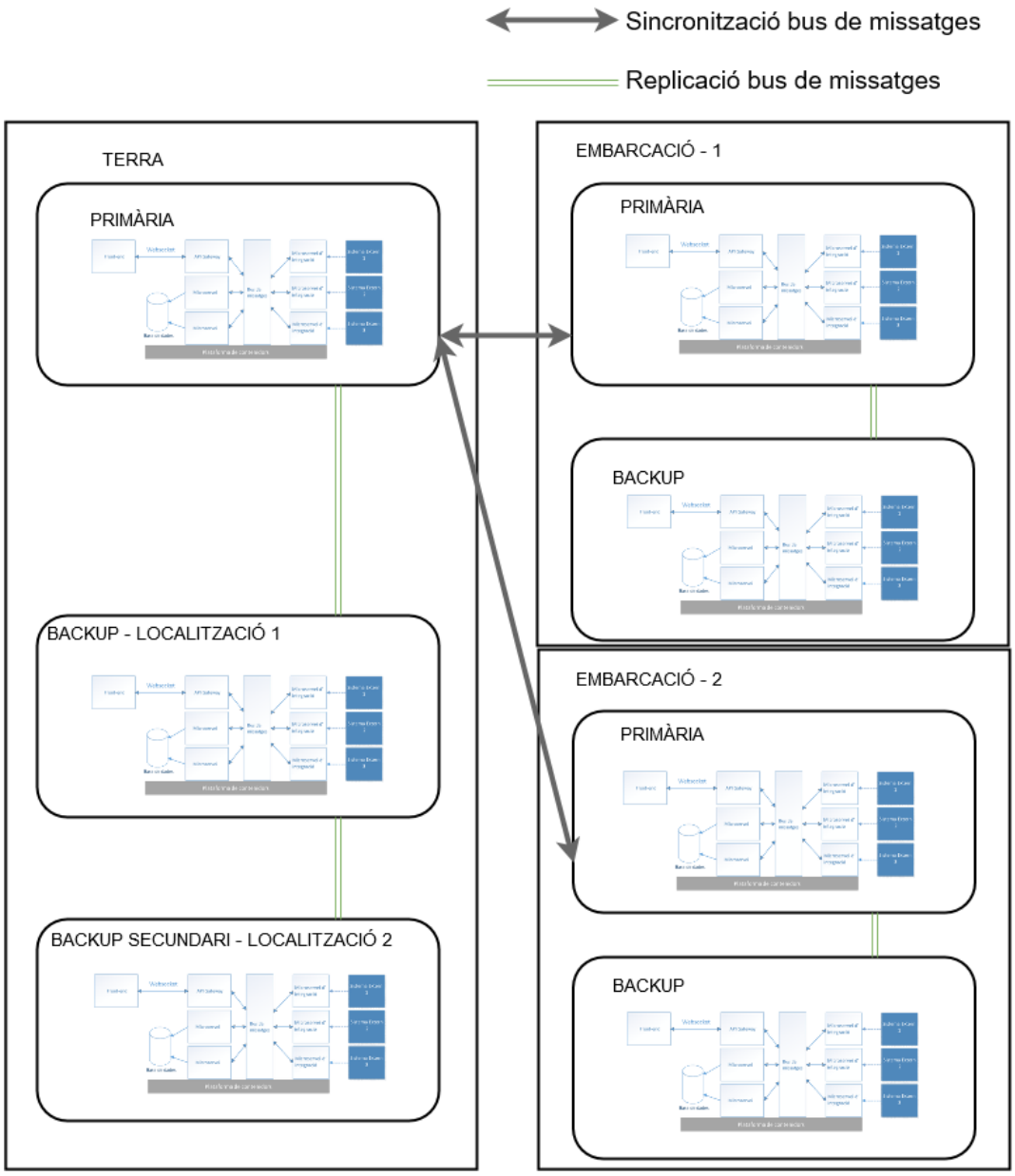
El nou model elimina els colls d'ampolla mitjançant la supressió de la capa de microserveis d'infraestructura. D'aquesta forma, és possible escalar minimitzant el risc present al model tradicional.

La minimització de la latència durant l'accés a dades s'aconsegueix a través de l'eliminació de la capa de microserveis d'infraestructura i l'ús del bus de dades com a canal de comunicació entre tots els components. L'enfoc dels fluxos d'informació cap a un paradigma de streaming millora els temps de disposició de la informació, ajudant a solucionar el problema de la latència present al model tradicional.

El nou disseny redueix la complexitat relacionada amb la redundància i la replicació de dades. La redundància segueix tenint el mateix pes que al disseny tradicional, millorant el rendiment i prevenint la pèrdua de dades. Ara bé, la replicació presenta una millora considerable respecte al model tradicional donat que, en aquesta ocasió, ja no és necessari replicar la base de dades. La nova proposta només exigeix la replicació del bus de missatges. La complexitat de la replicació queda dividida a la meitat gràcies a què la comunicació entre components ha de passar pel bus de missatges. Aquest canvi permet que, en front d'una fallada de la base de dades, sigui possible recuperar la informació perquè els missatges, que prèviament han viatjat pel bus, són la informació del sistema.

A continuació, es mostra el diagrama sobre com seria TRITON amb més d'una instància seguint la nova proposta. El diagrama mostra tres instàncies a terra i només dues instàncies a embarcacions.

