 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	<i>Arquitectura de referència microserveis</i>	N. revisió doc.: <i>0.7</i>
	<b>Descripció de l'arquitectura de referència</b>	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 1 / 58


Revisió	Redactat per	Revisat per	Aprobat per	Data aprovació	Data publicació

<b>Registre de canvis del document</b>
--

Revisió	Apartat	Data	Redactat per	Canvis
V0.1		24/02/2021	Minsait	Primer esborrany
V0.2		01/03/2021	Minsait	Canvi ordenació punts de l'índex. Apartat 5.1.2.2: Incorporades imatges d'arquitectura d'execució/operació. Eliminades referències a les solucions Azure a l'arquitectura de desenvolupament Apartat 6.1: S'incorporen gaps de desplegaments infra. com a codi i plataformes d'orquestració en cloud pública. Revisió de descripcions. S'afegeixen títols a imatges
V0.3		08/03/2021	Minsait	Revisió de redactats segons comentaris Revisió gràfic conceptualització arquitectura de referència Títol a il·lustracions
V0.4		22/03/2021	Minsait	Revisió de redactats segons comentaris Afegeides referències Revisió gràfics Afegeides conclusions PoC
V0.5		24/03/2021	Minsait	Revisió de comentaris i re-estructuració index
V0.6		30/03/2021	Minsait	Revisió de comentaris
V0.7		30/03/2021	Josep Antoni Mira	1a versió a publicar per revisió unitat d'arquitectura


<b>RESPONSABLE DEL DOCUMENT:</b> Minsait
--

<b>ARQUITECTE CTTI:</b> Josep Antoni Mira
---

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	<i>Arquitectura de referència microserveis</i>	N. revisió doc.: <i>0.7</i>
	<b>Descripció de l'arquitectura de referència</b>	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 2 / 58

## Í N D E X

1.	<b>Introducció .....</b>	<b>3</b>
1.1	Conceptualització de l'arquitectura de referència .....	3
2.	<b>Arquitectura de referència de microserveis .....</b>	<b>6</b>
2.1	Criteris d'aplicabilitat .....	6
2.2	Principis de l'arquitectura de referència .....	9
2.3	Models d'ús de l'arquitectura .....	10
2.3.1	Model de disseny funcional – distribució en microserveis .....	10
2.3.2	Model de DevOps.....	14
2.4	Arquitectura tècnica .....	23
2.4.1	Disseny conceptual.....	23
2.4.2	Disseny lògic .....	29
2.4.3	Disseny físic .....	47
3.	<b>Procés de creació de l'arquitectura de referència .....</b>	<b>49</b>
3.1	Principis de definició de l'arquitectura de referència .....	49
3.1.1	Orientació a l'autonomia .....	49
3.1.2	Contemplar que l'arquitectura ha d'evolucionar junt amb la tecnologia.....	49
3.2	Decisions del procés de creació de l'arquitectura .....	49
3.3	Prova de concepte .....	50
3.3.1	Bloc funcional PetStore .....	51
3.3.2	Bloc funcional de mascotes .....	52
3.3.3	Bloc funcional de comandes.....	53
4.	<b>Gap respecte a la situació actual i serveis potencialment comuns .....</b>	<b>55</b>
4.1	Gap respecte a la situació actual .....	55
4.2	Serveis potencialment comuns .....	56
5.	<b>Referències .....</b>	<b>58</b>

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 3 / 58

## 1. Introducció

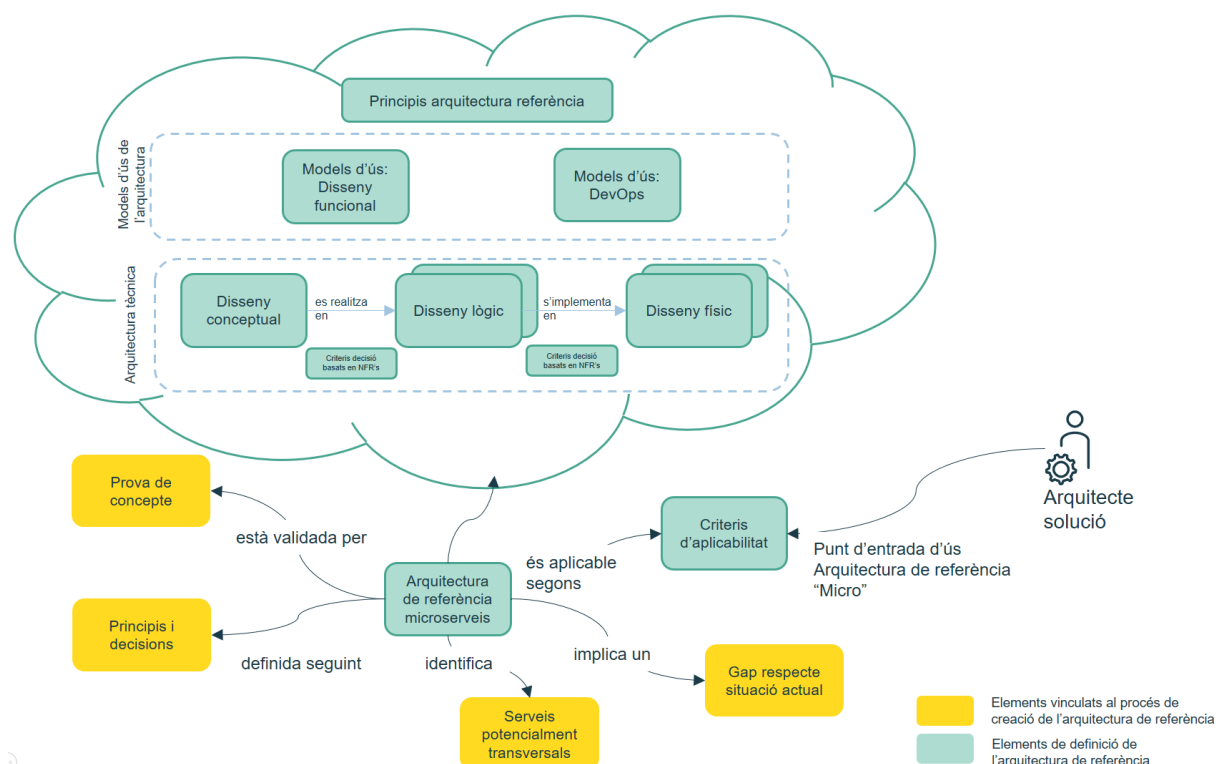
Davant la ràpida evolució dels estils d'arquitecturals, CTTI identifica la necessitat d'establir una arquitectura de referència per aquelles aplicacions que opten per arquitectures altament distribuïdes tant a nivell lògic com a nivell físic.

CTTI vol disposar d'una arquitectura de referència orientada al desplegament d'aplicacions en microserveis, microfrontends i funcions el que implica habilitar un ecosistema de components addicionals que suportin aquest estil d'arquitectura. Aquesta arquitectura de referència ha de permetre homogeneïtzar l'aplicació d'aquest tipus d'arquitectures entre els diferents proveïdors evitant, d'aquesta forma, la dispersió tecnològica.


### 1.1 Conceptualització de l'arquitectura de referència

El concepte d'arquitectura de referència pot ser entès des de múltiples perspectives i, des de cadascuna d'elles, la definició pot diferir. Un arquitecte de sistemes ho pot entendre com una definició d'infraestructura parametrizable segons valors de consum específics d'una solució i un arquitecte de software com un conjunt d'stacks de tecnologia possibles a usar per desenvolupar una solució. Hi ha certa ambigüitat en el concepte que aquest apartat té l'objectiu d'aclarir pel cas concret de l'arquitectura de referència de microserveis.

L'arquitectura de referència presentada en aquest document està formada pels conceptes mostrats a la següent figura



II-lustració 1 Model d'arquitectura de referència

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 4 / 58

La definició pròpiament de la “Arquitectura de referència de microserveis” està formada per dos grans blocs que serien els models d'ús de l'arquitectura i l'arquitectura tècnica. Si partim del model d'una descripció d'arquitectura segons el vist del document DA de CTTI, el primer bloc ens servirà de referència per les vistes més funcionals, de desenvolupament i d'operació i el segon per la vista de desplegament d'una descripció d'arquitectura.

El bloc de l'arquitectura tècnica està format per:

- **Disseny conceptual:** Identifica els components de l'arquitectura de referència des d'un punt de vista neutre tecnològicament definint quines capacitats aporten a l'arquitectura.
- **Disseny lògic:** És una realització tecnològica del disseny conceptual i explicita les tecnologies possibles a usar per realitzar cadascun dels components definits al disseny conceptual. Un disseny conceptual pot tenir múltiples realitzacions tecnològiques diferents.
- **Disseny físic:** És la implementació d'un disseny lògic sobre una arquitectura física concreta, per exemple, sobre on-premise, clouds públiques, etc. De forma similar al cas anterior, un disseny lògic pot tenir varies implementacions físiques.

El disseny lògic i físic dels sistemes d'informació depenen fortament del seu context concret. Els que es reflecteixen en aquesta arquitectura estan dissenyats i implementats amb un propòsit d'exemplificar l'ús de l'arquitectura de referència de microserveis. Els criteris de decisió que porten a un arquitecte a decidir un disseny lògic i un disseny físic queden sota la seva àrea de responsabilitat. Tenint en compte aquest escenari, l'arquitectura de referència pretén ajudar a aquesta presa de decisions.

Els models d'ús presenten les recomanacions i guies per usar l'arquitectura tècnica i realitzar el disseny de la solució de forma que aprofiti les qualitats d'aquesta. Es divideixen en:


- **Model d'ús de disseny funcional:** Proporciona guies i recomanacions per, principalment, generar la vista funcional i informacional del disseny de la solució.
- **Model d'ús DevOps:** Proporciona guies i recomanacions per, principalment, la vista de desenvolupament i d'operació del disseny de la solució.

Ambdós blocs estan complementats per:

- **Principis de l'arquitectura de referència :** Els principis són els valors fonamentals que s'apliquen en l'arquitectura i que guien els dos models d'ús indicats anteriorment.
- **Criteris d'aplicabilitat:** Inclou criteris que permeten fer una discriminació inicial de quines solucions poden ser candidates a usar l'arquitectura de referència i, per tant, treure'n benefici de les seves qualitats.

Entenem que aquests elements descrits fins aquí són els utilitzables per un arquitecte de solució per la presa de decisió sobre l'ús de l'arquitectura de referència i, en cas afirmatiu, els elements en els quals basar-se per fer la seva descripció d'arquitectura (DA).


Al concepte central “Arquitectura de referència micro” i els criteris d'aplicabilitat s'afegeixen dos elements que també entenem formen part de la definició de qualsevol arquitectura de referència. A diferència dels anteriors, aquests no estan orientats al seu ús per un arquitecte de solució sinó que formen part de l'exercici de definició:

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 5 / 58

- **Prova de concepte:** Solució desenvolupada seguint l'arquitectura de referència que serveix per exercitar l'arquitectura i validar-ne la definició.
- **Principis i decisions:** Són els principis rectors que s'han seguit al definir l'arquitectura de referència així com les decisions principals preses durant el seu disseny.

Addicionalment s'han incorporat dos aspectes que, tot i no complementar pròpiament la definició de l'arquitectura de referència, són rellevants per la seva implantació al CTTI:

- **Serveis d'arquitectura potencialment comuns:** Indica quins components de l'arquitectura de referència pot tenir sentit que siguin compartits per múltiples solucions i, per tant, convertir-se en serveis comuns.
- **Gap respecte a la situació actual:** Revisa quines necessitats d'evolució es deriven de l'arquitectura de referència respecte a la plantilla de descripció d'arquitectura (DA), el sistema d'integració contínua (SIC) i els components actualment de catàleg.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 6 / 58

## 2. Arquitectura de referència de microserveis

### 2.1 Criteris d'aplicabilitat

L'ús de l'arquitectura aporta avantatges, especialment, relacionats amb la capacitat de lliurar canvis i evolucions de la solució més ràpidament als usuaris però també introdueix dificultats i complexitats que fan que la decisió sobre l'ús d'aquest tipus d'arquitectura sigui rellevant.


A banda de l'avantatge principal de la rapidesa en quant a canvis podem identificar-ne, a nivell d'exemple, d'altres:

- Permet l'ús de les tecnologies més adequades per les funcionalitats de l'aplicació tant a nivell de llenguatges de programació com de plataformes de base.
- Facilita l'evolució tecnològica de l'aplicació mitigant, d'aquesta forma, el risc d'obsolescència tecnològica.
- Escala més eficientment ja que permet una distribució més adequada de recursos

En quant a les dificultats que es deriven poden enumerar les següents:

- Més complexitat en els processos de DevOps pel fet de l'existència de més components desplegats
- El diagnòstic d'errors pot ser més complex degut a la intervenció de múltiples components per resoldre una interacció
- Riscos derivats de les architectures distribuïdes per ser més proclius a errades de les comunicacions subjacents.

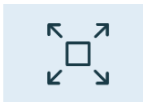
És per aquest motiu que s'estableixen uns **criteris d'aplicabilitat** per ajudar a la presa de decisió sobre l'ús de l'arquitectura. Aquests criteris es divideixen en **motivacions** que poden fer que una arquitectura de microserveis pugui ser beneficiosa per a un sistema i **condicionants** que hauria de complir el sistema per a poder-la aplicar amb garanties d'èxit.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 7 / 58

## Motivacions



**Rapidesa en l'entrega d'evolucions:** La solució cal que pugui evolucionar de forma ràpida i lliurar noves funcionalitats en períodes curts de temps.



**Altes necessitats d'escalabilitat:** L'estil arquitectural permet una escalabilitat més granular escollint quins components tenen més necessitats de recursos que d'altres. Per aplicacions on es preveuen alts creixements d'usuaris aquesta característica és un valor a considerar.



**Convivència múltiples tecnologies:** La solució té requeriments que es poden resoldre de forma més eficient usant diferents tecnologies.



**Orientació a la reutilització:** La solució es pensa des de l'inici orientada a que certs components siguin reutilitzables per d'altres solucions. L'existència de límits clars entre els diferents components que formen la solució faciliten mantenir la modularitat i el seu posterior reaprofitament.




**Alt desacoblament d'equips de desenvolupament:** La solució és gran i hi ha parts de la solució desenvolupades per equips organitzativament separats, per exemple, diferents proveïdors, etc.







**Adaptabilitat a canvis de context no previstos:** La solució ha de poder reaccionar ràpid a canvis en el context extern no previstos, per exemple, pics alts de demanda derivats d'una situació de pandèmia, etc.

### II-lustració 2 Motivacions


Per d'altre banda, l'ús de l'arquitectura també implica uns condicionants que es poden classificar en diferents aspectes: solució, procés de construcció, equip i infraestructura de desplegament.

 <b>Generalitat de Catalunya</b> <b>Centre de Telecomunicacions</b> <b>i Tecnologies de la Informació</b>	<i>Arquitectura de referència microserveis</i>	N. revisió doc.: <i>0.7</i>
	<b>Descripció de l'arquitectura de referència</b>	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 8 / 58

Aspecte	Condicionant	Descripció del condicionant
<b>Equip</b>  	Coneixement del domini funcional	L'adequada descomposició funcional de la solució requereix d'un coneixement important del domini funcional. Cal poder identificar clarament totes les funcionalitats de l'aplicació i poder-les agrupar de forma que siguin altament cohesives i minimitzant l'acoblament
	Disponibilitat de capacitats d'arquitecte funcional	Les vistes funcionals i informacionals d'una descripció d'arquitectura prenen molta rellevància i el perfil purament d'analista funcional no acostuma a tenir la perspectiva d'arquitectura necessària. El perfil d'arquitecte altament tecnòleg pot no sentir-se còmode en el detall funcional de la solució
	Experiència en arquitectures distribuïdes	L'arquitectura és altament distribuïda i aquest fet implica complexitats que no són presents en arquitectures monolítiques, per exemple, els errors derivats per indisponibilitats puntuals de serveis o els errors d'interaccions per problemes de comunicacions
	Disponibilitat de capacitats DevOps	Les processos d'integració contínua i monitorització de la solució incrementen la seva importància, complexitat i cal invertir-hi més esforç que en arquitectures monolítiques. El perfil ha d'aportar coneixements i experiència en la plataforma de desplegament (orquestradors de contenidors, etc.)
<b>Solució</b> 	Aplicació operacional	L'estil d'arquitectura no està pensat per solucions orientades a analítica de dades, business intelligence, etc. Tot i que s'està introduint el concepte de data mesh que coincideix a molt alt nivell amb l'estil d'arquitectura, no és aplicable
<b>Infraestructura</b> 	Disponibilitat d'una adequada plataforma d'execució	S'ha de disposar, en el CPD destí, d'una plataforma adequada de desplegament que doni resposta al disseny conceptual requerit per l'arquitectura, per exemple, proporcionant serveis de control i parada de serveis, balanceig de càrrega, etc.
	És clau disposar d'eines de monitorització	El disposar de múltiples components desplegats dificulta la monitorització i observabilitat global de la solució si no es disposa de les eines adequades
<b>Procés de construcció</b> 	Incremental i iteratiu	És important que el procés de construcció faci créixer progressivament la solució amb l'objectiu de poder anar confirmant les decisions d'arquitectura preses com, per exemple, les relacionades amb la descomposició funcional dels serveis.
	Ús d'eines d'automatització de la construcció, desplegament i testing	L'elevada quantitat d'artefactes a generar i desplegar fan inviable i molt susceptible a errors fer-ho de forma manual i és molt important l'ús d'eines d'automatització.

### II-lustració 3 Condicionants



 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 9 / 58


## 2.2 Principis de l'arquitectura de referència

L'objectiu principal de l'estil arquitectural descrit és **accelerar l'entrega de funcionalitats** de les solucions facilitant les pràctiques d'entrega i desplegament continuats. És important tenir en consideració que, tot i que l'estil d'arquitectura faciliti aquest punt, no és l'únic ingredient necessari. L'organització dels equips de desenvolupament i la metodologia utilitzada també tenen un pes molt important.