- Tres instàncies a terra:
    - una instància primària
        - una instància configurada com a backup primari
            - una instància configurada com a backup secundari
- Dos instàncies a embarcacions:
    - una instància primària a l'embarcació 1
        - una instància configurada com a backup primari
    - una instància primària a l'embarcació 2
        - una instància configurada com a backup primari

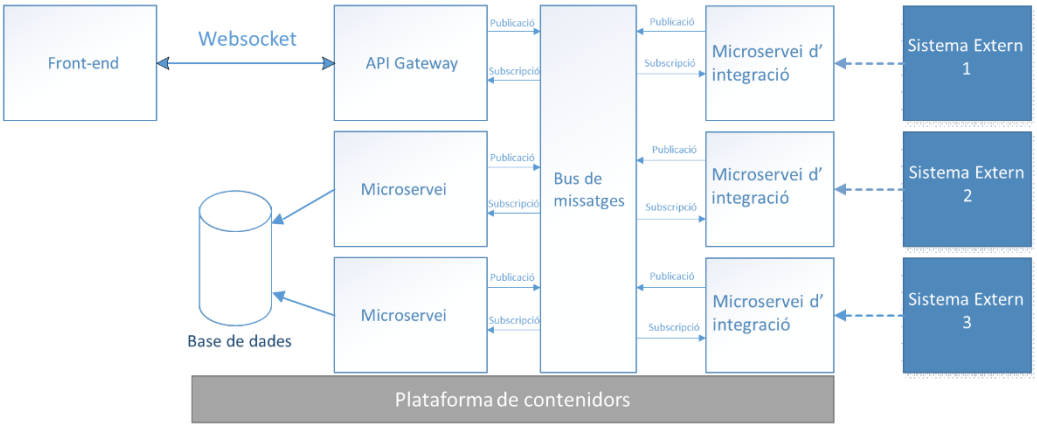


Il·lustració 7: Replicació i sincronització en la nova proposta

La nova proposta permet aprimar la infraestructura, reduir els potencials colls d'ampolla i simplificar la gestió de la replicació i la sincronització. No obstant, la nova proposta introdueix algunes alteracions respecte el model d'enviament de missatges, el model de persistència i el model de replicació.

5.2. Model d'enviament de missatges

A la nova proposta de disseny, el bus de dades és l'espina dorsal del sistema. Qualsevol missatge enviat d'un component a un altre passa pel bus de dades, per tant, tots els components han de tenir la capacitat de publicar i subscriure missatges.



Il·lustració 8: Disseny amb publicació i subscripció de missatges

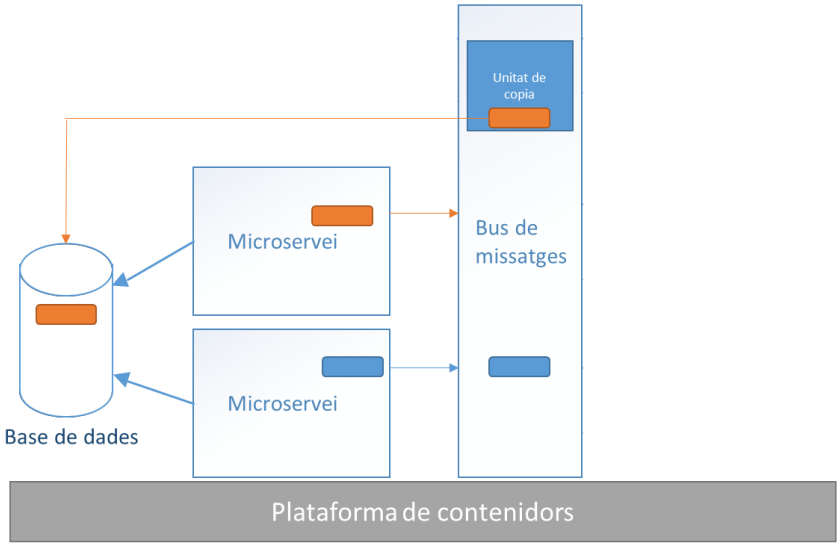
El diagrama mostra com tots els components publiquen i subscriuen missatges. Al tractar-se d'un mecanisme homogeni de publicació i subscripció de missatges, aquest és proveït pel infraestructura-framework.

5.3. Model de persistència

Complint amb el model d'enviament de missatges, la informació del sistema és persistent.

Ara bé, com es pot veure als diagrames, els microserveis tenen accés a una base de dades. Aquesta base de dades és purament funcional, és a dir, proporciona un espai d'emmagatzematge per a què els microserveis pugin realitzar creuaments i operacions sobre informació. El manteniment d'aquesta base de dades a la nova proposta de model aporta accés a certa informació amb un llenguatge formal de domini. El infraestructura-framework dóna l'opció de guardar una còpia del missatge a la base de dades al moment d'enviar el missatge.

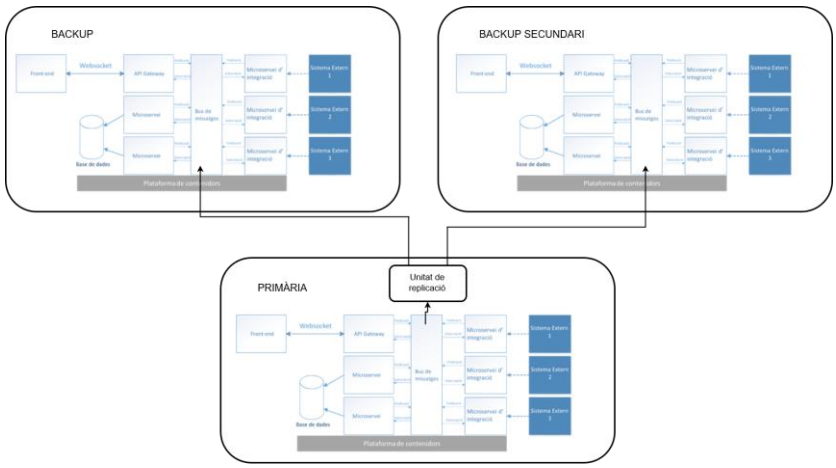
El següent diagrama mostra el recorregut de l'enviament d'un missatge amb còpia a la base de dades.



Il·lustració 9: Enviament d'un missatge amb còpia a la base de dades

5.4. Model de replicació

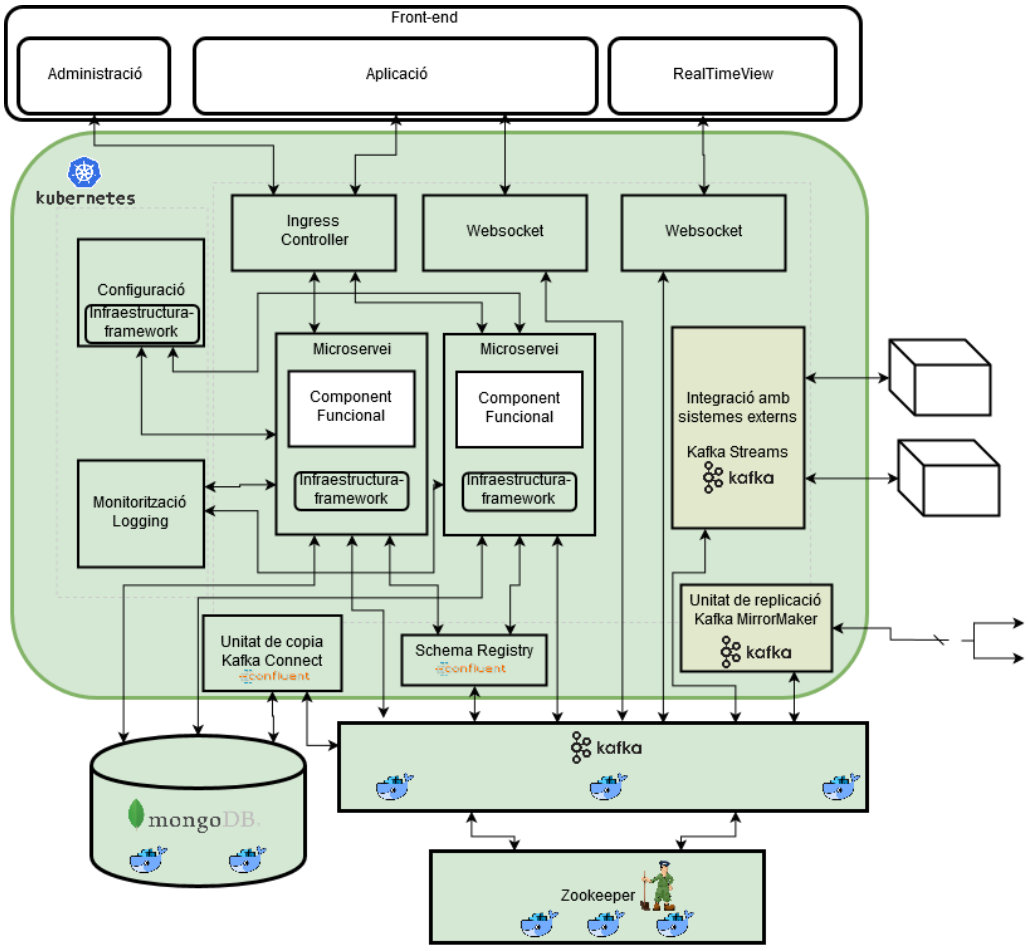
El model de replicació consisteix en l'habilitat de transferir els missatges del bus de dades de la instància principal a instàncies secundaries, assumint que el model d'enviament de missatges es compleix.



Il·lustració 10: Replicació del bus de missatges

6. Implementació

Aquest apartat explica com implementar la idea d'utilitzar el bus de dades com a base de dades distribuïda, consolidant una infraestructura orientada a contenidors per a un sistema de streaming de dades distribuït.



Il·lustració 11: Implementació de la idea

El diagrama mostra la implementació de la idea tenint present la nova proposta de disseny, els requeriments i el coneixement funcional de TRITON. Als següents apartats, s'exposen els components d'infraestructura així com la integració de la idea en el infraestructura-framework.

6.1. Apache Kafka i Apache Zookeeper

Apache Kafka i Apache Zookeeper són dos projectes de codi obert gestionats per l'Apache Sotware Fundation[2]. Apache Kafka[3] és una plataforma distribuïda de propòsit general per a la transferència de grans volums de dades en temps real.