Des de la perspectiva de l'arquitectura, aquest objectiu s'aconsegueix, principalment, permetent que tot el **cicle de vida d'un component sigui el màxim independent possible de la resta**. Aquest punt s'alinea amb una organització on un equip sigui responsable, d'extrem a extrem, des dels requeriments fins l'operació, de cadascun d'aquests components.

Per aconseguir aquest objectiu és necessari establir uns **models d'ús de l'arquitectura** ja que disposar d'una adequada arquitectura tècnica no és, en cap cas, suficient per aconseguir-ho. Els models d'ús s'han estructurat amb model de disseny funcional i model DevOps. En el primer, es descriu l'aproximació al disseny funcional de la solució i, en el segon, com aquest disseny funcional es trasllada al desenvolupament i a l'operació.

Els dos models han estat guiats, prèviament, per uns principis que, de forma general, marquen quin ús s'ha de fer de l'arquitectura per treure'n el màxim profit de la mateixa. Els podem entendre com un visió més sintètica dels models desenvolupats a continuació

 Generalitat de Catalunya <b>Centre de Telecomunicacions i Tecnologies de la Informació</b>	<i>Arquitectura de referència microserveis</i>	N. revisió doc.: <i>0.7</i>
	<b>Descripció de l'arquitectura de referència</b>	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 10 / 58

#### Preferència de descomposició funcional orientada a domini funcional

- La descomposició funcional d'una solució en diferents blocs es pot realitzar des de diferents perspectives, per exemple, al voltant de les dades o de la tecnologia. La descomposició funcional preferent és la orientada a domini de negoci aplicant tècniques, per exemple, de Domain-driven design.
- És preferible començar amb blocs funcionals grans que es descomposen més a mida que es coneix més en detall el domini que començar amb blocs funcionals molt granulars.

#### Alta cohesió i baix acoblament

- Els principis que han de guiar el disseny funcional són els d'aconseguir blocs funcionals amb un baix acoblament, és a dir, que el canvi en un d'ells no impliqui canvis a la resta i una alta cohesió entesa com que el codi que es modifica junt ha d'estar arquitecturalment junt.

#### Automatitzar construcció, proves i desplegament

- L'automatització en els cicles de desenvolupament, proves i desplegament esdevé clau. La complexitat de gestionar un nombre gran d'artefactes i una manca d'automatització pot comprometre seriosament un dels objectius principals de l'arquitectura com és el d'accelerar l'entrega de software als usuaris.

#### Cicle de vida independent

- L'ús de l'arquitectura ha de mantenir que els diferents blocs funcionals puguin ser desenvolupats, provats, desplegaments i documentats de forma independent uns d'altres. Això accelera l'entrega i pot facilitar la repartició dels mateixos entre diferents equips de desenvolupament permetent, d'aquesta forma, una elevada escalabilitat en el procés de construcció i entrega. En aquesta línia, equips centralitzats que puguin esdevenir colls d'ampolla s'haurien d'evitar.

#### L'operació com a “first-class citizen” de la solució

- El baix acoblament dels diferents blocs funcionals ha de mantenir-se a nivell d'execució de forma que hi hagi tolerància a fallades entre ells i aïllament de recursos per evitar interferències mútues. La monitorització i observabilitat de la solució s'ha de considerar de forma global ja que la monitorització individual de cada bloc funcional no dona una visió completa de la solució.

### II·lustració 4 Principis d'ús

## 2.3 Models d'ús de l'arquitectura


### 2.3.1 Model de disseny funcional – distribució en microserveis

L'objectiu del disseny funcional és la distribució adequada del conjunt de funcionalitats de la solució en diferents blocs funcionals. Una adequada distribució de les funcionalitats ha d'aconseguir que la major part de requeriments d'evolució de la solució impliquin la modificació d'un únic bloc funcional.

Llavors, quina mida ha de tenir un bloc funcional? Podríem dir que ha de ser tant petit com es pugui i tant gran com els requeriments necessitin.

D'una forma més concreta i per descriure millor aquest equilibri, un bloc funcional quant més petit és, més gestionable, ja que:

- Redueix el risc en els desplegaments donat que el canvi està molt acotat.
- Les proves de regressió són més àgils ja que el codi desplegat és menor.
- L'enteniment de la seva funcionalitat és més simple.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 11 / 58

però, per d'altra banda, una descomposició en blocs funcionals excessivament petits pot provocar que l'evolució derivada d'un requeriment de negoci impacti a varis. Això implica múltiples desplegaments, possible coordinació entre equips de desenvolupament, etc. el que allunya de l'objectiu primer d'aquest tipus d'arquitectura.

Els principis bàsics que han de guiar la descomposició funcional són els ja identificats i desenvolupats fa dècades per Larry Constantine [YOU79] en els seus treballs de disseny estructurat: **alta cohesió** i **baix acoblament**.

#### *Baix acoblament*

Quan un bloc funcional està poc acoblat amb un altre, un canvi en un d'ells no implica canviar l'altre. Què porta a un bloc funcional a estar més acoblat amb un altre? Per exemple;

- Estils d'integració dependents de tecnologia (RMI, etc)
- Necessitat de conèixer molta informació de l'altre bloc funcional
- Requerir de la invocació de moltes operacions diferents d'un altre bloc funcional

#### *Alta cohesió*

El codi que es modifica junt, que dona resposta a una funcionalitat concreta, ha d'estar arquitecturalment junt. Des d'aquesta perspectiva, poden haver diferents criteris per establir el perímetre dels blocs funcionals, per exemple:


- **Domini de negoci:** S'agrupen les funcionalitats definint els perímetres alineats amb les funcions identificades a nivell del negoci.
- **Organitzacional:** S'agrupen les funcionalitats alineant amb l'estructura dels equips de desenvolupament i les seves capacitats de treball conjunt i comunicació. Per exemple, en una solució on intervenen múltiples proveïdors podria tenir sentit modularitzar de forma que cadascun pugui ser autònom en la seva part.
- **Dades:** Es poden agrupar totes les funcionalitats que tenen relació amb un tipus de dada concreta, per exemple, dades bancàries candidates a haver de passar processos d'auditoria específics.
- **Tecnologia:** S'agrupen les funcionalitats segons la tecnologia que s'usa pel seu desenvolupament. Aquest enfocament pot tenir sentit en contextos on hi ha unes tecnologies clarament diferenciades per resoldre certes funcionalitats.

De forma general i salvant solucions amb necessitats molt específiques, el criteri de preferència és el domini de negoci i, en segona instància, l'organitzacional.

#### **2.3.1.1 Tècniques per definir perímetres segons el domini de negoci**

La descomposició basada en aquest criteri requereix un alt coneixement funcional del domini i l'aplicació dels principis bàsics de disseny identificats en l'apartat anterior, és a dir, definir perímetres que tinguin un baix acoblament i una alta cohesió.

L'avantatge d'aquest criteri de descomposició és que existeixen tècniques o mètodes que poden facilitar la identificació adequada d'aquests perímetres sobresortint, entre aquestes, el **Domain-driven design** (DDD) [EVA04] i [VER13].

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 12 / 58

Domain Driven Design (DDD) és una aproximació al disseny de software que posa en el centre de l'activitat de disseny el domini de negoci. Els conceptes clau que se'n deriven serien:

- Domini: És el problema de negoci que s'està analitzant per a donar la solució.
- Ubiquitous Language: Es el llenguatge amb el qual els experts del negoci el descriuen i que ha de servir de nexa d'unió entre la descripció del domini i la codificació de la solució.
- Bounded context: És el concepte que proporciona DDD per acotar els diferents dominis.

La identificació de "bounded contexts" es correspon a la descomposició funcional que estem dissenyant. A banda de DDD, també s'han desenvolupat d'altres tècniques que ajuden a fer aquesta descomposició com, per exemple, **l'event storming** [BRA13] que és un exercici de brainstorming col·laboratiu que permet, també, delimitar aquests perímetres funcionals.

Per detall d'ambdós tècniques, consultar la bibliografia referenciada.

### 2.3.1.2 Validació de la descomposició funcional

Tal com s'ha detallat a l'apartat, una adequada descomposició funcional és clau per poder aconseguir els objectius que persegueix l'arquitectura. Tot i que la millor forma d'assegurar que s'ha realitzat correctament és tenir un ampli coneixement del domini, aquí fem menció a alguns aspectes que poden ajudar a validar si la idoneïtat de la descomposició.

Per exemplificar ambdós aspectes ens basarem en una aplicació hipotètica Petstore que s'ha dividit en dos blocs funcionals:


- Botiga: Conté les funcionalitats que realitzar una comanda de compra sobre alguna mascota disponible així com el seguiment de les comandes per part del client i empleats de la botiga.
- Mascotes: Conté les funcionalitats de mostrar el catàleg de mascotes disponibles per vendre així com l'administració de les mateixes per part dels empleats de la botiga.

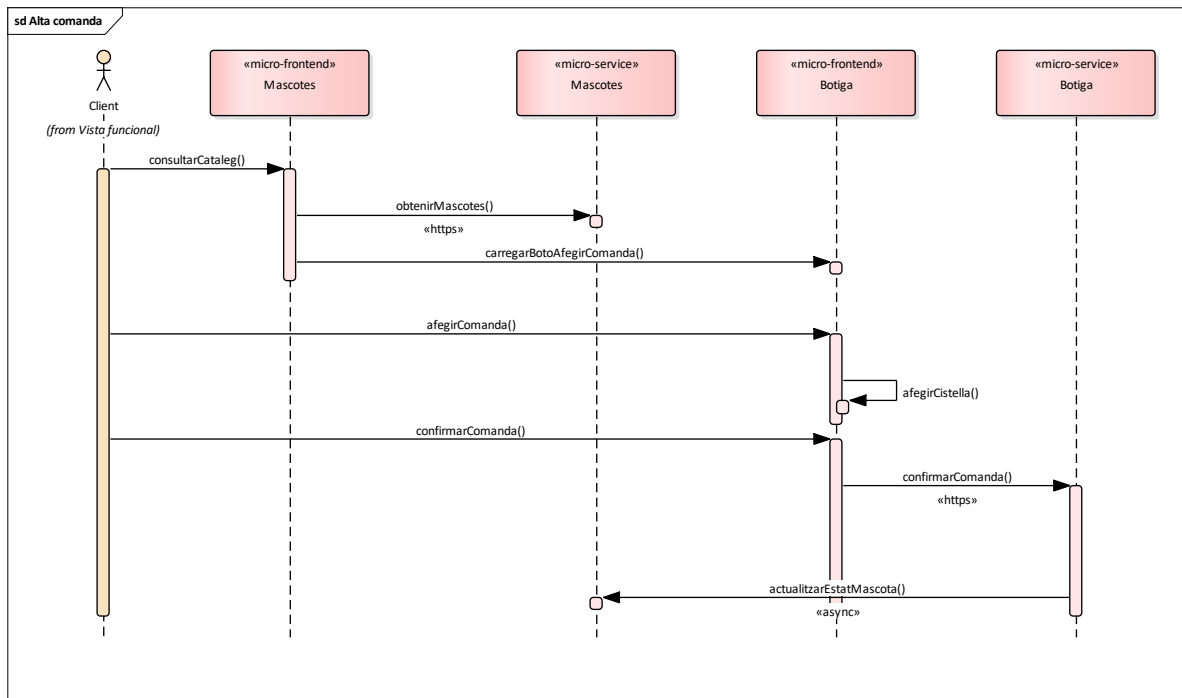
#### "Chatty communications"

En una arquitectura distribuïda les comunicacions entre components tenen un cost (rendiment, risc de fallida, etc) molt més elevat que en d'altres arquitectures. És per això que la necessitat d'una gran quantitat d'interaccions entre blocs funcionals per resoldre els escenaris d'ús principals d'una solució pot ser un indicador d'una descomposició que pot ser problemàtica.

La traducció dels escenaris d'us principals de la solució en diagrames de seqüència on es pugui veure les interaccions necessàries per resoldre'ls entre blocs funcionals es considera una bona eina per detectar problemes en el disseny.

A nivell d'exemple, es mostra el diagrama de seqüència de l'escenari d'alta de comanda a la PetStore on intervenen el bloc funcional de "Mascotes" i de "Botiga".

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 13 / 58



II-lustració 5 Diagrama de seqüència


Al diagrama es veuen clarament dos punts d'interacció entre ambdós blocs funcionals i que no hi ha un nombre gran d'interaccions entre ells per resoldre l'escenari.

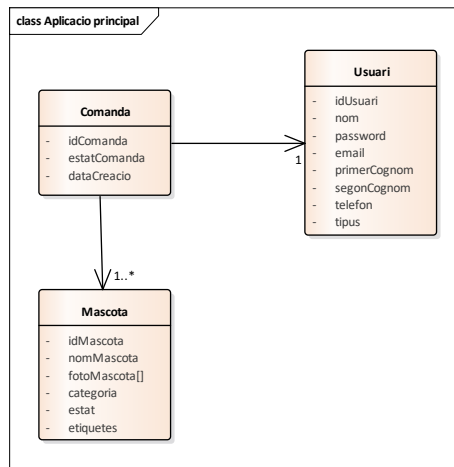
### Models compartits

La descomposició funcional de la solució també implica la descomposició dels models d'informació i la distribució del model global de la solució entre els diferents blocs funcionals propietaris. En aquesta descomposició del model d'informació tots els blocs funcionals poden requerir compartir parts del seu model amb d'altres.

Si la quantitat d'informació a compartir implica que s'ha de replicar bona part del model entre blocs funcionals també és un indicatiu d'una descomposició funcional problemàtica.

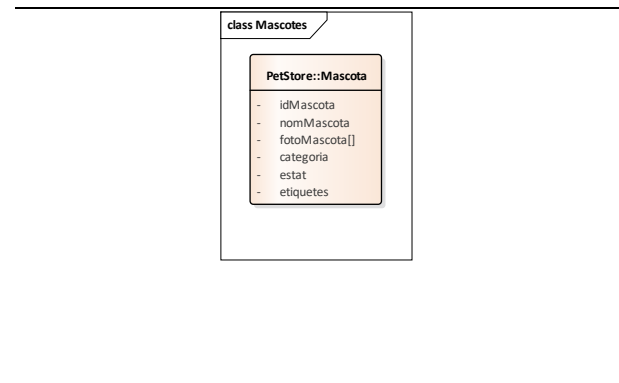
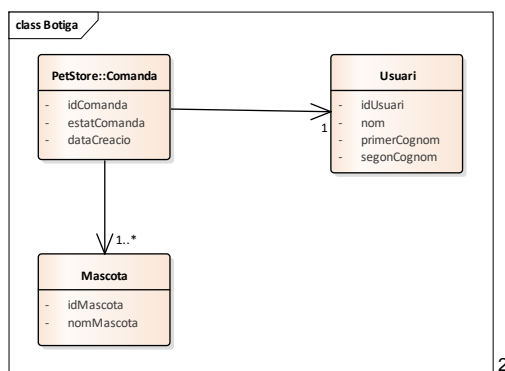
A nivell d'exemple, es mostra el diagrama global de dades de la PetStore i els diagrames dels blocs funcionals de "Mascotes" i de "Botiga"

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 14 / 58



II-lustració 6 Diagrama global PetStore

En els gràfics següents es pot veure com hi ha certes dades, concretament el nom de la mascota, que està en ambdós models però, en cap cas, hi ha cap rèplica de tot el model de dades. La “Botiga” únicament requereix un petit conjunt de dades de les mascotes i d'usuari per donar servei.



## 2.3.2 Model de DevOps


### 2.3.2.1 Model de desenvolupament

#### 2.3.2.1.1 Patró principal de desenvolupament (BFF – Backend for frontend)

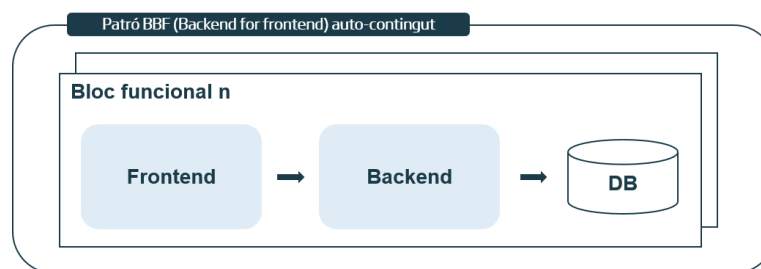
Per a cada bloc funcional, el desenvolupament hauria de seguir una metodologia BFF (Backend for frontend) adaptada a una arquitectura de microserveis.

Com a conseqüència, per a cada bloc funcional s'hauria de crear un únic projecte, que contindrà tant la part front com la part back. Això té l'avantatge que cada bloc funcional podria

<sup>2</sup> A nivell d'exemple s'ha pres la decisió de disseny de replicar l'atribut nomMascota degut a que aquest, s'assumeix, canvia poc i per no haver de sol·licitar-lo a cada visualització al servei de mascotes.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 15 / 58

ser desenvolupat per equips diferents i independents. A més, cada bloc funcional hauria de disposar de la seva pròpia base de dades.

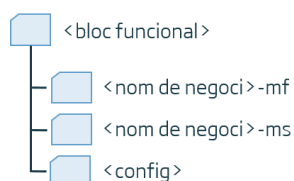


II-lustració 7 Patró Backend for frontend


- **Frontend:** Conjunt de micro-frontends que proporcionen les funcionalitats del bloc al usuari.
  - Només accedeix al seu backend.
  - Alt acoplament del front-end al backend
- **Backend:** Servei que implementa la lògica de negoci per donar resposta al seu frontend
  - Si cal, pot fer crides a serveis backend d'altres microserveis
- **DB:** Base de dades pròpia per a cada bloc funcional

### 2.3.2.1.2 Estructura de projecte

Per poder traçar el concepte de bloc funcional a través de totes les capes, en el repositori de codi font, els diferents components del projecte s'estructuraran dins d'un mateix projecte GIT organitzats en carpetes. S'hauria de crear una carpeta per al codi de la part front i una altra per la part back. A més, s'hauria de crear una carpeta per a la configuració que hauria de contenir les plantilles i les configuracions dels components d'infraestructura dels que està compost el bloc funcional. Això té com a conseqüència que les capes front i back del mateix bloc funcional tindran el mateix cicle de vida a nivell de desenvolupament, versionat i desplegament. Un exemple de l'organització d'un projecte GIT d'un bloc funcional podria ser la següent:



II-lustració 8 Estructura de projecte

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 16 / 58

- Avantatges:
  - La gestió del codi i de la configuració es simplifica a causa de que tot està en el mateix projecte
  - El codi i la configuració es versionen conjuntament
- Desavantatges:
  - Un canvi de configuració requereix recompilar el projecte generant una nova versió de la imatge.

Si algun requeriment no funcional requereix que en el bloc funcional hagi més d'un microservei o algun altre artefacte d'algun altre tipus es afegirà a la seva carpeta corresponent dins el projecte GIT del bloc funcional. De la mateixa manera, si per requeriments de projecte fos necessari que diferents components del mateix bloc funcional tinguin cicles de vida independents, es podria dividir el bloc funcional en diversos projectes GIT

#### **2.3.2.1.3 Descomposició funcional de l'estructura del projecte**

En el cas en què un microservei d'un bloc funcional tingui requisits no funcionals particulars que diferenciïn o identifiquen dues o més parts diferenciades dins del microservei, es recomana dividir el microservei en diferents components (dins del mateix bloc funcional) per poder així aplicar la particularitat cadascun (p.e. escalat diferenciat). Aquesta descomposició es podria realitzar de dues formes diferents:


- Dins el mateix projecte GIT: Per a cada component identificat es podrien crear carpetes separades dins del mateix projecte GIT. Les plantilles de configuració de tots els components dels que es compon el projecte GIT estarien tots a la mateixa carpeta de configuració. Això té com a conseqüència que tots els components del projecte GIT tindrien el mateix cicle de vida a nivell de desenvolupament, versionat i desplegament.
- Projectes GIT separats: Es podria descompondre el bloc funcional en diversos projectes GIT. D'aquesta manera els components dels diferents projectes GIT tindrien cicles de vida separats i diferenciats a nivell de desenvolupament, versionat i desplegament.

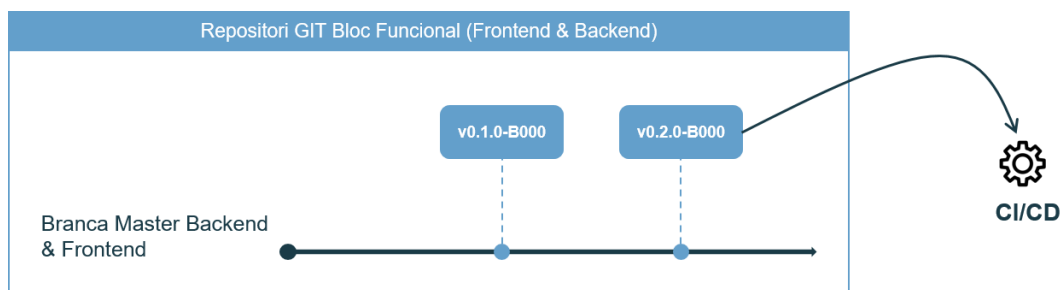
Els requeriments de cada projecte determinaran l'estratègia a seguir per descompondre un bloc funcional en components

#### **2.3.2.1.4 Versionat**

S'hauria de crear una única versió que afecta al mòdul funcional complet per igual, és a dir, el cicle de vida de les diferents capes del bloc funcional es gestiona amb una única versió de bloc funcional. A causa de que les plantilles de configuració formen part del mateix projecte, aquestes plantilles també es versionen conjuntament amb el codi de les capes front i back. Per tant, el codi de les dues capes i la configuració tindran el mateix cicle de vida a nivell de desenvolupament, versionat i desplegament.



 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 17 / 58



Il·lustració 9 Versionat

Si per requeriments del projecte s'ha hagut de dividir el bloc funcional en diversos projectes GIT, cada projecte s'hauria de versionar per separat, de manera que cada projecte tindrà un propi cicle de vida.


#### 2.3.2.1.5 Pipelines

Per a cada bloc funcional s'han de crear les següents pipelines:

- Una pipeline principal que construeix i desplega el bloc funcional complet: Aquesta pipeline hauria d'encarregar-se del desplegament de les dues capes, tant front com back. Primer s'hauria de construir els artefactes per a les capes front i back. A continuació, haurà de executar els tests automàtics i finalment hauria de fer el desplegament del bloc funcional complet als diferents entorns.
- Pipelines per desplegar versions concretes: Aquesta pipeline hauria de permetre el rollback a una versió concreta que ja s'havia desplegat anteriorment. Atès que el bloc funcional es versiona com a unitat, aquesta pipeline permetria fer rollback tant de codi com de configuració.
- Pipelines per realitzar rollbacks: Aquesta pipeline hauria de permetre el rollback a la versió anterior.
- Si és possible, pipelines per aturar, arrancar i reiniciar els serveis del bloc funcional en els diferents entorns

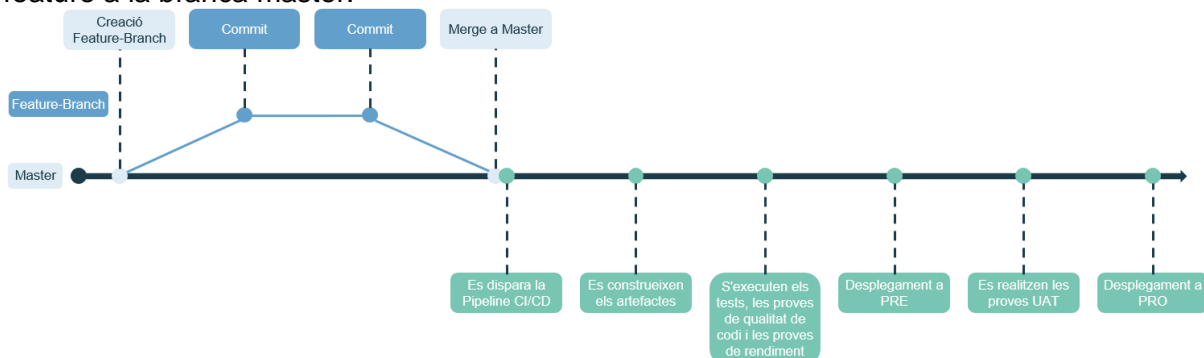
#### 2.3.2.1.6 Metodologia de branching

Per a cada repositori GIT es recomana trunk-based development, adaptada a les necessitats de CTTI. El projecte disposarà únicament de la branca master i de branques feature on es realitzarà el desenvolupament de les diferents funcionalitats:

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 18 / 58

- **Master:** És la branca principal de projecte, on va el codi per generar l'aplicació en entorns de producció.
- **Feature:** Branques generades a partir de la branca master. Sobre elles es faran els desenvolupaments de noves funcionalitats (o correccions de codi) i els canvis es tornen a passar a la branca master un cop finalitzats.

La pipeline de desplegament treballa exclusivament sobre la branca master i hauria de ser capaç de realitzar successivament els desplegaments als diferents entorns del CTTI. En el cas de disposar dels entorns d'Integració, Preproducció i Producció, la pipeline hauria de desplegar el bloc funcional primer a Integració, un cop superades satisfactòriament les fases de compilació i dels tests automàtics. Després d'haver validat el correcte funcionament de l'entorn d'Integració, es realitzaria el desplegament a Preproducció. A continuació, es realitzaran les proves UAT i un cop superades totes les validacions, el bloc funcional finalment es desplegaria a Producció. En la II-lustració 10 s'esquemmatitza el funcionament de la pipeline en el cas de disposar només dels entorns de Preproducció i Producció. També es pot veure que l'execució de la pipeline es dispara després d'haver realitzat un merge d'alguna branca feature a la branca master.



II-lustració 10 Branching


- **Master:** És la branca principal de projecte, on va el codi per generar l'aplicació en entorns de producció.
- **Feature:** Branques generades a partir de la branca master. Sobre elles es faran els desenvolupaments de noves funcionalitats (o correccions de codi) i els canvis es tornen a passar a la branca master un cop finalitzats.

### 2.3.2.1.7 Model de desplegament

A nivell de desplegament la capa back del bloc funcional es desplega en un contenidor dedicat i la capa front es desplega en un directori d'un contenidor front global, que serveix les capes front de tots els blocs funcionals de l'aplicació.

Avantatges:

- Ús més eficient dels recursos sense comprometre la independència dels equips.
- Es redueix la complexitat de les capes front

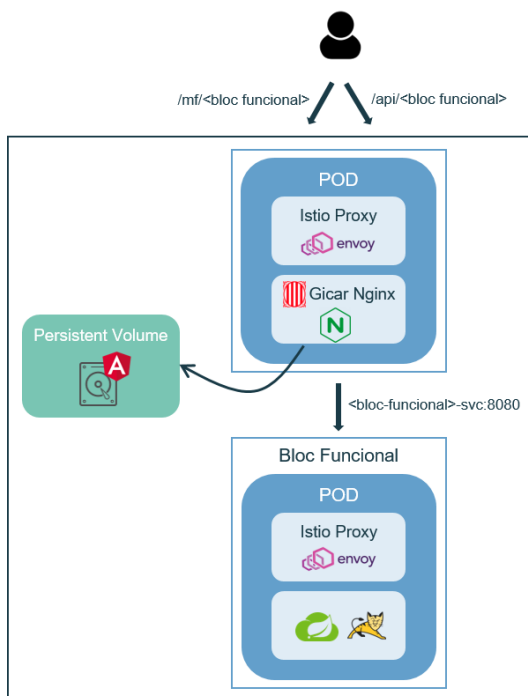
 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 19 / 58

- El rendiment de les capes front millora, pel fet que tot es serveix des del mateix servidor HTTP
- El nombre de components desplegats és més reduït

A nivell de model de desplegament és important l'aïllament de recursos entre blocs funcionals

A la següent imatge es mostra el model de desplegament d'un bloc funcional amb:

- Un microfrontend en Angular que es desplega en un Volum Persistent muntat al contenidor Nginx que serveix el frontal
- Un microservei en Java amb Spring Boot que es desplega en un POD independent




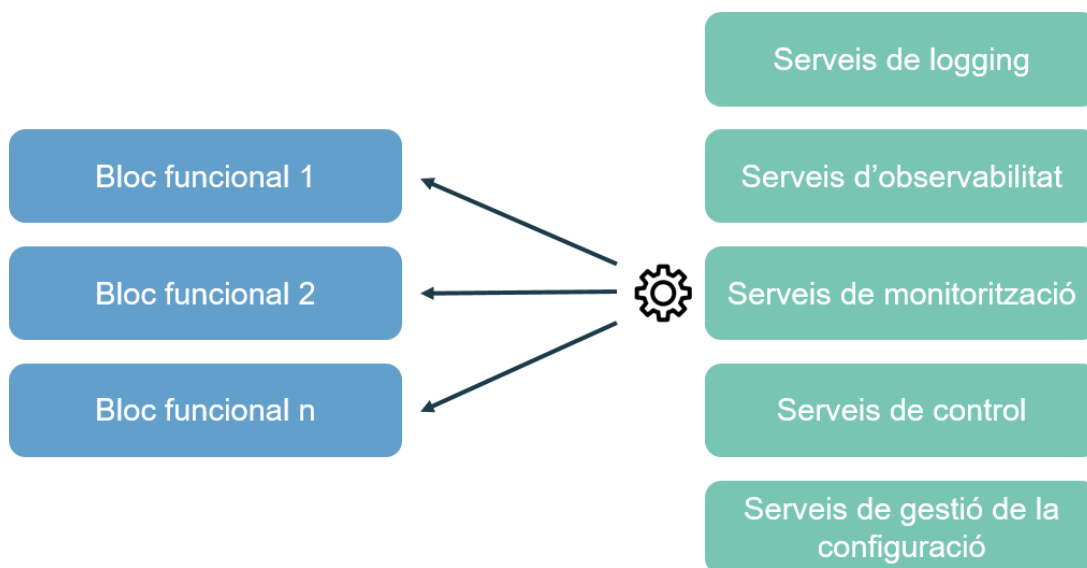
II-lustració 11 Model de desplegament

### 2.3.2.2 Model d'operacions

#### 2.3.2.2.1 Operacions

A continuació, es mostra la relació entre el concepte de bloc funcional amb les eines operacionals que requereix una arquitectura de microserveis.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 20 / 58




II-lustració 12 Model operacions

- **Serveis de monitorització:**

- Monitoritzen les mètriques i recursos consumits dels blocs funcionals
- Per a cada bloc funcional s'hauria de crear un dashboard independent que mostra les mètriques dels components dels que està compost el bloc amb detall. A més, caldria disposar d'un dashboard global d'overview que recull les mètriques bàsiques de sistema de tots els components de l'aplicació.

- **Serveis de logging:** Extreuen els logs dels diferents blocs funcionals i per a cada bloc els visualitzen en dashboards independents per a la seva explotació. En aquest dashboard en principi només es mostrarien els logs de la part back. Per poder veure logs de la part front, es podria desplegar una servei que rep els logs dels navegadors i els escriu en Kibana.
- **Serveis d'observabilitat:** Permeten traçar una transacció entre diversos blocs funcionals utilitzant i visualitzen aquesta informació en un dashboard global. Visualitzen les crides realitzades entre els diferents blocs funcionals en temps real
- **Serveis de gestió de la configuració:** Les plantilles de configuració dels components dels que es compon un bloc funcional estan emmagatzemades en els repositoris Git corresponents en carpetes de configuració. Durant la fase de desplegament, les pipelines s'encarregaran que s'apliquin les configuracions adequades a cada components en funció de l'entorn.
- **Serveis de control:** La majoria dels serveis de control no s'implementen a nivell dels blocs funcionals, sinó serà la pròpia infraestructura que supervisarà el correcte


 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 21 / 58

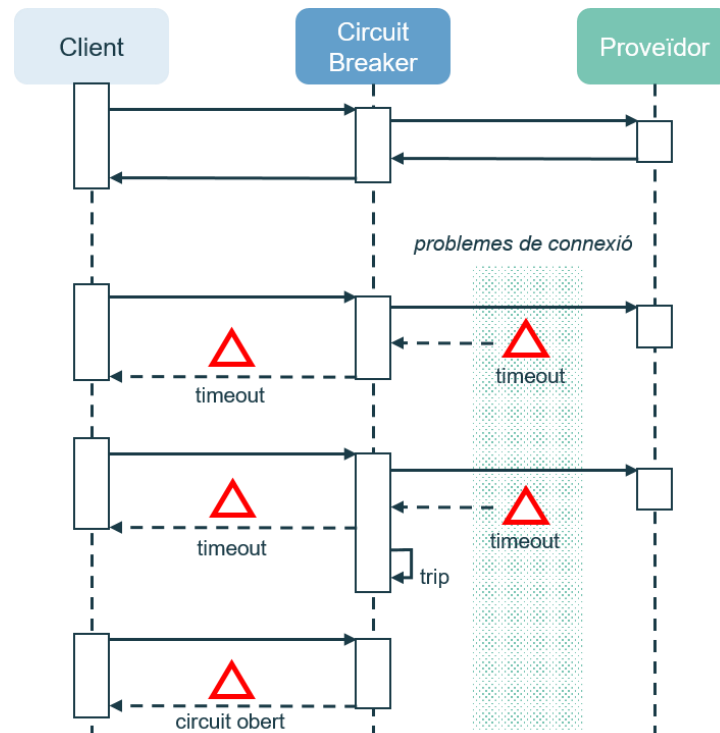
funcionament de tots els serveis de l'aplicació. Els components dels que està compost un bloc funcional només haurien de proporcionar endpoints per als healthiness- and readiness-checks que indicant el seu estat. En el següent apartat es detallen els conceptes tant dels healthiness- i readiness-checks com de Circuit Breaker, relacionats amb els serveis de control.

#### 2.3.2.2.2 Disponibilitat i resiliència

- **Healthiness and Readiness checks:** Per detectar i solucionar ràpidament problemes en blocs funcionals es recomana que els serveis de cada bloc funcional implementin Healthiness i Readiness Checks.
  - **Healthiness Checks:** Indiquen si el servei està aixecat.
  - **Readiness Checks:** Indiquen si el servei està funcionant correctament, tenint en compte totes les dependències d'aplicacions i serveis amb els quals s'ha d'integrar, como por exemple base de dades o serveis de cues de missatgeria.
- **Resiliència dels serveis (Circuit Breaker):** En una arquitectura de microserveis és molt comú tenir una gran quantitat de crides remotes entre els diferents serveis. Cal protegir el sistema davant de problemes en les crides remotes, perquè el servei pugui respondre ràpidament (fail fast) i per evitar cascades de crides errònies. Aquest fet implica la necessitat d'implementar el patró Circuit Breaker. Al mercat existeixen diferents solucions per implementar aquest patró, com per exemple: Hystrix de Netflix o en el propi Istio. Si l'aplicació es desplega en una eina d'orquestració de contenidors, per a cada bloc funcional es recomana implementar les funcionalitats de Circuit Breaker en el propi Service Mesh. En cas contrari, cada bloc funcional hauria d'implementar el patró de Circuit Breaker, per exemple utilitzant alguna llibreria de tercers, com per exemple Hystrix de Netflix.