En la implementació, Kafka és el bus de missatges i el sistema d'emmagatzematge. Ara bé, al tractar-se d'una plataforma distribuïda, és necessària una coordinació distribuïda i en alta disponibilitat, sent aquí on intervé Apache Zookeeper[4].

Kafka és el punt de comunicació entre tots els components de la infraestructura i utilitza publicacions i subscripcions, procedents de components de la infraestructura, per accedir a la informació emmagatzemada en tòpics. Cada tòpic està encriptat i protegit amb contrasenya, aportant d'aquesta forma separació lògica entre col·leccions d'informació.

Per assegurar l'alta disponibilitat tant de Zookeeper com de Kafka, ambdues s'executen en clúster i fora de Kubernetes permetent aconseguir el millor rendiment possible a l'hora d'emmagatzemar informació.

NOTA: Per aconseguir persistència a Kubernetes cal usar NFS[5] i, tot sovint, comporta compromisos de rendiment.

Per desplegar Kafka i Zookeper en contenidors, s'utilitza màquines virtuals en clúster prèviament aprovisionades amb Docker[6] Per exemple en un clúster de tres màquines, es procedeix de la següent manera:

1. Desplegament de Zookeeper a els tres màquines usant la imatge oficial de Confluent[7](empresa fundada per l'equip que va construir Apache Kafka):

   ```
   docker run -d --net=host --restart always --name=zookeeper -e ZOOKEEPER_CLIENT_PORT=2181 -e
   ZOOKEEPER_SERVER_ID=1 -e "ZOOKEEPER_SERVERS=kafka-cluster-1:2888:3888;kafka-cluster-
   2:2888:3888;kafka-cluster-3:2888:3888" -v zk-data-cp:/var/lib/zookeeper/data -v zk-datalog-
   cp:/var/lib/zookeeper/log confluentinc/cp-zookeeper:4.0.0
   ```

2. Desplegament de Kafka a els tres màquines usant la imatge oficial de Confluent:

```
docker run -d --net=host --restart always --name=kafka -e "KAFKA_ZOOKEEPER_CONNECT=kafka-cluster-1:2181,kafka-cluster-2:2181,kafka-cluster-3:2181" -e KAFKA_HOST_NAME=0.0.0.0 -e KAFKA_BROKER_ID=1 -e KAFKA_AUTO_CREATE_TOPICS_ENABLE=false -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://<MACHINE_IP>:9092 -e KAFKA_DEFAULT_REPLICATION_FACTOR=3 -e KAFKA_MIN_INSYNC_REPLICAS=2 -v kafka-cp:/var/lib/kafka/data confluentinc/cp-kafka:4.0.0
```

D'aquesta forma, totes les màquines que formen part del clúster de Zookeeper són visibles entre elles i, al mateix temps, són visibles per les màquines que formen part del clúster de Kafka que, a més a més, estan coordinades per Zookeeper.

6.2. MongoDB

MongoDB[8] és una base de dades NoSQL orientada a documents de codi obert. En la implementació, MongoDB és la base de dades per a operacions amb informació realitzades pels microserveis. Cada col·lecció de MongoDB està encriptada i protegida amb contrasenya aportant separació lògica entre la informació.

Per assegurar l'alta disponibilitat de MongoDB, aquest s'executa en clúster i fora de Kubernetes. D'aquesta forma, s'aconsegueix el millor rendiment possible a l'hora d'emmagatzemar la informació.

NOTA: Per aconseguir persistència a Kubernetes cal usar NFS i, tot sovint, comporta compromisos de rendiment.

Per desplegar MongoDB en contenidors, s'utilitza màquines virtuals en clúster prèviament aprovisionades amb Docker Per exemple en un clúster de tres màquines, es procedeix de la següent manera:

1. Desplegament d'una instància primària de MongoDB a els tres màquines usant la imatge oficial de Bitnami:

```
docker run -d --net=host --name mongodb --restart always -e MONGODB_REPLICA_SET_MODE=primary -v mongodb:/bitnami bitnami/mongodb:3.6
```

2. Desplegament d'instàncies secundàries de MongoDB a els tres màquines usant la imatge oficial de Bitnami:

```
docker run -d --net host --name mongodb --restart always -e MONGODB_REPLICA_SET_MODE=secondary
-e MONGODB_PRIMARY_HOST=<PRIMARY_MACHINE_IP > -e MONGODB_PRIMARY_PORT_NUMBER
=27017 -v mongodb:/bitnami bitnami/mongodb:3.6
```

D'aquesta forma es disposa d'una base de dades per a microserveis.

6.3. Kubernetes

Kubernetes[9] és un sistema de codi obert que permet automatitzar el desplegament, l'escalat i el cicle de vida dels contenidors. Seguint una aproximació IAC (Infraestucture As Code) en la solució, Kubernetes és la plataforma de contenidors.

Kubernetes funciona en clúster. Per aprovisionar les màquines del clúster, s'utilitza OpenStack[10] que és una eina que permet automatitzar aquesta tasca. Cada clúster es configura desplegant tots el components d'infraestructura necessaris. A partir d'aquí, el clúster pot allotjar contenidors amb microserveis funcionals i escalar.

Al mateix temps, es fa ús de l'isolament lògic entre recursos mitjançant namespaces. D'aquesta forma un clúster pot allotjar múltiples còpies dels components d'infraestructura i els funcionals. L'ús dels namespaces ens permet en pocs minuts desplegar una còpia exacte de tots els components d'un namespace en un segon namespace del mateix clúster. Aquest mecanisme permet balancejar les peticions entre els dos namespaces maximitzant el rendiment i l'ús dels recursos. Inclús en cas de degradació del propi clúster, la còpia es pot realitzar en un namespace d'un clúster de backup maximitzant la disponibilitat del sistema.

Als següents subapartats, es presenta Kafka Connect, Schema Registry, Kafka MirrorMaker i Kafka Streams. Aquests són els components d'infraestructura que es despleguen a Kubernetes necessaris per usar Kafka com a base de dades distribuïda.

6.3.1. Kafka Connect

Kafka Connect[11] és un framework inclòs en Apache Kafka que permet integrar connectors amb dos funcions: exportar dades de tòpics de Kafka cap a altres sistemes i importar dades cap a Kafka des d'altres sistemes. La funció que ens interessa és la primera, exportar dades des de Kafka cap a altres sistemes, en concret exportar dades cap a la base de dades per a microserveis funcionals, un MongoDB.

En la solució, la unitat de còpia és un connector[12]. Aquest està basat en un connector de codi obert que fa ús del framework Kafka Connect i que està recolzat per la comunitat i per Confluent. Amb lleugeres modificacions sobre el codi obert del connector, es disposa d'una unitat de còpia on, a través d'una API REST, es poden configurar tasques. Aquestes configuracions, simplificant molt, consisteixen a identificar tòpics de Kafka (localització de les dades d'origen) i col·lecció de MongoDB (destinació de les dades d'origen). A continuació es mostra com és una configuració:

POST: http://<MACHINE_IP_MONGODB_CONNECTOR>/connectors

```
{
    "name": "mongo-sink-configurations",
    "config": {
        "connector.class": "at.grahsl.kafka.connect.mongodb.MongoDbSinkConnector",
        "topics": "<KAFKA_TOPIC>",
        "mongodb.connection.uri": "mongodb://<MACHINE_IP_MONGODB>/<COLLECTION>",
        "mongodb.collection": "configurations",
        "mongodb.document.id.strategy": "ProvidedOrGeneratedInValueStrategy",
        "key.converter": "org.apache.kafka.connect.json.JsonConverter",
        "key.converter.schemas.enable": "false",
        "value.converter": "org.apache.kafka.connect.json.JsonConverter",
        "value.converter.schemas.enable": "false",
        "mongodb.delete.on.null.values": "true",
        "mongodb.document.id.strategies": "ProvidedOrGeneratedInValueStrategy"
    }
}
```

6.3.2. Schema Registry

Schema Registry[13] ofereix una interface RESTful per guardar i recuperar esquemes, és de codi obert i està gestionat per Confluent. La particularitat és que els esquemes s'emmagatzemen a Kafka i que tots els missatges es serialitzen utilitzant Avro. Apache Avro[14] és un sistema de serialització d'informació de codi obert gestionat per l'Apache Sotware Fundation.

Disposar d'aquesta eina permet comprovar que un missatge compleix amb un esquema abans de ser enviat. Aquest mecanisme permet reforçar l'organització i estructuració de la informació en els tòpics de Kafka. Per últim, és possible la publicació, l'emmagatzemament i la recuperació de grans quantitats d'informació serializada de Kafka. De totes maneres existeix la possibilitat de publicar, emmagatzemar i recuperar informació en tòpics de Kafka no serializada, la decisió queda en mans dels programadors.

6.3.3. Kafka MirrorMaker

Kafka MirrorMaker[15] és una utilitat que permet copiar la informació d'un clúster de Kafka a un altre. Aquesta utilitat és de codi obert, forma part del propi projecte Apache Kafka. En la solució, és la unitat de replicació del bus de dades.

La utilitat permet configurar la informació a llegir de tòpics del clúster d'origen i escriure-la en tòpics amb el mateix nom al clúster destí. Al mateix temps és escalable i permet executar múltiples instàncies de la utilitat amb la mateixa configuració, augmentar el rendiment i la tolerància de fallada.

6.3.4. Kafka Streams

Kafka Streams[16] és una API que facilita l'ús de streams de dades en temps real. Aquesta API forma part de les API client del projecte Apache Kafka. En la solució és el component principal de la integració amb sistemes externs.

Centrar la implementació dels sistemes d'integració amb sistemes externs en el streaming de dades en temps real i l'emmagazematge a Kafka, és una decisió basada en la volumetria de dades i funcionalitats que s'esperen. Ara bé, no cal reinventar la roda, si ja existeix una API oficial que en facilita el desenvolupament.

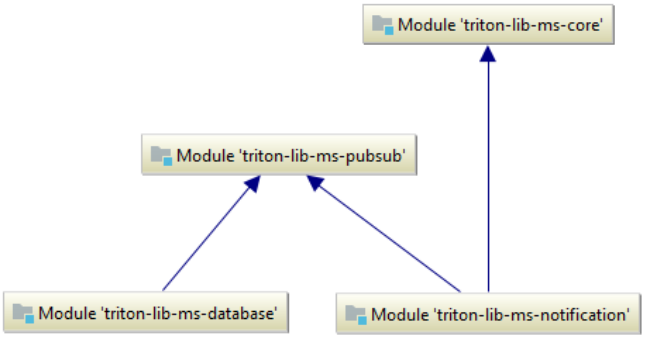
6.4. Intraestructura-framework

El infraestructura-framework és la llibreria que recopila totes aquelles funcions que són comuns per a tots els components i que involucren en algun moment un component d'infraestructura, per exemple Kafka o MongoDB.