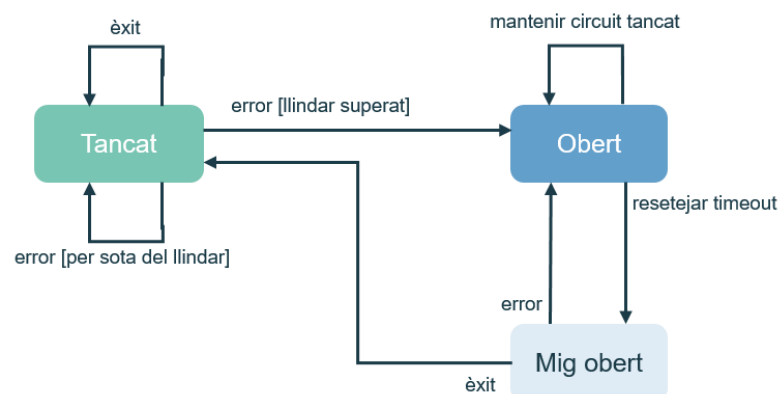
En la Il·lustració 13 es detalla el funcionament conceptual d'un Circuit Breaker. Mentre el servei del proveïdor no pateix problemes, el Circuit Breaker passa la crida a el servei destí. No obstant això, quan detecta un problema en el servei del proveïdor, el Circuit Breaker respon a el servei que va realitzar la crida ràpidament amb un missatge d'error (timeout), sense passar la crida a el servei destí.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 22 / 58




II-lustració 13 Comportament esquemàtic d'un Circuit Breaker

En la II-lustració 14 es mostren els estats pels quals passa un Circuit Breaker. Quan no hi ha problemes de connectivitat a el servei d'un proveïdor, el Circuit Breaker està tancat. Quant detecta algun problema, el component passa a l'estat obert per evitar que les crides es propaguin fins al sistema de proveïdor. A més, el Circuit Breaker és capaç de detectar quan el servei del proveïdor torni a estar disponible, mitjançant reintents de connexió. Per a això es posa en estat mig obert. Si es confirma que el servei del proveïdor torna a estar disponible, es posa en estat tancat, sinó es manté obert.



II-lustració 14 Diagrama d'estat de un Circuit Breaker

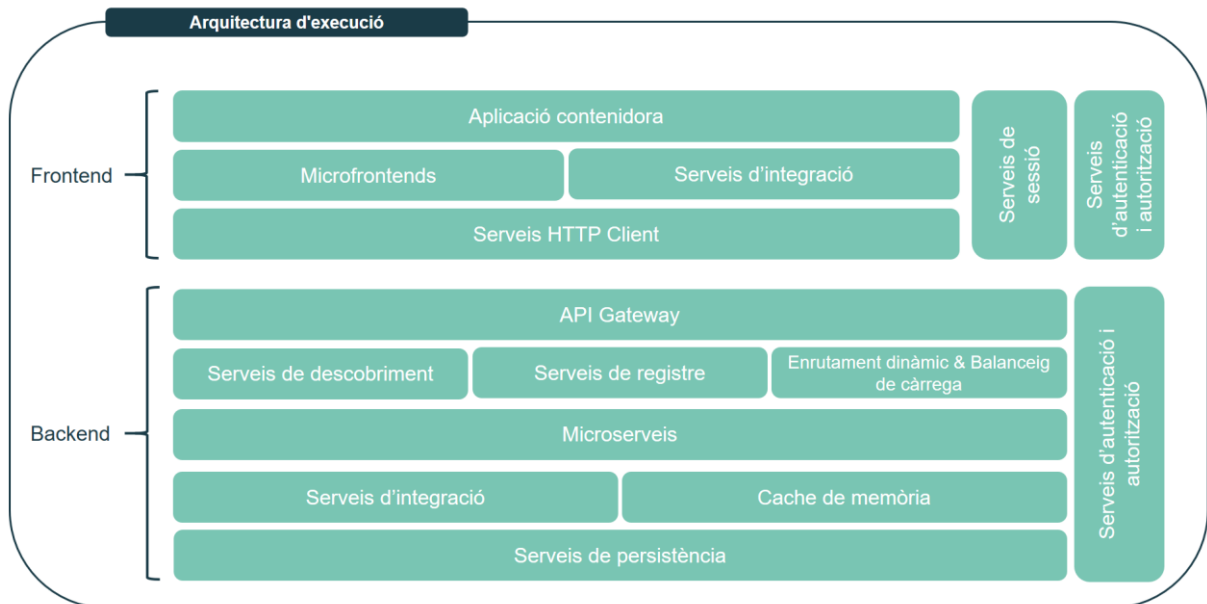
 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 23 / 58

## 2.4 Arquitectura tècnica

### 2.4.1 Disseny conceptual


El disseny conceptual identifica i descriu els seus components des d'un punt de vista neutre tecnològicament i s'ha dividit en tres blocs:

- **Arquitectura d'execució:** Inclou els components necessaris per tal de poder executar els microserveis i microfrontends de la solució i que aquests interactuïn entre ells. El fet que l'arquitectura sigui altament distribuïda requereix l'existència de certs components que no són necessaris en arquitectures monolítiques. Aquests components són els que donen resposta als requeriments dels usuaris de negoci.



Il·lustració 15 Disseny conceptual. Arquitectura d'execució

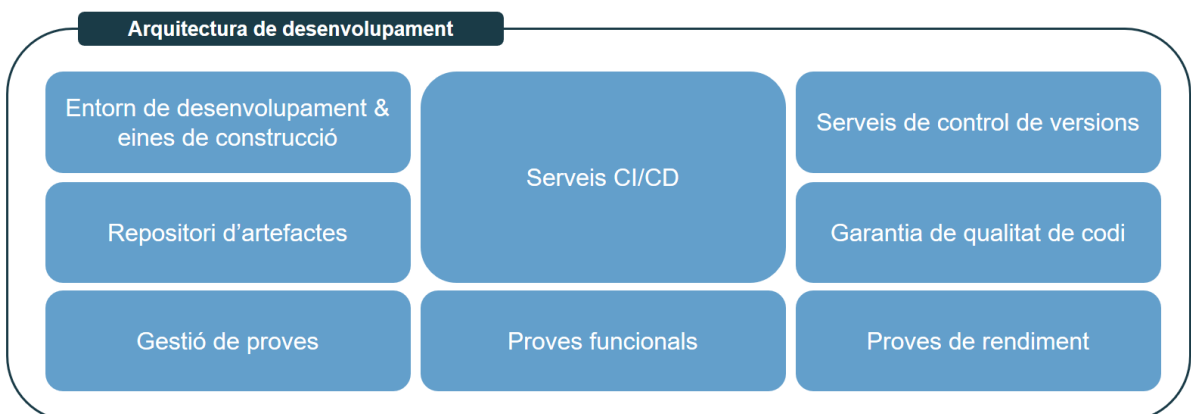
- **Arquitectura d'operació:** Els components d'aquest bloc permeten que, un cop desplegada i en execució la solució, aquesta pugui ser supervisada i es pugui assegurar el seu nivell de servei. En arquitectures altament distribuïdes no és suficient una visió independent de cada artefacte de la solució sinó que calen visions agregades de monitorització, observabilitat, etc. Aquests components donen resposta als perfils d'operació de la solució.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 24 / 58



II-lustració 16 Disseny conceptual. Arquitectura d'operació

- **Arquitectura de desenvolupament:** Inclou els components per poder desenvolupar i desplegar els microfrontends i microserveis que componen la solució. Per treure màxim profit de l'arquitectura és necessari que els diferents artefactes de la solució tinguin un cicle de vida el més autònom possible. Aquests components donen servei als equips de desenvolupament de la solució.




II-lustració 17 Disseny conceptual. Arquitectura de desenvolupament

#### 2.4.1.1 Arquitectura d'execució

La descripció dels blocs conceptuals de l'arquitectura d'execució de front-end són les següents:




 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 25 / 58

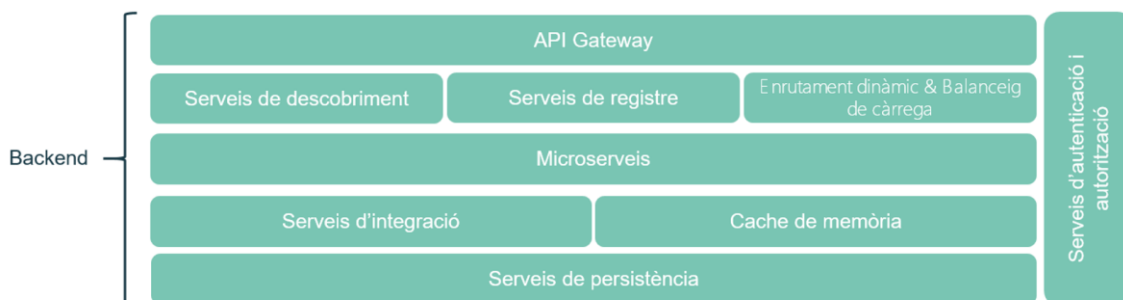


II-l·lustració 18 Disseny conceptual. Arquitectura d'execució - Frontend

- **Aplicació contenidora:** És l'aplicació que té, com a responsabilitat principal, la composició dels diferents microfrontends creant un frontal únic i que farà de punt d'entrada de l'usuari de negoci. No hauria de suposar una nova aplicació dins del sistema, sinó que es pot triar un dels blocs funcionals (normalment el que tingui una funcionalitat més core) perquè assumeixi la responsabilitat d'aplicació contenidora.
- **Microfrontends:** Components que descomponen l'aplicació en elements d'interfície d'usuari individuals i independents. Tenen un baix grau d'acoblament entre ells i poden tenir el seu propi cicle de vida de desplegament. No es mostren a l'usuari per si mateixos, per a això, s'incrusten en l'aplicació contenidora que és la que compon les pantalles fent ús dels diferents microfrontends.
- **Serveis HTTP Client:** Serveis que controlen l'accés a les funcionalitats de backend i que proporcionen un punt d'accés comú a les mateixes. Homogeneïtzen l'accés al backend i poden afegir elements comuns a les peticions així com un tractament comú als errors de comunicació.
- **Serveis de sessió:** Serveis encarregats de mantenir i gestionar la informació d'interès relativa a la navegació de l'usuari que està usant l'aplicació.
- **Serveis d'integració:** Serveis que proporcionen mecanismes per implementar la integració entre els diferents microfrontends i l'aplicació contenidora. La integració entre microfrontends, en cas necessari, no és directa i usen aquests serveis.
- **Serveis d'autenticació i autorització:** Serveis que proporcionen les funcionalitats d'integració amb la gestió d'identitats i d'obtenció del token per accedir als serveis de backend. Addicionalment incorporen les funcionalitats d'autorització controlant qui pot fer què.


i la dels serveis necessaris de backend:

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 26 / 58



Il·lustració 19 Disseny conceptual. Arquitectura d'execució - Backend

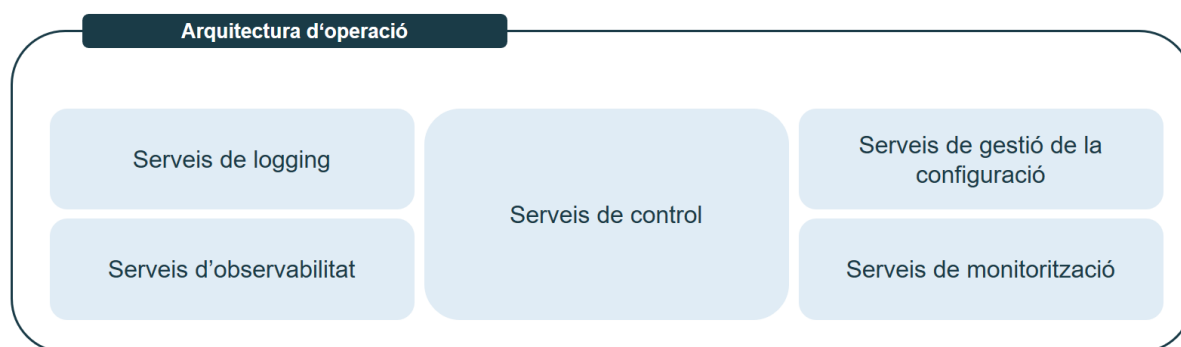
- **API Gateway:** Proporciona un punt d'entrada únic a tots els microserveis de la solució aplicant les regles de redirecció internes i garantint-ne el control d'accés. És un component de redirecció lleuger orientat, principalment, a la publicació de les API's pel front-end de la solució.
- **Serveis de descobriment:** Els serveis de descobriment implementen la lògica que interroga els serveis de registre per localitzar, en el moment de consum, la ubicació física del servei a invocar.
- **Serveis de registre:** Proporciona funcionalitats per tal que els serveis, en el moment de posar-se en marxa, hi publiquin la seva ubicació física associant-la a un nom i, d'aquesta forma, fer-se disponible pels serveis de descobriment.
- **Serveis d'integració:** Proporcionen mecanismes per implementar diferents estils d'integració entre els micro-serveis de la solució, per exemple, cues de missatgeria o events. D'igual forma proporcionen aquests serveis per integracions amb d'altres sistemes externs incloent, per determinades casuístiques, publicacions externes d'API's.
- **Cache de memòria:** Emmagatzematge de dades d'alt rendiment orientat a eficientar l'accés als serveis de persistència. Addicionalment també pot emmagatzemar dades amb baixos requeriments de durabilitat com, per exemple, dades de sessió.
- **Microserveis:** Són els serveis que implementen la lògica pròpia de la solució i que són el resultat de la descomposició funcional de la mateixa. De forma general implementen també la lògica vinculada a d'altres requeriments no funcionals com, per exemple, monitorització, operació, etc.
- **Serveis de persistència:** Serveis de persistència de les dades d'alta durabilitat que proporcionen la possibilitat d'independitzar el magatzem de dades de cadascun dels microserveis.
- **Serveis d'autenticació i autorització:** Serveis que proporcionen el token necessari per accedir als serveis via l'API Gateway així com els rols associats a l'usuari autenticat.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 27 / 58

- **Enrutament dinàmic i balanceig de càrrega:** Els components de balanceig de càrrega usen els serveis de descobriment per cercar on existeixen instàncies del servei sol·licitat i, d'aquesta forma, encaminar les peticions cap a una instància concreta. La funcionalitat garanteix detecció d'indisponibilitats per deixar d'enviar peticions. L'enrutament dinàmic permet decisions d'encaminament de peticions basat en regles de negoci.


#### 2.4.1.2 Arquitectura d'operació

La descripció dels blocs conceptuals de l'arquitectura d'operació són les següents:



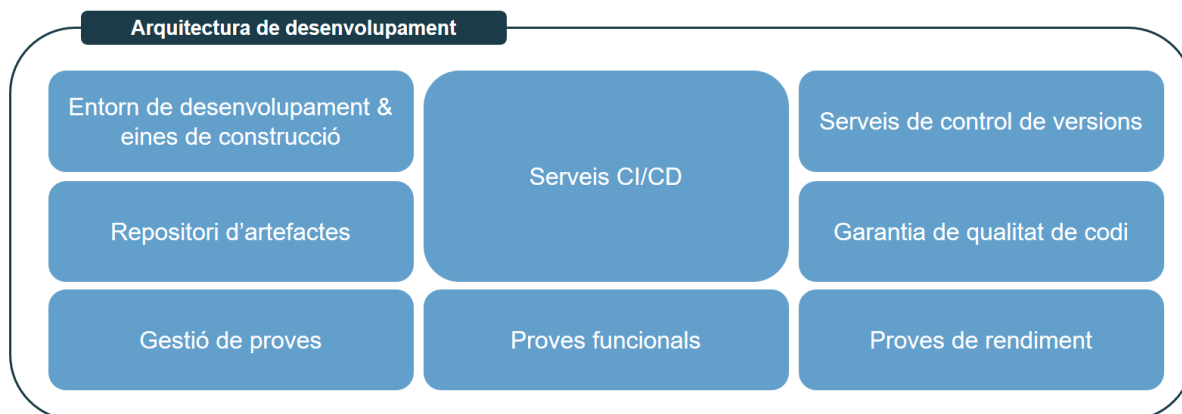
Il·lustració 20 Disseny conceptual. Arquitectura d'operació

- **Serveis de control:** Serveis que permeten realitzar accions de parada i arrancada dels serveis, auto-escalat, circuit breaker, healthiness- i readiness-checks, etc.
- **Serveis d'observabilitat:** Són els serveis que recopilen informació per donar una visió del comportament intern del sistema com, per exemple, per fer el seguiment de la traçabilitat de les peticions entre microserveis, etc.
- **Serveis de gestió de la configuració:** Proporciona capacitats de configuració a tots els microserveis de manera que no sigui necessària la distribució de fitxers de propietats. Els microserveis poden consumir la configuració al iniciar-se i actualitzen la configuració sense necessitat de reinicis.
- **Serveis de logging:** Serveis de centralització de la informació de logging generat pels diferents microserveis de forma que sigui explotable de forma global a nivell de solució i no de microservei concret.
- **Serveis de monitorització:** Serveis que permeten veure en temps real les mètriques que caracteritzen externament el comportament del sistema, per exemple, consums de memòria, CPU, temps de resposta, etc.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 28 / 58

### 2.4.1.3 Arquitectura de desenvolupament


La descripció dels blocs conceptuals de l'arquitectura de desenvolupament són les següents:



II-lustració 21 Disseny conceptual. Arquitectura de desenvolupament

- **Entorn de desenvolupament i eines de construcció:** És un programa o conjunt de programes que engloben totes les tasques necessàries per al desenvolupament d'un programa o aplicació. A més també inclou les eines de construcció i gestió de les dependències.
- **Repositori d'artefactes:** Emmagatzema i proporciona accés a paquets compilats de software amb suport multi-llenguatge.
- **Servei de control de versions:** Emmagatzema el codi suportant el versionat del mateix i proporcionat les funcionalitats de branching, merge, etc.
- **Proves de rendiment:** Eines d'automatització de les proves de rendiment per garantir que els temps de resposta són adequats a diferents nivells de càrrega.
- **Gestió de casos de prova:** Eines de gestió de proves per emmagatzemar informació sobre com s'han de fer les proves, planificacions de les activitats de prova i informar-ne sobre l'estat i resultats.
- **Prova funcional:** Eines d'automatització de les proves funcionals
- **Garantia de qualitat de codi:** Eina d'inspecció de la qualitat del codi multi llenguatge que ofereix informes reportant aspectes com indicadors de la complexitat del codi, duplicitats, vulnerabilitats de seguretat, etc.
- **Serveis CI/CD:** Serveis que implementen el procés d'automatització de la construcció i desplegament en els diferents entorns incorporant cicles de proves cada vegada que un membre de l'equip de desenvolupament fa canvis al servei de control de versions.