L'objectiu principal d'encapsular totes aquestes funcions és definir clarament i de forma única l'accés als components d'infraestructura, per a qualsevol component, sigui funcional o no, que ho necessiti. Per exemple el servei de configuració no és un component funcional, però fa servir el infraestructura-framework per gestionar les configuracions. Al mateix temps, un microservei també fa servir el framework per dur a terme operacions funcionals.

L'objectiu secundari consisteix a simplificar la complexitat, intentant que cada funcionalitat sigui el més transparent possible a l'hora d'usar-la. Això és especialment important quan aquesta llibreria ha de poder ser usada per tercers que no estan familiaritzats amb la infraestructura.

El infraestructura-framework està implementat en Java 1.8 i organitza les funcionalitats en quatre grans mòduls:



Il·lustració 12: Mòduls del infraestructura-framework

- Triton-lib-ms-core: definicions de la configuració més bàsica, com per exemple models, excepcions, aspectes, anotacions o sessions. Aquest és el mòdul més simple però més usat ja que, pràcticament, qualsevol component en farà us.

- Triton-lib-ms-pubsub: funcionalitats essencials per a la comunicació amb Kafka. Aquestes són:
    - pub(tòpic, msg): publicació d'informació a un tòpic Kafka.
    - sub(tòpic, callback): consumició d'informació d'un tòpic de Kafka de forma indefinida.
    - unSub(): stop de la consumició d'informació indefinida.
    - search(tòpic, data): cerca d'informació en un tòpic de Kafka a partir d'una data.
- Triton-lib-ms-database: funcionalitats CRUD (Create, Read, Update, Delete) per la manipulació d'informació sense garantia de persistència a MongoDB, ara bé reaprofitant funcionalitats ja implementades a triton-lib-ms-pubsub:
    - persist(msg): enviament persistent d'informació amb còpia a la base de dades.
    - remove(msg): esborrat d'informació persistent amb esborrat a la base de dades.
- Triton-lib-ms-notifications: funcionalitat d'enviament de notificacions a través de Kafka:
    - sendNotification(tipus, msg): enviament d'una notificació amb un tipus, per exemple alarma o baix, i un missatge.

7. Metodologia

En el desenvolupament de tot el projecte, se segueix una metodologia de desenvolupament Scrum. Scrum és una metodologia de desenvolupament àgil[17] i iterativa que permet fer front a problemes complexos maximitzant el valor del producte.

A l'inici de cada iteració i, segons la disponibilitat de l'equip, s'estableix l'objectiu i es defineixen les tasques amb les quals s'assolirà aquest objectiu. Una iteració dura tres setmanes, al final d'aquestes es valora si s'ha assolit l'objectiu o cal prolongar-lo a la següent iteració o bé repensar l'objectiu. JIRA[18] és el sistema que s'utilitza per la gestió de tasques i iteracions.

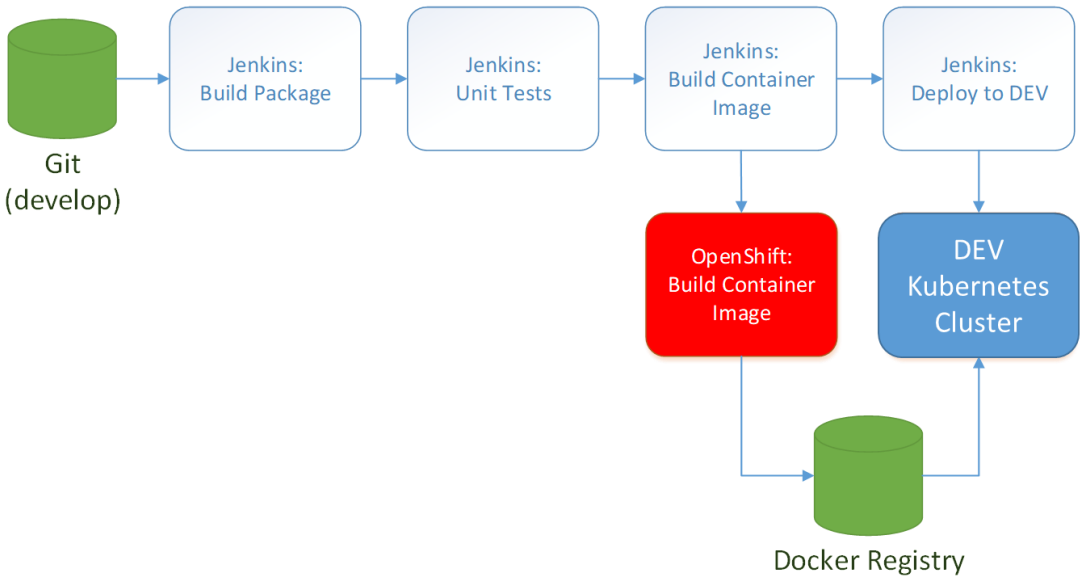
Els desenvolupaments finalitzats són desplegats primerament en un entorn d'infraestructura de desenvolupament (DEV), entorn no productiu exclusiu per a proves de l'equip de desenvolupament, i posteriorment en un entorn d'infraestructura d'User Acceptance Testing (UAT), entorn no productiu on tercers tenen accés a la infraestructura per desplegar codi, tests i identificar errors.