De forma general, en aquest apartat s'entén el codi des d'un punt de vista ampli incloent les definicions d'infraestructura

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 29 / 58

## 2.4.2 Disseny lògic

El disseny lògic és una realització concreta del disseny conceptual especificant les tecnologies que poden resoldre cada bloc. com s'ha comentat anteriorment poden haver diferents realitzacions però, en aquest cas, n'hem especificat dues a mode d'exemple:

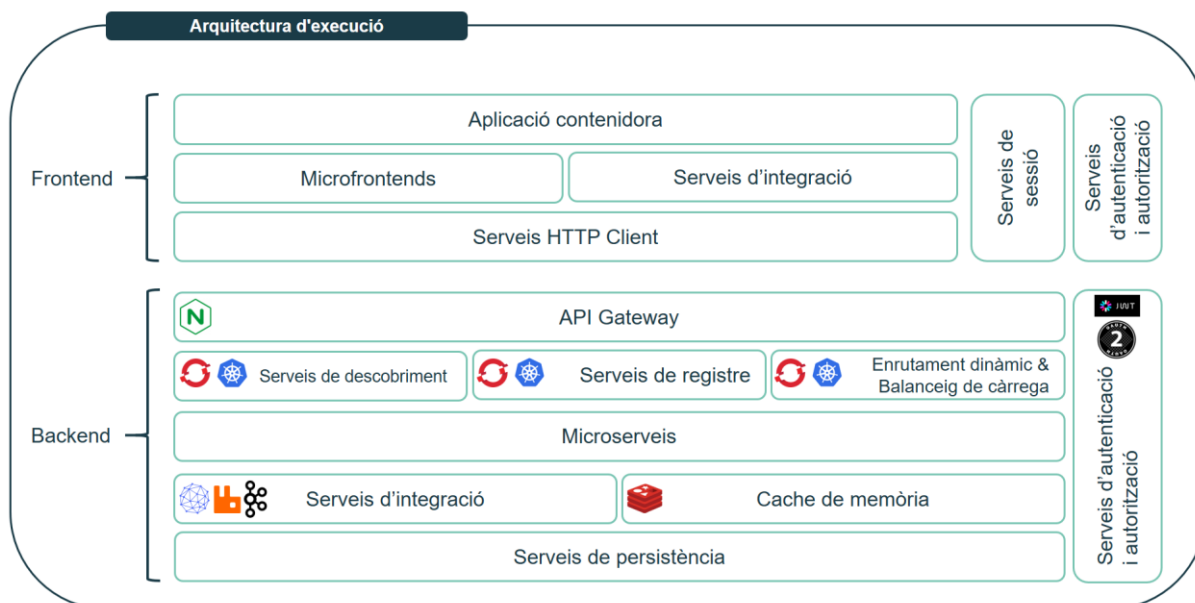
- Disseny lògic neutre respecte a proveïdor de cloud
- Disseny lògic basat en el cloud públic Azure

En els dos dissenys lògics realitzats s'ha tingut en compte un escenari en el qual els microserveis corren en un únic CPD i sobre una única plataforma d'orquestració Kubernetes o OpenShift.

### 2.4.2.1 Disseny lògic neutre respecte a proveïdor cloud


#### 2.4.2.1.1 Arquitectura d'execució

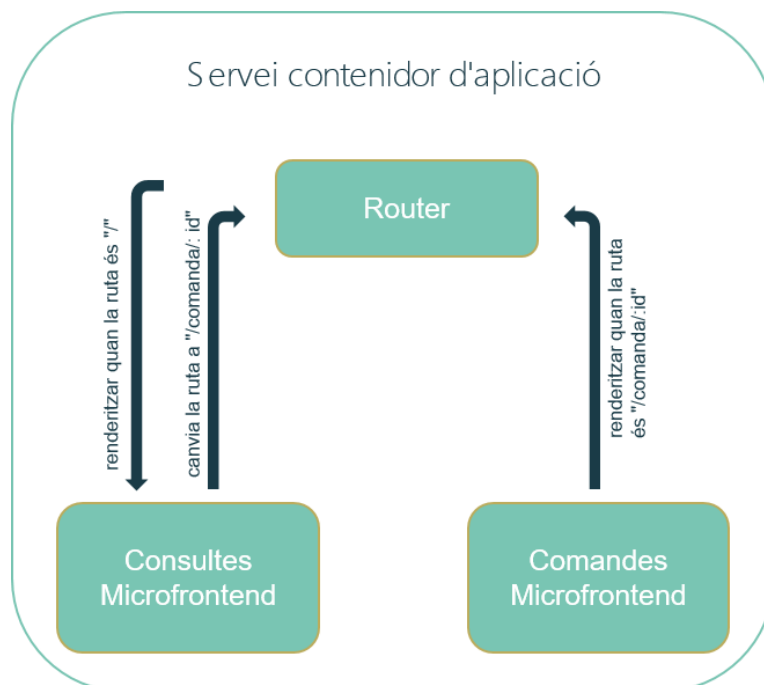
La descripció dels blocs lògics de l'arquitectura d'execució de front-end són les següents:



Il·lustració 22 Disseny lògic. Arquitectura d'execució

- **Aplicació contenidora:** És l'aplicació que compon les distintes vistes que formen el frontal. Això s'aconsegueix incrustant els diferents microfrontends, a la aplicació contenidora, mitjançant l'ús dels Custom Elements. Els Custom Elements incrustats encapsulen els diferents microfrontends. Al carregar les pàgines, l'aplicació contenidora, permet, al navegador, aixecar una nova instància de cada microfrontend incrustat. D'aquesta manera es renderitza el HTML de cada un i es disponibilitza la seva funcionalitat.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 30 / 58




II-lustració 23 Servei contenidor d'aplicació

- **Microfrontends:** Cada bloc funcional ha d'implementar el seu propi microfrontend independent, de manera que s'asseguri la independència entre els diferents blocs funcionals. Els microfrontends poden ser desenvolupats en qualsevol llenguatge o framework de front-end acceptat per CTTI però és recomana mantenir l'homogeneïtat tecnològica dins de l'aplicació. A continuació s'indiquen alguns dels beneficis que proporciona l'ús de microfrontends:
  - Actualitzacions incrementals
  - Codi més simple i desacoblat
  - Desplegament independent
  - Equips autònoms

A dia d'avui hi ha diferents estils d'Integració para los microfrontends: Iframe, Javascript, Web Components, Build-time Integration. De tots a els estils anteriors és recomana fer la Integració mitjançant Web Components de fent us de Custom Elements. La integració via web Components és una variació de la Integració via Javascript. En aquest cas cada microfrontend defineix un element HTML personalitzat perquè el contenidor creï una instància (Custom Elements). Cada microfrontend s'inclou a la pàgina mitjançant una etiqueta `<script>` i, al carregar-se, exposa una funció global com a punt d'entrada. L'aplicació contenidora després determina què microfrontend ha de muntar i crida a la funció rellevant per dir-li a cada microfrontend quan i on renderitzar-se.

A diferència de la integració en temps de compilació, amb aquest estil, podem canviar qualsevol microfrontend de forma independent sense necessitat de compilació ni de desplegaments. I a diferència dels iframes, tenim total flexibilitat per crear integracions

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 31 / 58

entre les nostres microfrontends com vulguem. A més, aquest estil d'integració permet l'ús del patró "lazy loading", el que optimitza els temps de descàrrega i millora l'experiència d'usuari.


El resultat final aquí és bastant similar al resultat de la Integració Javascript, la principal diferència és que es poden utilitzar les capacitats que ofereix el navegador.

Pel que fa a l'accés a les APIs, cada microfrontend ha d'atacar únicament a l'API de la seva bloc funcional.

Principals característiques que han de tenir els microfrontends:

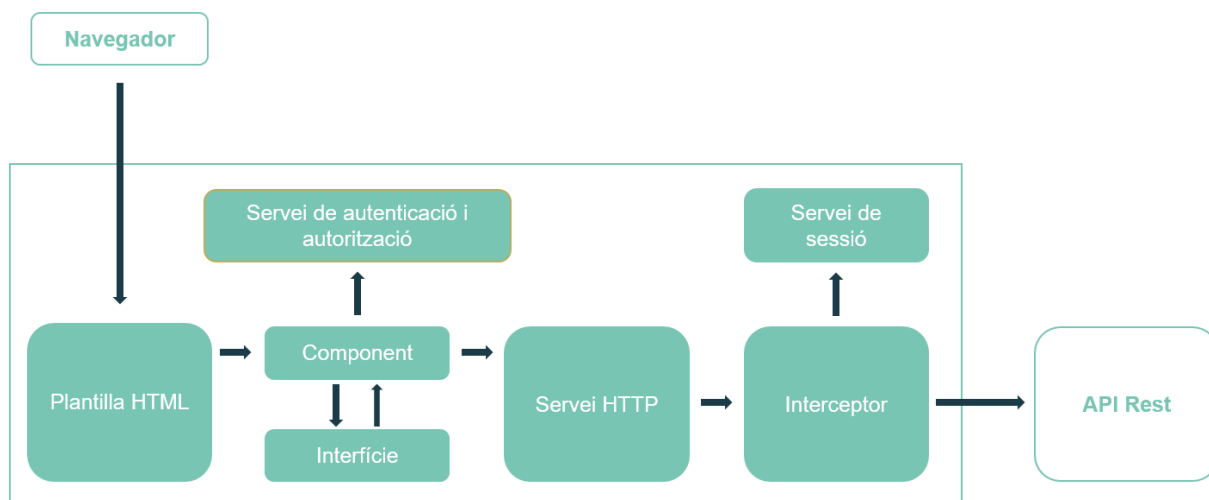
- **Agnòstics a la tecnologia.** S'ha de assegurar que es pot actualitzar el stack o fins i tot la versió del framework usat per al desenvolupament sense que sigui necessari coordinar el canvi en tots els microfrontends. A dia d'avui es recomana l'ús de Custom Elements ja que són una bona forma d'encapsular la implementació mentre que proporciona una interfície neutra.
- **Assegurar l'aïllament.** S'han de crear aplicacions independents i Autònomes. No s'ha de compartir runtime, fins i tot si tots fan servir el mateix framework. És recomanable evitar l'ús d'estats compartits o variables globals.
- **Establir nomenclatures.** És recomanable establir nomenclatura o prefixos que ajuden a identificar de forma clara i ràpida a l'equip owner de cada microfrontend.
- **Funcions natives de navegador.** Fer ús, de forma preferent, de funcions natives del navegador en lloc d'APIs personalitzades.
- **Resiliència.** La seva funció ha de ser útil, fins i tot si Javascript falla o no s'ha executat encara.
- **Serveis HTTP Client:** Els serveis HTTP Client centralitzen els accessos als serveis backend. Totes les crides que es realitzen des de la capa frontend als serveis backend es canalitzen a través d'aquest servei. D'aquesta manera, es pot utilitzar un servei HTTP Client per afegir de forma centralitzada propietats comunes a les crides a la capa backend, com ara la injecció transparent d'un token JWT per a l'autenticació correcta en els serveis backend. A més, es pot utilitzar un servei de HTTP Client per a la gestió centralitzada d'errors en les crides als serveis backend. Els serveis HTTP s'encarreguen d'interceptar tot els errors per mostrar a l'usuari un missatge d'error adequat. Els serveis HTTP se solen implementar a mida per als diferents projectes segons les necessitats.
- **Serveis de sessió:** Els serveis de sessió s'encarreguen de mantenir i gestionar la informació d'interès relativa a l'usuari que està utilitzant l'aplicació. Aquests serveis solen estar integrats amb els navegadors Web de cada usuari i solen guardar tant el token JWT de l'usuari com les seves propietats més rellevants, com el seu nom o els seus rols. Per això, els navegadors Web tenen incorporats un Session Storage, que es pot utilitzar per guardar dades temporals en el propi navegador de cada usuari. A



 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 32 / 58

més, també es poden utilitzar Cookies per guardar i gestionar informació relativa a la sessió de l'usuari en el seu navegador Web.


- Serveis d'autenticació i autorització:** Els serveis d'autenticació proporcionen les funcionalitats d'integració amb la gestió d'identitats i d'obtenció del token per accedir als serveis de backend. Quan l'usuari encara no està autenticat, els serveis d'autenticació s'encarreguen de mostrar el formulari de login i gestionar el procés d'autenticació amb la capa back perquè aquesta generi un token JWT. En una aplicació per al CTTI l'autenticació d'usuari se sol delegar a GICAR, qui mostra els formularis de login i valida les credencials introduïts. En aquest sentit, durant el login d'usuari dels serveis d'autenticació només s'encarreguen d'interceptar la capçalera HTTP generada per GICAR i passar-la al servei backend d'autenticació perquè es generi un token JWT. En relació al logout, els serveis d'autenticació proporcionen a l'usuari una URL perquè es pugui fer la desconnexió en GICAR. Els serveis d'autorització s'encarreguen de renderitzar les pantalles en funció dels permisos d'usuari. Obtenen els rols d'usuari dels serveis de sessió i s'encarreguen de mostrar a l'usuari adequadament el contingut de les pantalles, habilitant o des habilitant certs components HTML, com botons, camps de text, etc.

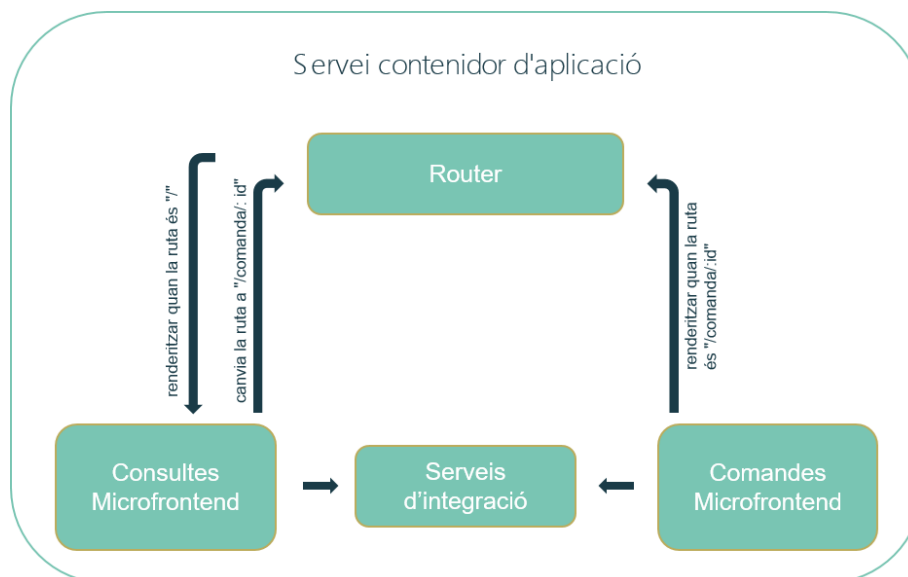


II-Il·lustració 24 Diagrama Frontend

- Serveis d'integració:** Els serveis d'integració centralitzen la comunicació entre els diferents components de la capa frontend. Per a això, l'aplicació contenidora es subscriu als esdeveniments emesos pels diferents microfrontends incrustats i els intercepta per propagar-les als microfrontends destí perquè aquests duguin a terme les accions corresponents, com ara un refresc de la interfície gràfica. De la mateixa manera, cada microfrontend hauria de disposar d'un servei que centralitza l'enviament de les seves diferents esdeveniments a l'aplicació contenidora. Amb aquest patró s'assegura una solució neta per a la comunicació entre els diferents serveis dels quals es compon la capa frontend i s'evita que els esdeveniments estiguin repartits per diversos components. Cada microfrontend disposarà d'un servei centralitzat que propaga tots els esdeveniments a l'aplicació contenidora i l'aplicació contenidora és l'únic component subscript als esdeveniments generats per tots els microfrontends. Els serveis d'integració s'haurien d'implementar a mida per a cada projecte




 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 33 / 58



II·lustració 25 Servei contenidor d'aplicació amb serveis d'integració

i la de backend:

- **API Gateway:** L'API Gateway és el punt d'accés centralitzat a l'aplicació. Actua com a proxy per redirigir les crides als components backend corresponents i es pot utilitzar per servir el contingut estàtic de l'aplicació. A més, aquest servei es pot encarregar de tasques comunes i transversals a tots els serveis. Per exemple, si l'aplicació està integrada amb GICAR o amb Vàlid, aquest component s'encarregaria de la integració amb aquests sistemes utilitzant un agent Shibboleth. Les funcionalitats de l'API Gateway es poden implementar amb un servei NGINX, utilitzant les imatges proporcionades pel CTTI que ja estan integrades amb Shibboleth.
- **Serveis de descobriment, serveis de registre, enrutament dinàmic i balanceig de càrrega:**
  - **Openshift:** OpenShift és una plataforma de gestió de contenidors basada en Kubernetes per al desenvolupament i la implantació d'aplicacions empresarials.
  - **Kubernetes:** Kubernetes és un sistema de codi lliure per a l'automatització del desplegament, ajust d'escala i maneig d'aplicacions en contenidors. Kubernetes facilita l'automatització i la configuració declarativa. Té un ecosistema gran i en ràpid creixement. El suport, les eines i els serveis per Kubernetes estan àmpliament disponibles. Ofereix un entorn d'administració centrat en contenidors. Orquestra la infraestructura de còmput, xarxes i emmagatzematge perquè les càrregues de treball dels usuaris no hagin de fer-ho. Això ofereix la simplicitat de les Plataformes com a Servei (PaaS) amb la flexibilitat de la Infraestructura com a Servei (IaaS) i permet la portabilitat entre proveïdors d'infraestructura.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 34 / 58

- **Serveis d'integració:**

- **Integracions síncrones (HTTP):** Per aquest tipus d'integracions és recomanable fer us de les solucions d'API Manager corporatives.