Els desplegaments als entorns de DEV i UAT dels components d'infraestructura finalitzats segueixen uns procediments. A continuació es tracten les restriccions imposades pel procediment i seguidament l'esquema del procediment per cada un dels entorns:

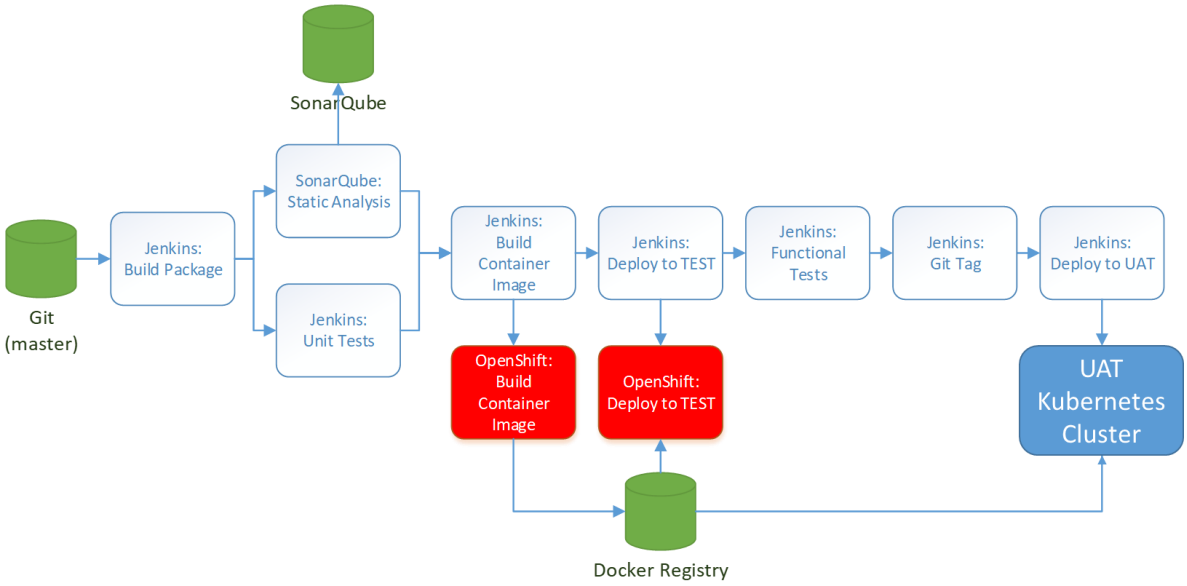
- Control i gestió de versions: el codi resideix a un repositori Git i segueix el model de branques de Git[19].

- Automatització de processos de compilació, test, construcció d'imatges i desplegaments: les tasques de Jenkins[20] permeten automatitzar la majoria d'accions per poder desplegar components:
    - Compilació i tests: la primera tasca és recuperar el codi utilitzant Git, compilar i passar els tests. Es monitoritza que la compilació sigui correcta i que tots els tests es passen.
    - Anàlisi de qualitat de codi: la segona tasca, usant les regles de SonarQube[21], és avaluar el codi produït. A DEV l'anàlisi de la qualitat no impedeix finalitzar un desplegament però a UAT sí que ho imèdeix.
    - Construcció de la imatge pel contenidor: la tercera tasca, usant la API d'OpenShift[22], és construir la imatge immutable que permet executar el component compilat com a contenidor.

- Control i versions d'imatges: la quarta tasca, usant un Docker Registry[23] privat, és guardar la imatge generada al pas anterior.
- Proves funcionals: aquesta tasca és únicament per desplegament en UAT. Consisteix a assegurar que el component no trenca cap funcionalitat.
- Registrar el tag de la versió: aquesta tasca és únicament per desplegament en UAT. Consisteix a marcar al repositori de git el codi amb un número de versió vàlid.
- Desplegament al clúster: la última tasca, als dos entorns consisteix a desplegar el component al clúster de Kubernetes que pertoqui.



Il·lustració 13: Procediment per desplegar a DEV



Il·lustració 14: Procediment per desplegar a UAT

8. Planificació

En el moment en el que s'escriu aquest document, s'han dut a terme quatre iteracions. Cada una d'elles amb un objectiu, a continuació es detalla l'objectiu de cada iteració:

- Primera iteració: analitzar i entendre requisits i proposar un disseny.
- Segona iteració: disposar d'un entorn de desenvolupament amb la infraestructura mínima.
- Tercera iteració: disseny i implementació del infraestructura-framework.
- Quarta iteració: creació d'un segon entorn amb la infraestructura i publicació del infraestructura-framework a tercers.

Pel que fa a les properes iteracions, de moment, estan orientades a solucionar la integració amb sistemes externs, donar suport, estabilitzar la infraestructura i el infraestructura-framework.

9. La meva implicació

La meva implicació al projecte ha estat focalitzada en dur a terme la idea d'usar Kafka com a base de dades distribuïda: des del disseny fins al desplegament i la incorporació de les funcionalitats al framework.

El projecte TRITON sencer està dividit en 15 repositoris, a continuació es llisten els tres repositoris on més he col·laborat:

- Triton-lib-ms-parent: el repositori pare del infraestructura-famework. He estat el segon col·laborador més actiu, amb 50 commits a la branca master.

- Triton-arch-ci-config: el repositori amb tot el codi de la configuració de la infraestructura. He estat el tercer col·laborador amb 7 commits a la branca master.

- Kafka-poc: el repositori amb el codi de mostra per demostrar i testejar Apache Kafka. He estat el col·laborador més actiu amb 17 commits a la branca master.

NOTA: El número de commits ha estat extret al moment d'escriure la memòria sobre la branca master utilitzant "*git shortlog -sne*". No s'han tingut en compte els commits d'administració o automatització realitzats pels usuaris "jenkis" o "everis".

10. Conclusions i treball futur

Al llarg del transcurs d'aquest projecte, podem afirmar que la idea d'usar Apache Kafka com a base de dades distribuïda és possible. Al moment en el que s'escriuen aquestes conclusions, el disseny presentat on Kafka actua com a base de dades distribuïda està implementat satisfactòriament en dos entorns no productius .

Tot hi que l'objectiu és arribar a desplegar un entorn productiu, la feina assolida fins ara solidifica les bases per a què això sigui possible. Durant el transcurs d'aquests mesos també s'han après valuoses lliçons com: pensar i dissenyar en distribuït, entendre les fortaleses i debilitats del núvol de tecnologies usades, produir documentació tècnica de qualitat i, per últim, les responsabilitats associades a actuar com a proveïdor d'una plataforma per a tercers.

El treball futur a fer es pot resumir en dos objectius clarament diferenciats, el primer sortir a producció i el segon l'evolució de la plataforma. El primer objectiu és bàsicament sortir a producció, després de demostrar la viabilitat d'usar Apache Kafka com a base de dades distribuïda. El segon objectiu està més enfocat a millorar el disseny actual de la infraestructura i el infraestructura-framework. Algunes de les primeres possibilitats de millora identificades, pel que fa a la infraestructura, és passar a un model de malla de serveis (service mesh) i, pel que fa al infraestructura-framework, és estandarditzar els streams per a totes les comunicacions amb Kafka i introduir el KSQL[24], una utilitat que potencialment permetria als microserveis fer les operacions sobre dades al mateix Kafka enlloc de a MongoDB.

A nivell professional ha estat un projecte molt interessant des del inici i les tecnologies usades estan a l'ordre del dia. A nivell personal m'enduc la satisfacció d'haver pogut col·laborar, aportar i realitzar aquesta memòria en un projecte com aquest.

Bibliografia

[1] Everis Home Page
https://www.everis.com/global/en
[Consulta: 09/04/18]

[2] Apache Sotware Fundation
https://www.apache.org/
[Consulta: 12/05/18]

[3] Apache Kafka
https://kafka.apache.org/
[Consulta: 12/05/18]

[4] Apache Zookeeper
http://zookeeper.apache.org/
[Consulta: 12/05/18]

[5] NFS – Network File System
https://kubernetes.io/docs/concepts/storage/volumes/
[Consulta: 12/05/18]

[6] Docker Home Page
https://www.docker.com/
[Consulta: 12/05/18]

[7] Confluent
https://www.confluent.io/about/#about_confluent
[Consulta: 14/05/18]

[8] MongoDB Home Page
https://www.mongodb.com/
[Consulta: 12/05/18]

[9] Kubernetes Home Page
https://kubernetes.io/
[Consulta: 14/05/18]

[10] OpenStack Home Page
https://www.openstack.org/
[Consulta: 14/05/18]

[11] Kafka Connect
https://www.confluent.io/product/connectors/
[Consulta: 14/05/18]

[12] Imatge del Connector
https://hub.docker.com/r/jlarriba/kafka-connect/
[Consulta: 14/05/18]

[13] Shcema Registry
https://www.confluent.io/confluent-schema-registry/
[Consulta: 14/05/18]

[14] Apache Avro
https://avro.apache.org/index.html
[Consulta: 14/05/18]

[15] Kafka MirrorMaker
https://kafka.apache.org/documentation/#basic_ops_mirror_maker
[Consulta: 14/05/18]

[16] Kafka Streams
https://kafka.apache.org/documentation/streams/
[Consulta: 14/05/18]

[17] Agile Manifesto
http://agilemanifesto.org/
[Consulta: 26/05/18]

[18] JIRA
https://es.atlassian.com/software/jira
[Consulta: 28/05/18]

[19] Git branch model
https://www.atlassian.com/git/tutorials/comparing-workflows
https://nvie.com/posts/a-successful-git-branching-model/
[Consulta: 28/05/18]

[20] Jenkins
https://jenkins.io/
[Consulta: 28/05/18]

[21] SonarQube
https://www.sonarqube.org/
[Consulta: 28/05/18]

[22] OpenShift
https://www.openshift.com/
[Consulta: 28/05/18]

[23] Docker Registry
https://docs.docker.com/registry/
[Consulta: 28/05/18]

[24] KSQL
https://www.confluent.io/blog/ksql-open-source-streaming-sql-for-apache-kafka/
[Consulta: 28/05/18]