- **Integracions asíncrones (publish-subscribe):**

- **Apache Kafka:** Apache Kafka és un sistema de transmissió de dades distribuït amb capacitat d'escalat horitzontal i tolerant a fallades. Gràcies al seu alt rendiment ens permet transmetre dades en temps real utilitzant el patró de missatgeria publish / subscribe.
- **RabbitMQ:** RabbitMQ és un Sistema de cues de missatges de codi obert que ens permet executar codi de les nostres aplicacions asíncronament.


- **Cache de memòria:** Redis millora el rendiment i l'escalabilitat d'una aplicació que faci servir en gran mesura els magatzems de dades de back-end. És capaç de processar grans volums de sol·licituds d'aplicació al mantenir en la memòria del servidor les dades als quals s'accedeix amb freqüència. En aquesta memòria es pot escriure i llegir ràpidament. Redis incorpora una solució crítica d'emmagatzematge de dades de baixa latència i alt rendiment en les aplicacions modernes.

- **Microserveis:** Els microserveis es poden desplegar / executar de dues maneres diferents:


- **Conteneritzats sobre una plataforma d'orquestració (Openshift/Kubernetes):**

- **Serverless:** Serverless és un model d'execució en el que el proveïdor en el núvol (AWS, Azure o Google Cloud) és responsable d'executar un fragment de codi mitjançant l'assignació dinàmica dels recursos i cobrant només per la quantitat de recursos utilitzats per executar el codi. El codi, generalment, s'executa dins de contenidors sense estat que poden ser activats per una varietat d'esdeveniments que inclouen sol·licituds HTTP, esdeveniments de base de dades, serveis de coles, alertes de monitoratge, càrrega d'arxius, esdeveniments programats (treballs cron), etc. El codi que s'envia al proveïdor en el núvol per a l'execució és generalment en forma d'una funció. Per tant, serverless a vegades es denomina "Funcions com a servei" o "FaaS". En el món serverless normalment es requereix que adopti una arquitectura basada en microserveis. No es recomana executar tota l'aplicació dins d'una sola funció com un monòlit.

- **Cloud públic:** La majoria de proveïdors de cloud públic ofereixen aquest servei des de fa ja alguns anys. A continuació s'indiquen els més usuals:
  - Azure Functions
  - AWS Lambda

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 35 / 58

- Cloud Functions
  - **Cloud privat:** En l'actualitat CTTI no disposa de contractes en vigor amb els seus proveïdors de cloud privada que ofereixin aquesta modalitat de desplegament / execució
- **Serveis de persistència:** Aquests serveis dependran fortament de la problemàtica concreta a resoldre per la Solució. Per a l'elecció del Servei de persistència que millor s'adapti en cada cas, s'han de tenir en compte, principalment, els requisits del nostre sistema pel que fa a: escalabilitat, latència, rendiment i consistència de dades. A més, cada tipus de base de dades té uns usos més concrets per als quals han estat dissenyades. A continuació s'indiquen els principals tipus de serveis de persistència disponibles a l'actualitat:
  - **Relacionals.** Les bases de dades relacionals emmagatzemen conjunts de dades com: taules amb files i columnes on tota la informació s'emmagatzema com un valor en una cel·la específica. Les dades en un RDBMS es gestionen utilitzant SQL. Encara que existeixen diferents implementacions, SQL està estandarditzat i proporciona un nivell de predictibilitat i utilitat. Aquest tipus de bases de dades poden ser usades per: Situacions on la integritat de les dades és absolutament primordial, dades altament estructurats, automatització de processos interns, entre d'altres.
  - **Documents.** Un magatzem de documents és una base de dades no relacional que emmagatzema dades en documents JSON, BSON o XML. Compten amb un esquema flexible. A diferència de les bases de dades SQL, on els usuaris han de declarar l'esquema d'una taula abans d'inserir dades, aquí els magatzems de documents no imposen l'estructura del document. Aquest tipus de bases de dades poden ser usades per: dades no estructurats o semiestructurats, gestió de contingut, anàlisi de dades, entre d'altres.
  - **Clau/valor.** Un magatzem de clau-valor és un tipus de base de dades no relacional on cada valor està associat amb una clau específica. També es coneix com una matriu associativa. La "clau" és un identificador únic associat només amb el valor. Aquest tipus de bases de dades poden ser usades per: emmagatzemar recomanacions, perfils d'usuari i configuracions, dades no estructurats com ressenyes o comentaris en un bloc, informació de sessió, dades a les quals s'accedeix amb freqüència però que no s'actualitzen amb freqüència.
  - **Columnes amples.** Són bases de dades dinàmiques no relacionals orientades a columnes. Aquest tipus de bases de dades poden ser usades per a: anàlisi de big data on la velocitat és important o emmagatzematge de dades en big data entre d'altres.
  - **Grafs.** Aquestes bases de dades fan servir grafs que permeten representar interaccions complexes entre dades. No tenen esquemes, el que permet una gran flexibilitat similar a les d'un document "valor-clau". També admet de manera similar les relacions que es generen en una base de dades relacional.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 36 / 58

- **Cerca.** Mes que un tipus de base de dades, aquest tipus de solucions són motors de cerca, però s'ha considerat rellevant incloure'ls a causa de la gran popularitat que estan guanyant en els darrers anys, sobretot Elasticsearch. Aquest tipus de solució pot ser usada per millorar l'experiència de l'usuari amb resultats de cerca més ràpids

**Serveis d'autenticació i autorització:** Amb aquesta finalitat, la CTTI proporciona GICAR. GICAR és un servei que facilita la gestió de les identitats (persones) que treballen o col·laboren amb la Generalitat de Catalunya (Funcionaris, interins, empreses públiques, externs, etc.), així com la gestió i el control de l'accés als recursos per part de les identitats -entenent com a recurs qualsevol eina, sistema d'informació o aplicació necessari per a que el professional desenvolupi les seves funcions. Com s'ha comentat anteriorment, GICAR proporciona Servei d'autorització, però si el sistema a desenvolupar requereix d'un Servei d'autorització amb una complexitat molt elevada o té requisits que GICAR no pugui complir, també es pot fer un desenvolupament a mida. A dia d'avui, la integració amb GICAR es realitza amb l'agent Shibboleth mitjançant el protocol SAML2. Per a això CTTI proporciona dues imatges Docker preparades per a aquesta integració: una basada en Apache (<https://git.intranet.gencat.cat/3048-intern/imatges-docker/gicar-shibboleth-openshift/tree/1.0.3>) i una altra basada en Nginx (<https://git.intranet.gencat.cat/3048-intern/imatges-docker/gicar-nginx-openshift/tree/1.0.0>).


#### 2.4.2.1.2 Arquitectura d'operació

La descripció dels blocs lògics de l'arquitectura d'operació són les següents:




II·lustració 26 Disseny lògic. Arquitectura d'operació

- **Serveis de control:** Les eines que s'utilitzen en aquesta capa són les següents:
  - **Kubernetes:** Kubernetes disposa d'un agent que observa les instàncies aixecades de cada servei, en cas de detectar problemes en alguna d'elles és capaç d'aturar la instància i aixecar una nova. També s'encarrega d'assegurar que sempre estan aixecades el nombre correcte d'instàncies de cada servei en funció dels paràmetres d'escalat i dels healthiness- i readiness-checks.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 37 / 58

- **Serveis d'observabilitat:** Les eines que s'utilitzen en aquesta capa són les següents:
  - **Jaeger:** Jaeger és un sistema distribuït de traces que es pot utilitzar per monitoritzar arquitectures basades en microserveis. Ofereix una interfície web per visualitzar la propagació distribuïda de context, el monitoratge de transaccions distribuïdes, l'anàlisi de la causa arrel i l'anàlisi de la dependència entre serveis. Jaeger està integrat a la plataforma Istio.
  - **Kiali:** Kiali és una aplicació que ofereix una consola de administració que proporciona panells de control, observabilitat i permet operar un service-mesh amb capacitats sòlides de configuració i validació. Visualitza l'estructura del service-mesh dels serveis a l'inferir la topologia de trànsit i mostra l'estat de la seva malla. Kiali proporciona mètriques detallades, validació potent, accés a Grafana i una sòlida integració per al seguiment distribuït amb Jaeger.
  - **Istio:** Istio compta amb un agent / proxy que controla tot el tràfic entrant i sortint de cada contenidor. D'aquesta manera Istio controla les comunicacions i extreu la informació de observabilitat que s'explota a través de Jaeger i Kiali.
- **Serveis de gestió de la configuració:**
  - **ConfigMaps:** La configuració dels microserveis de sistema es realitzarà mitjançant ConfigMaps de Kubernetes. Un ConfigMap és un objecte de l'API de Kubernetes que permet emmagatzemar la configuració dels diferents serveis en funció de l'entorn. Permet externalitzar les configuracions del codi font.
  - **Secrets:** Un secret és un objecte que conté una petita quantitat de dades confidencials codificats en Base64 com contrasenyes, un símbol, o una clau. Per evitar que aquesta informació es trobi dins del codi font en fitxers de propietats o de configuració, es poden utilitzar objectes de tipus Secret per injectar aquesta informació en temps d'operació. Es pot configurar l'accés a aquests Secrets perquè només usuaris autoritzats puguin veure i modificar el seu contingut, el que permet més control sobre com es fa servir, i redueix el risc de exposició accidental.
- **Serveis de logging:** Les eines que s'utilitzen són les que formen part del stack EFK:
  - **Elasticsearch:** Elasticsearch és un motor de cerca basat en Lucene. Proveeix un motor de cerca de text complet, distribuït i amb capacitat multitenant amb una interfície web RESTful i amb documents JSON.
  - **Fluentd:** Fluentd és un programari que recol·lecta les traces de log. Intercepta en temps real les traces escrites pels diferents microserveis i les envia a Elasticsearch per a la seva depuració i indexació.
  - **Kibana:** Kibana es una aplicació open source que proporciona capacitats de visualització de dades i de cerca de dades indexades a Elasticsearch. Permet

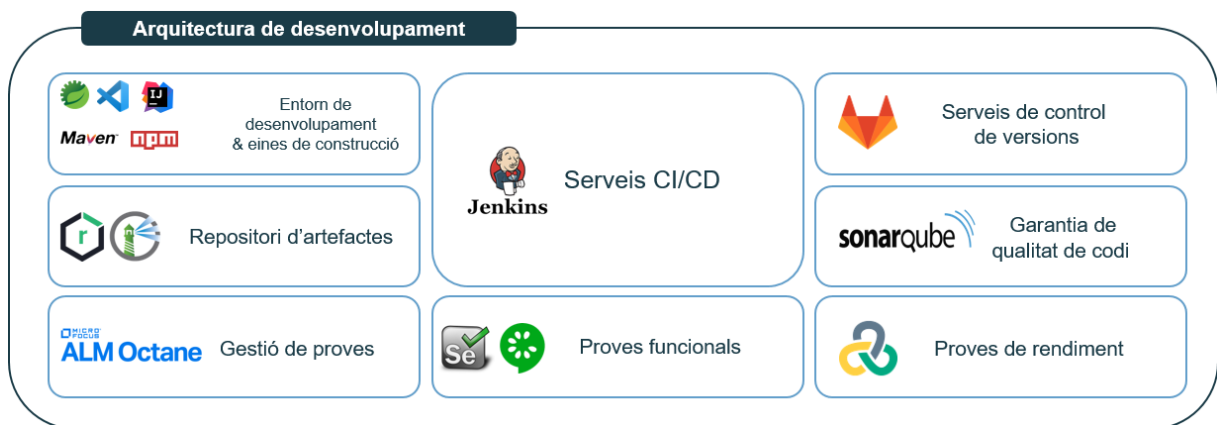
 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 38 / 58

visualitzar els logs de diferents serveis de l'aplicació a dashboards dissenyats a mida.

- **Serveis de monitorització:** Les eines de monitorització que s'utilitzen són:
  - **Prometheus:** Prometheus és una aplicació de programari lliure que s'utilitza per a la supervisió i alerta d'esdeveniments dels serveis. Registra mètriques en temps real en una base de dades de sèries de temps utilitzant un model d'extracció HTTP, amb consultes flexibles i alertes en temps real.
  - **Grafana:** Grafana és un programari de visualització de gràfics i anàlisi de codi obert. Permet consultar, visualitzar, alertar i explorar mètriques dels serveis a partir de sèries temporals emmagatzemats en una base de dades.

#### 2.4.2.1.3 Arquitectura de desenvolupament

A continuació, es descriuen els blocs lògics de l'arquitectura de desenvolupament, basant-se majoritàriament en les eines de les que disposa l'SIC (Servei d'Integració Continua) del CTTI. De forma general, en aquest apartat s'entén el codi des de un punt de vista ampli incloent també les definicions d'infraestructura.




Il·lustració 27 Disseny lògic. Arquitectura de desenvolupament

- **Entorn de desenvolupament i eines de construcció:**

Entorns de desenvolupament:

- **Eclipse (STS):** Spring Tool Suite (STS) és un conjunt d'eines per crear aplicacions Spring. Es pot instal·lar com a complement d'una instal·lació existent d'Eclipse JEE o es pot instal·lar independent. La versió independent de STS també s'inclou amb Eclipse EE, de manera que totes les funcions d'Eclipse per al desenvolupament de Java EE també estan disponibles a STS.
- **Visual Studio Code:** Visual Studio Code és un editor de codi font desenvolupat per Microsoft per a Windows, Linux i mac OS. Inclou suport per a la depuració, control integrat de Git, ressaltat de sintaxi, finalització intel·ligent codi,




 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 39 / 58

fragments i refactorització de codi. També és personalitzable, de manera que els usuaris poden canviar el tema de l'editor, les dreceres de teclat i les preferències. Aquest editor es fa servir majoritàriament per al desenvolupament de codi de front-end.

Eines de gestió de dependències i de construcció de codi:

- **Webpack:** Webpack és una eina de compilació empaquetació de codi estàtic per projectes basats en JavaScript. Col·loca en un graf de dependències a tots els elements que formen part del projecte de desenvolupament, como fitxers JavaScript, HTML, CSS, plantilles, imatges, fonts, etc., i genera un bundle amb el contingut estàtic.
- **Npm:** És el sistema de gestió de paquets per defecte per NodeJs. Per utilitzar paquets, el projecte ha de contenir un fitxer anomenat package.json. Dins d'aquest fitxer, s'especifiquen les metadades específics per al projecte. Les metadades ajuden a identificar el projecte i actuen com una línia de base perquè els usuaris obtinguin informació al respecte. NPM es compon de dues parts principals: una eina CLI per a la publicació i descàrrega de paquets, i un repositori en línia que alberga paquets de JavaScript. Pot ser usat tant en frontal com en backend.
- **Apache Maven:** Apache Maven és una eina que estandarditza la configuració d'un projecte en tot el seu cicle de vida, com per exemple en totes les fases de compilació i empaquetat i la instal·lació de mecanismes de distribució de llibreries, perquè puguin ser utilitzades per altres desenvolupadors i equips de desenvolupament. També contempla temes relacionats amb la integració contínua, per poder realitzar l'execució de test unitaris i proves automatitzades, test d'integració, etc. Maven s'utilitza sobretot per empaquetar projecte basats en Java. Altres llenguatges de programació com Go o .NET no requereixen de cap eina externa per empaquetar el codi, ja que inclouen aquesta funcionalitat de fàbrica.
- **Repositori d'artefactes:**
  - **Nexus:** Nexus és un repositori que permet guardar les diferents versions dels artefactes. Suporta una gran varietat de formats, inclòs Java i npm.
  - **Harbor:** Harbor és un registre de codi obert que protegeix els artefactes amb polítiques i control d'accés basat en rols, garanteix que les imatges siguin escanejats i estiguin lliures de vulnerabilitats, i signa les imatges com fiables. Harbor ofereix compliment, rendiment i interoperabilitat per ajudar a administrar artefactes de manera consistent i segura en plataformes de computació natives del núvol com Kubernetes i Docker.
- **Servei de control de versions:**
  - **GitLab:** Gitlab és un servei de control de versions i desenvolupament de programari col·laboratiu basat en Git. A més de gestor de repositoris, el servei

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 40 / 58

ofereix també allotjament de wikis i un sistema de seguiment de tasques i errors.

- **Proves de rendiment:**

- **Microfocus Load Runner:** Microfocus Load Runner és una eina per provar aplicacions, mesurar el comportament i el rendiment del sistema sota càrrega. LoadRunner pot simular milers d'usuaris simultàniament mitjançant un programari d'aplicació, enregistrant i analitzant posteriorment el rendiment dels components clau de l'aplicació. Simula l'activitat de l'usuari generant missatges entre components de l'aplicació o simulant interaccions amb la interfície d'usuari, com ara pressions de tecles o moviments del ratolí. Els missatges i les interaccions que es generaran s'emmagatzemen en scripts. LoadRunner pot generar els scripts registrant-los, com ara registrar peticions HTTP entre un navegador web client i el servidor web d'una aplicació.

- **Gestió de casos de prova:**

- **Microfocus ALM Octane:** ALM Octane és una solució de gestió del cicle de vida del programari centrada en millorar la velocitat, la qualitat i la capacitat d'ampliació del lliurament de programari per a les organitzacions que adopten la pràctica de lliurament de DevOps, Agile i Lean. Les característiques que ofereix ajuden als equips de desenvolupament a lliurar aplicacions de forma ràpida sense que la qualitat o l'experiència de l'usuari es vegin afectades.

- **Proves funcionals:**

- **Selenium:** Selenium és una eina que s'utilitza per automatitzar proves funcionals. Permet reproduir fluxos de navegació entre formularis web i pantalles simulant el comportament d'usuaris reals. Les proves funcionals s'han de fer per bloc funcional per garantir l'autonomia dels blocs funcionals i dels equips de desenvolupament verticals i sistemàticament per tenir proves més completes
- **Cucumber:** Cucumber és una eina per implementar metodologies com el Behaviour Driven Development (BDD) o desenvolupament basat en comportament, que permet executar i automatitzar proves funcionals definits en text pla.


- **Garantia de qualitat de codi:**

- **SonarQube:** SonarQube és una plataforma per avaluar la qualitat de la font, realitzant una anàlisi estàtica sobre aquest codi, amb l'objectiu de mostrar advertències sobre diferents punts a millorar. Es pot integrar amb diferents llenguatges de programació, com ara Java o JavaScript.

- **Serveis CI/CD:**

- **Jenkins:** Jenkins ajuda en l'automatització del procés de desenvolupament de programari mitjançant integració contínua i facilita aspectes del lliurament contínua. Admet eines de control de versions com CVS, Subversion i Git i pot executar projectes basats en Ant i Maven, així com seqüències d'ordres de



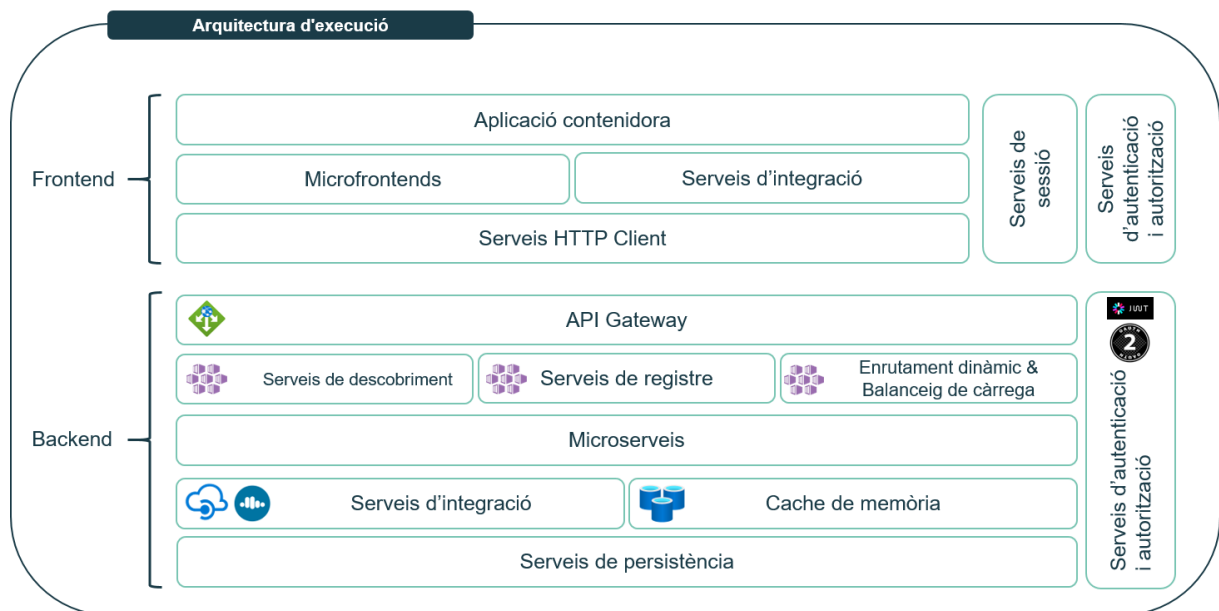
 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 41 / 58

consola i altres aplicacions. A més, es pot entregar amb eines de testing i control de qualitat. El autoservei del SIC del CTTI està basat en Jenkins. Aquest servei té la capacitat de generar automàticament les pipelines d'un projecte a partir d'una plantilla en què es descriuen els passos a executar.

## 2.4.2.2 Disseny lògic basat en el cloud públic Azure


### 2.4.2.2.1 Arquitectura d'execució

La descripció dels blocs lògics de l'arquitectura d'execució de frontend és la mateixa que en el disseny neutre pel que fa a el proveïdor de cloud, per tant, en aquest punt només es descriuen els punts referits a serveis de backend.




Il·lustració 28 Disseny lògic Azure. Arquitectura d'execució

- **API Gateway:** Azure Gateway és un servei que pot exercir com a punt únic d'entrada a l'aplicació. Ofereix la possibilitat de configurar regles per reenviar el tràfic als diferents serveis de què està compost l'aplicació. Inclou serveis que sondegen l'estat dels diferents serveis amb què està integrat per assegurar que només es reenvia tràfic a serveis operatius .
- **Serveis de descobriment, serveis de registre, enrutament dinàmic i balanceig de càrrega:**
  - **Azure Kubernetes Service (AKS):** Azure Kubernetes Service (AKS) és un servei de orquestració de contenidors basat en Kubernetes. Simplifica la implementació, l'administració i l'ús de Kubernetes com un servei orquestrador de contenidors totalment administrat. Els serveis desplegats a AKS es poden

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 42 / 58


gestionar a través de el propi portal d'Azure o en línia de comandament, utilitzant l'eina de administració kubectl de Kubernetes. AKS ofereix totes les funcionalitats que ofereix un sistema de Kubernetes estàndard, descrit en l'apartat de l'arquitectura de execució.

- **Serveis d'integració:**
  - **Integracions síncrones (HTTP):**
    - **Azure API Manager:** Azure API Manager és un servei totalment gestionat que permet als clients publicar, protegir, transformar, mantenir i controlar les API. Es pot crear una façana API que actua com una "porta principal" a través de la qual les aplicacions externes i internes poden accedir a les dades o a la lògica empresarial implementada pels serveis de backend personalitzats que s'executen a Azure, per exemple. al servei d'aplicacions o al servei Azure Kubernetes, o allotjat fora d'Azure, en un centre de dades privat o local. La gestió de l'API gestiona totes les tasques relacionades amb la mediació de les crides de l'API, incloses l'autenticació i l'autorització de sol·licituds, el límit de tarifes i l'aplicació de quotes, la transformació de sol·licituds i respostes, el registre i el seguiment i la gestió de versions de l'API.
  - **Integracions asíncrones (publish-subscribe):**
    - **Confluent Cloud:** Confluent Cloud és una plataforma de transmissió d'esdeveniments cloud-native totalment gestionada, impulsada per Apache Kafka. Apache Kafka és una plataforma de transmissió distribuïda de codi obert que permet construir aplicacions basades en esdeveniments a gran escala. Permet accelerar el desenvolupament de serveis basats en esdeveniments i aplicacions en temps real amb l'eliminació de tota la gestió de Kafka. Confluent Cloud simplifica la ingesta, integració i processament de dades a AWS, Azure i Google Cloud.
- **Cache de memòria:**
  - **Azure Cache for Redis:** Azure Cache for Redis proporciona un magatzem de dades en memòria basat en el programari de Redis. Redis millora el rendiment i l'escalabilitat d'una aplicació que faci servir en gran mesura els magatzems de dades de back-end. És capaç de processar grans volums de sol·licituds d'aplicació al mantenir en la memòria del servidor les dades als quals s'accedeix amb freqüència. En aquesta memòria es pot escriure i llegir ràpidament. Redis incorpora una solució crítica d'emmagatzematge de dades de baixa latència i alt rendiment en les aplicacions modernes.
- **Microserveis:** Els microserveis es poden desplegar / executar de dues maneres diferents:
  - **Com contenidors sobre AKS:** En aquesta tipologia de desplegament el microservei es desplega i executa dins d'un contenidor a ecosistema AKS.
  - **Azure Functions (Serverless):** Azure Functions és un servei serverless allotjat en el núvol pública de Microsoft Azure. Azure Functions i serverless, en

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 43 / 58

general, estan dissenyats per accelerar i simplificar el desenvolupament d'aplicacions. El desenvolupament d'aplicacions tradicionals exigeix tenir en consideració la infraestructura de TI subjacent, en aquest cas Azure Functions permet abstrure de la infraestructura i centrar-se en el desenvolupament. Característiques bàsiques d'Azure Functions:

- Azure Functions és una plataforma de còmput dirigida per esdeveniments (event-driven serverless computi platform).
  - Permet resoldre problemes complexos d'orquestració.
  - Permet executar codi sense configuracions addicionals de infraestructura
  - Escalat automatitzat i flexible basat en el volum de càrrega, mantenint el focus en afegir valor en lloc de gestionar la infraestructura
  - Model integrat de programació basat en disparador que ajuden a respondre a esdeveniments i connectar-se sense problemes amb altres serveis
  - Experiència de desenvolupament d'extrem a extrem, des de la creació i depuració fins al desplegament i supervisió amb eines integrades i capacitats de DevOps
  - Varietat de llenguatges de programació i opcions d'allotjament
- **Serveis de persistència:** Les diferents tipologies de bases de dades s'han explicat anteriorment en el punt 2.4.2.1.1. A continuació es detallen algunes de les bases de dades que ofereix Azure i l'ús més recomanat per a cadascuna d'elles.
  - **Azure Database for PostgreSQL:** Crear aplicacions escalables, segures i totalment administrades amb PostgreSQL de codi obert, escalar horitzontalment bases de dades PostgreSQL d'un sol node amb alt rendiment o migrar càrregues de treball de PostgreSQL i Oracle al núvol.
  - **MongoDB Atlas:** MongoDB Atlas és el servei de base de dades MongoDB en el núvol. Permet crear aplicacions amb baixa latència i alta disponibilitat o bé migrar les càrregues de treball NoSQL al núvol.
  - **Azure SQL Database:** Crear aplicacions modernes en el núvol amb un servei de base de dades relacional sempre actualitzat que inclogui procés sense servidor, emmagatzematge a hiperescala i característiques automatitzades i basades en intel·ligència artificial per optimitzar el rendiment i la durabilitat.
  - **Instància administrada de Azure SQL:** Migrar les càrregues de treball de SQL a Azure mantenint la compatibilitat amb SQL Server, amb una plataforma com a servei totalment administrada i perenne.
  - **SQL Server a Virtual Machines:** Migrar les càrregues de treball de SQL a Azure amb facilitat i mantenint la compatibilitat amb SQL Server i l'accés a nivell de sistema operatiu.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 44 / 58

- **Azure Database for MySQL:** Proporcionar alta disponibilitat i escalat elàstic a aplicacions web i mòbils de codi obert amb un servei de base de dades MySQL de la comunitat administrat, o bé migrar les càrregues de treball MySQL al núvol.
- **Azure Database for MariaDB:** Proporcionar alta disponibilitat i escalat elàstic a aplicacions web i mòbils de codi obert amb un servei de base de dades MariaDB de la comunitat administrat.
- **Azure Cosmos DB:** Crear aplicacions amb baixa latència i alta disponibilitat o bé migrar les càrregues de treball de Cassandra, MongoDB i altres càrregues de treball NoSQL al núvol.
- **Azure Cache for Redis:** Alimentar aplicacions ràpides i escalables amb un magatzem de dades en memòria que admeti codi obert.
- **Azure Database Migration Service:** Agilitzar la transició al núvol amb un senzill procés de migració autoguiat.
- **Azure Managed Instance for Apache Cassandra:** Modernitzar aplicacions i clústers de dades de Cassandra existents amb un servei d'instància administrada
- **Serveis d'autenticació i autorització:** Els serveis d'autenticació i autorització s'han explicat amb detall en l'apartat 2.4.2.1.1.


#### 2.4.2.2.2 Arquitectura d'operació

La descripció dels blocs lògics de l'arquitectura d'operació són les següents:



Il·lustració 29 Disseny lògic Azure. Arquitectura d'operació

- **Serveis de control:** Azure Kubernetes Service (AKS) està basat en Kubernetes i ofereix les mateixes funcionalitats per controlar els diferents serveis. Les capacitats de Kubernetes com servei de control estan detallats en l'apartat 2.4.2.1.2


 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 45 / 58

- **Serveis de gestió de la configuració:**

- **ConfigMaps:** La configuració dels microserveis es realitzarà mitjançant ConfigMaps de Kubernetes. Un ConfigMap és un objecte de l'API de Kubernetes que permet emmagatzemar la configuració dels diferents serveis en funció de l'entorn. Permet externalitzar les configuracions del codi font.
- **Secrets:** Azure Key Vault és un servei en el núvol per a l'emmagatzematge dels secrets i l'accés a aquests de forma segura. Un secret és tot allò l'accés desitja controlar de forma estricta, com les claus API, les contrasenyes, els certificats o les claus criptogràfiques. El servei Key Vault admet dos tipus de contenidors: magatzems i grups HSM administrats. Els magatzems permeten emmagatzemar programari i claus, secrets i certificats recolzats per HSM.


- **Serveis de monitorització, de logging i d'observabilitat:**

- **Azure Monitor:** Azure Monitor inclou tant funcionalitats de servei de monitorització com de servei de logging i d'observabilitat. Maximitza la disponibilitat i el rendiment de les aplicacions i serveis amb una completa solució que permet recopilar, analitzar i administrar dades telemètriques tant en el núvol com a entorns locals. Aquesta solució ajuda a entendre com funcionen les aplicacions i li permet identificar de manera proactiva els problemes que els afecten i els recursos de què depenen. Entre els exemples del que pot fer amb Azure Monitor s'inclouen:
  - Detecció i diagnòstic de problemes en aplicacions i dependències amb Application Insights.
  - Correlació de problemes d'infraestructura amb Azure Monitor per a màquines virtuals i Azure Monitor per a contenidors.
  - Aprofundiment en les seves dades de supervisió amb Log Analytics per a la solució de problemes i diagnòstics profunds.
  - Suport tècnic d'operacions a escala amb alertes intel·ligents i accions automatitzades.
  - Creació de visualitzacions amb panells de Azure.
- **Azure App Insights:** Azure App Insights és un servei que proporciona eines per monitoritzar, analitzar i detectar errors de rendiment en les aplicacions allotjades en aquest núvol. A més, també ofereix la possibilitat d'inserir traces personalitzades i registrar errors en aquestes aplicacions. La utilització del servei permet tenir d'una forma ràpida i precisa, les dades d'anàlisi relacionats amb el rendiment i funcionament de l'aplicació. També ofereix la possibilitat de configurar alertes personalitzades, en les quals, per exemple, podem rebre un correu electrònic cada vegada que es produeixi un determinat error.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	<i>Arquitectura de referència microserveis</i>	N. revisió doc.: <i>0.7</i>
	<b>Descripció de l'arquitectura de referència</b>	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 46 / 58

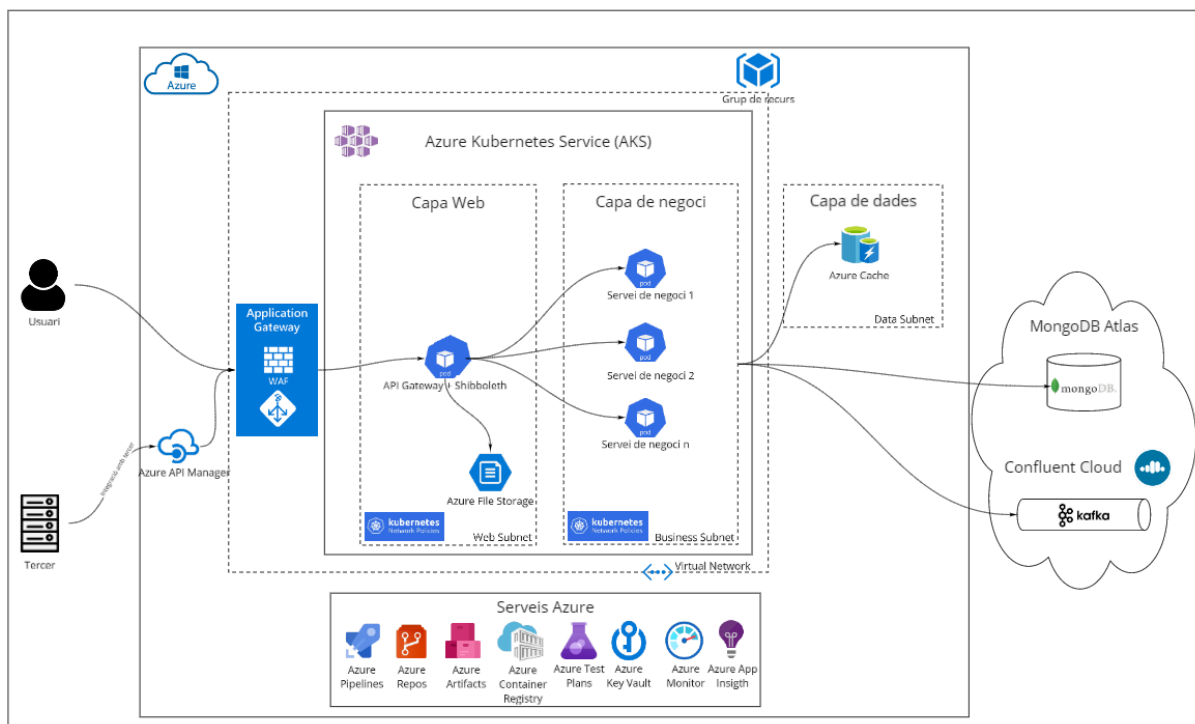
#### **2.4.2.2.3      *Arquitectura de desenvolupament***

De forma general, en aquest apartat s'utilitzaran les eines descrites amb detall en l'apartat: Disseny lògic neutre pel que fa al cloud / Arquitectura de desplegament on es fa referència a eines SIC i no SIC proporcionades per CTTI per resoldre tots els serveis necessaris en aquest building block.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 47 / 58


### 2.4.3 Disseny físic

El disseny físic identifica i descriu els components que formen part o integren el sistema, i com aquests es configuren o interactuen. Aquest disseny físic és la implementació de el disseny lògic basat en cloud públic d'Azure.




II·l·lustració 30 Disseny físic basat en Azure

- **Azure:** El disseny físic es basa en el disseny lògic basat en el cloud públic d'Azure
- **Azure Kubernetes Service (AKS):** Com ja s'ha comentat anteriorment, AKS és un servei de Kubernetes completament gestionat en cloud públic Azure.
- **Grup de recurs:** Un grup de recurs és un contenidor que emmagatzema els recursos relacionats amb la solució de Azure.
- **Application Gateway:** Azure Application Gateway, es tracta d'un gateway o passarel·la entre l'usuari i l'aplicació que té diferents responsabilitats:
  - Permet gestionar el trànsit a diferents servidors de la teva backend.
  - També és un Web Application Firewall, amb les opcions de detecció o prevenció.
  - Terminador SSL.
  - Redireccionador de peticions.
  - Afinitat de sessió amb el servidor.
  - Compatibilitat nativa amb HTTP2 i websocket.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 48 / 58

- **Capa Web:** La capa o subxarxa de web serà la zona on es desplegaran els components de front. Aquesta subxarxa està protegida amb Kubernetes Network Policies:
  - **API Gateway:** És desplegarà un contenidor NGINX com API Gateway.
  - **Azure File Storage:** Es disposarà d'Azure File Storage per a l'emmagatzematge dels recursos estàtics de front.
- **Capa de negoci:** La capa o subxarxa de negoci serà la zona on es desplegaran els microserveis de negoci. Aquesta subxarxa està protegida amb Kubernetes Network Policies.
- **Capa de dades:** La capa o subxarxa de dades és la zona on es disposaran els serveis gestionats de bases de dades:
  - **Azure Cache:** Es farà ús d'un servei gestionat de Azure Cache.
- **Serveis Azure:** Es farà ús dels següents serveis Azure:
  - Azure Pipelines
  - Azure Repos
  - Azure Artifacts
  - Azure Container Registry
  - Azure Test Plans
  - Azure Key Vault
  - Azure Monitor
  - Azure App Insights
- **Serveis externs desplegats en Azure:**
  - **MongoDB Atlas:** Es disposarà d'una base de dades MongoDB en Atlas.
- **Marketplace Azure:**
  - **Confluent Cloud:** Es disposarà d'un servei gestionat Apache Kafka a Confluent Cloud.



 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 49 / 58

### 3. Procés de creació de l'arquitectura de referència

#### 3.1 Principis de definició de l'arquitectura de referència

Els principis especificats en aquest apartat, juntament amb els [“Principis d'arquitectura de sistemes d'informació”](#) de CTTI [CTT21], han guiat el procés de definició, com a tal, de l'arquitectura de referència. És important distingir-los dels principis d'ús de l'arquitectura de referència de microserveis (veure 2.1) que indiquen quines són els criteris principals de disseny que ha de seguir una solució que usi l'arquitectura de referència.

##### 3.1.1 Orientació a l'autonomia

L'arquitectura de referència ha de permetre als arquitectes que dissenyen sistemes per a la Generalitat validar, en la mesura del possible, la conformitat amb aquesta i comprovar que el disseny s'adhereix a les recomanacions i/o guies proposades.

Aquest principi ve motivat perquè la dependència d'equips centralitzats per la presa de decisions crea colls d'ampolla i condiona les planificacions, posposant l'obtenció de resultats tangibles per als promotors dels sistemes. L'orientació a l'autonomia ha de minimitzar aquestes situacions i facilitar la presa de decisions sobre les arquitectures a dissenyar i les posteriors revisions que realitzi la Unitat d'Arquitectura. Aquest afegeix un esforç de publicació i difusió dels criteris d'aplicabilitat i que siguin prou entenedors utilitzant el llenguatge propi dels arquitectes del sector i dels que són proveïdors de la Generalitat en concret. Aquest llenguatge compartit facilitarà la seva avaluació i comprensió per la comunitat d'arquitectes i accelerarà la seva implementació.

##### 3.1.2 Contemplar que l'arquitectura ha d'evolucionar junt amb la tecnologia


La tecnologia canvia molt ràpidament i la vinculada a l'estil arquitectural de micro-serveis no és una excepció, per tant, no es concep l'arquitectura com quelcom rígid sinó en una evolució constant que la pugui fer vigent per molts canvis que es produeixen en les tecnologies vigents.

Aquest plantejament ve motivat per la curta perspectiva temporal de vigència de les tecnologies subjacents. Aquest fet condiona tant el tipus d'exercici a realitzar com el govern de l'arquitectura resultant. En aquest sentit, es consideren més estables les fases de disseny més conceptuals que les més properes al disseny físic. Les pautes donades a nivell lògic i físic han de estar basades més en patrons que en paradigmes tecnològics concrets.

Tot això implica que el disseny ha d'estar, en la mesura, orientat a l'evolució sent recomanable un govern de l'arquitectura de referència que la revisi de forma periòdica, per exemple, disposant full de ruta de desenvolupament revisat de forma periòdica, per poder ajustar aquells aspectes de l'arquitectura que hagin envellit. Es considera preferible fer un exercici complet però àgil versus un llarg procés formal de definició.

#### 3.2 Decisions del procés de creació de l'arquitectura


Durant el procés de definició de l'arquitectura de referència s'han pres decisions, de caràcter general, que n'han condicionat l'exercici i, per tant, el resultat.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 50 / 58

Títol	Descripció	Relacionat amb ...
Alternativa basada en contenidors	Es considera que, partint del disseny conceptual, podem tenir diverses realitzacions tecnològiques. Per exemple es pot tenir una realització tecnològica basada en el paradigma serverless, una sobre PaaS, sobre contenidors, etc.  Es determina treballar alternatives basades en contenidors	Disseny lògic
Disseny conceptual sense referència a contenidors	Es considera que, a nivell conceptual, no aplica les referències a capacitats basades en contenidors perquè, aquesta, és una alternativa de realització tecnològica.	Disseny conceptual
Disseny lògic orientat a Azure i Openshift	Es determina realitzar, a banda d'un disseny lògic basat en Openshift, un disseny lògic basat en Azure per evitar implementar, sobre cloud Azure, un disseny lògic poc òptim i natural sobre el cloud	Disseny lògic
Decisions tecnològiques no prescriptives	L'arquitectura de referència no serà, des del punt de vista tecnològic, prescriptiva en el sentit d'obligatorietat en l'ús de determinades tecnologies per evitar-ne una ràpida obsolescència i un cost alt de manteniment.	Disseny lògic
Documentació tècniques disseny	No és objectiu de l'arquitectura de referència documentar tècniques o metodologies de les que existeix bibliografia suficient com, per exemple, Domain-driven design. [EVA03] S'acorda fer-ne breu esment i referenciar a la bibliografia existent	Models d'ús
Heterogeneïtat tecnològica	Es considera interessant exercitar l'heterogeneïtat tecnològica en el frontend en quant a diferents versions del mateix framework JavaScript puguin coexistir en vistes a facilitar processos graduals de solució de l'obsolescència tecnològica dels sistemes.. De forma similar, també es considera d'interès provar, a nivell de backend, algun framework orientat a micro-serveis com quarkus per comparar beneficis respecte a Spring Boot. A nivell de BBDD, no es considera necessari provar heterogeneïtat de tecnologies	PoC

### 3.3 Prova de concepte

Per posar en pràctica l'arquitectura de referència s'ha desenvolupat una PoC d'una PetStore. La aplicació desenvolupada ha estat publicada al cloud públic d'Azure. La PoC que s'ha desenvolupat està composta pels següents blocs funcionals:

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 51 / 58

En general, en els microfrontends, s'han utilitzat diferents versions d'un mateix framework de desenvolupament (Angular) (<https://angular.io/>) amb la intenció de provar la viabilitat d'executar un procés gradual de gestió de l'obsolescència. En aquest cas es pot dir que és viable l'execució d'aquest tipus de processos i que les tecnologies proposades permeten la convivència de diferents versions de la mateixa framework i així com de diferents frameworks (encara que aquesta última part no s'ha provat explícitament).

En els backends s'han provat diferents frameworks i també llenguatges. La conclusió que podem obtenir d'aquest exercici és que té certs avantatges l'ús de llenguatges i / o frameworks cloud native més moderns. En general, presenten un millor rendiment, millor gestió de la concurrència, millors temps d'arrencada en contraposició que a l'ésser llenguatges i / o frameworks menys madurs o amb menys recorregut és més difícil muntar equips amb experiència sòlida en els mateixos.

Pel que fa a la capa de base de dades, a nivell general, en l'aplicació PetStore, s'ha fet ús d'una base de dades MongoDB en Atlas administrada i en el núvol.

A nivell general, a la PetStore, s'ha fet ús d'un sistema de missatgeria asíncrona basat en Apache Kafka a confluent Cloud (<https://confluent.cloud/>) administrat i en el núvol. S'ha mesurat la latència que pot haver-hi entre les operacions d'escriptura i lectura en els topics i, de mesurava, està sobre els 15ms.

### 3.3.1 Bloc funcional PetStore

Aquest bloc funcional és el responsable de la gestió d'usuaris, mostrar al l'usuari els avisos i alertes rellevants per a ell i a més, té la responsabilitat de ser l'aplicació contenidora.

#### 3.3.1.1 Microfrontend

El microfrontend d'aquest bloc funcional esta desenvolupat amb Angular v11.0.6


#### 3.3.1.2 Microservei

El microservei d'aquest bloc funcional està desenvolupat amb SpringBoot v2.4.1 (<https://spring.io/projects/spring-boot>). Spring Boot és un dels frameworks de microserveis més populars en l'actualitat. Està basat en Java i proporciona un conjunt d'eines per a construir ràpidament aplicacions Spring que siguin fàcils de configurar i de desenvolupar.

#### Avantatges:

Spring és un ecosistema molt sòlid i provat. Hi ha una gran varietat de llibreries compatibles amb Spring, que permeten estendre les seves capacitats i funcionalitats, com la securització del servei de la Petstore amb un token JWT. A més, Spring es pot integrar fàcilment amb la majoria de sistemes externs existents al mercat, com MongoDB i Kafka en el cas del microservei de la Petstore. Finalment, pel fet que es tracta d'un ecosistema molt madur de molts anys, és fàcil trobar desenvolupadors que dominen aquesta tecnologia.

#### Desavantatges:

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 52 / 58

El gran desavantatge de Spring en comparació amb Quarkus i Go és el seu temps d'arrencada. En el cas de la Petstore, el microservei necessita gairebé 5 segons per arrencar, el que en la majoria dels casos no hauria de ser un problema. No obstant això, l'ecosistema de Spring no és tan apte per a tecnologies serverless, que requereixen temps d'arrencada molt curts.

### 3.3.1.3 BDD

Aquest bloc funcional fa us d'una base de dades MongoDB en Atlas (aq\_petstore\_db).

### 3.3.1.4 Missatgeria

Aquest bloc funcional fa us d'un sistema de missatgeria asíncron Apache Kafka en Confluent Cloud per a la lectura dels avisos i alertes que la resta de microserveis escriuen. Llegeix del següent tòpic: avis-nou

## 3.3.2 Bloc funcional de mascotes

Aquest bloc funcional és el responsable de la gestió de mascotes i catàleg de mascotes.

### 3.3.2.1 Microfrontend

El microfrontend d'aquest bloc funcional està desenvolupat amb Angular v9.1.13 encapsulat com a Web Component fet us de


### 3.3.2.2 Microservei

El microservei d'aquest bloc funcional està desenvolupat amb Go v1.15.8 (<https://golang.org/>). Go és un llenguatge de programació concurrent i compilat inspirat en la sintaxi de C, que intenta ser dinàmic com Python i amb el rendiment de C o C ++. Go ha estat dissenyat per Google.

#### Avantatges:

- **Fase de desenvolupament:** Go encara no és una tecnologia molt estesa. Tot i això, ha estat possible cobrir tots els requeriments del microservei de Mascotes amb Go. El servei s'ha pogut protegir amb un token JWT i s'ha pogut integrar sense problemes amb la base de dades de MongoDB. Per requeriments del servei, no ha estat necessari la integració amb Kafka, encara que també hi ha documentació que cobreix aquesta necessitat (<https://docs.confluent.io/clients-confluent-kafka-go/current/overview.html>). No obstant això, sembla que aquesta integració podria ser una mica més complexa que en el cas de Spring i Quarkus, pel fet que requereix d'un compilador de C.
- **Fase de compilació i d'execució:** Al contrari de serveis basats en Java, Go no necessita cap màquina virtual per executar els serveis. Quan es compila un projecte de Go, es genera un fitxer executable que no requereix de cap programari addicional per a ser executat. El temps d'arrencada de l'microservei de Mascotes també és molt curt, ja que el servei triga tot just un segon a arrencar. Això vol dir que serveis desenvolupats a Go són aptes per a tecnologies serverless.

#### Desavantatges:

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 53 / 58

Go encara no és una tecnologia molt estesa. Per aquesta raó, la integració amb altres tecnologies i sistemes externs podria ser encara una mica més complicat que en el cas de Spring o Quarkus. Tot i que en el cas del microservei de Mascotes no s'han trobat dificultats, és possible que hi hagi encara requeriment que difícilment es puguin cobrir amb Go. Finalment, pel fet que el llenguatge de programació de Go encara no és un llenguatge molt establert, podria ser difícil trobar desenvolupadors que dominen aquesta tecnologia.

### 3.3.2.3 BDD

Aquest bloc funcional fa us d'una base de dades MongoDB en Atlas (aq\_mascotes\_db).

## 3.3.3 Bloc funcional de comandes

Aquest bloc funcional és el responsable de la gestió de comandes.

### 3.3.3.1 Microfrontend


El microfrontend d'aquest bloc funcional esta desenvolupat amb Angular v

### 3.3.3.2 Microservei

El microservei d'aquest bloc funcional està desenvolupat amb Quarkus v1.11.1.Final (<https://quarkus.io/>) que està basat en Java, però se sol utilitzar amb una màquina virtual específica basada en GraalVM. D'aquesta manera es pretén que els temps d'arrencada siguin mínims. No obstant això, Quarkus també es pot utilitzar amb una màquina virtual JVM clàssica.

#### Avantatges:

- **Fase de desenvolupament:** Durant la implementació del servei no s'han trobat dificultats o restriccions importants. S'ha pogut integrar el servei de Comandes sense problemes amb tots els serveis necessaris, com ara Kafka (<https://quarkus.io/guides/kafka>), MongoDB (<https://quarkus.io/guides/mongodb-panache>) o crides REST síncrones al microservei de Mascotes (<https://quarkus.io/guides/rest-client>). Per a la integració amb MongoDB disposa de la llibreria quarkus-mongodb-panache que implementa el patró de Entity i Repository per poder accedir a la base de dades mitjançant ORM (Object-Relational Mapping), tal com es coneix d'Hibernate. A més, s'ha pogut protegir el servei sense problemes mitjançant un token JWT, implementant un accés als diferents serveis en funció del rol d'usuari. A la pàgina oficial de Quarkus es disposa d'una gran varietat d'exemples, on es mostra com integrar Quarkus amb una gran varietat de tecnologies.
- **Fase de compilació:** Per facilitar les tasques de compilació i empaquetatge es pot utilitzar Maven. A més, quan es crea un projecte Quarkus es generen automàticament fitxers Docker per poder executar el servei en una màquina virtual nativa amb GraalVM o en una JVM clàssica.
- **Fase de execució:** Una de les grans avantatges de Quarkus és el temps d'arrencada. Amb una JVM clàssica el microservei de comandes s'aixeca en menys d'un segon, el que habilita l'ús de Quarkus per a serveis serverless. Lamentablement no s'ha pogut mesurar el temps d'arrencada utilitzant el mode natiu amb GraalVM, tal com s'explica a continuació.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 54 / 58

#### Desavantatges:


- L'únic desavantatge que s'ha trobat amb l'ús de Quarkus és que de cap manera natiu no suporta URL que utilitzen el protocol mongodb+srv per connectar amb MongoDB, tal com s'explica en <https://quarkus.io/guides/mongodb>. Atès que la base de dades MongoDB facilitat en l'entorn Azure de la Petstore utilitza aquest protocol, el microservei de Comandes s'ha hagut d'arrencar utilitzant una JVM clàssica.

#### **3.3.3.3 BDD**

Aquest bloc funcional fa us d'una base de dades MongoDB en Atlas (aq\_comandes\_db).

#### **3.3.3.4 Missatgeria**

Aquest bloc funcional fa us d'un sistema de missatgeria asíncron Apache Kafka en Confluent Cloud per escriure avisos i alertes que el microservei PetStore llegeix i mostra a l'usuari /s a qui va destinat. Escriu en el següent tòpic: avis-nou

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 55 / 58

#### 4. Gap respecte a la situació actual i serveis potencialment comuns

##### 4.1 Gap respecte a la situació actual

En aquest apartat s'identifiquen gaps que s'han identificat de la revisió de l'arquitectura descrita a l'apartat anterior enfront el document de disseny d'arquitectura actual, el sistema d'integració contínua (SIC) i el catàleg de serveis TIC.

##### Disseny d'arquitectura (DA)

- **Resum de recursos:** La quantitat de recursos individuals que cal per aquest tipus de arquitectures fa difícil el tractament de la taula de recursos de la vista de desplegament. Es considera necessari afegir algun tipus de taula resum per tenir les necessitats agregades.
- **Diagrames de seqüència escenaris principals d'ús:** Incorporar recomanació i exemples per documentar els escenaris d'ús principals i els diagrames de seqüència corresponents.
- **Models compartits:** Incorporar recomanació i exemples per documentar el model d'informació global i els particulars de cada servei identificant, a alt nivell, les parts compartides.


##### Servei d'integració contínua (SIC)

- **Versió global de solució:** Poder disposar d'un codi de versió únic de la solució que mantingui la relació amb el codis de versió dels diferents components que la componen en una configuració determinada. Aquesta capacitat pot estar fora de l'abast del SIC i requerir algun altre tipus d'eina.
- **Infraestructura com a codi:** Las pipelines del SIC haurien de permetre el desplegament de la infraestructura configurada (com a codi).
- **Desplegament en cloud pública:** Permetre desplegament d'infraestructura com a codi en plataformes de orquestració en cloud pública.
- **Pipelines de contenidors en mode auto-servei:** Disposar de pipelines d'autoservei per a la construcció i desplegaments de projectes basats en contenidors.

##### Catàleg de serveis TIC

- **Flexibilitat en les relacions RAM/CPU:** La diferent naturalesa de components que s'aixequen en contenidors requereix de major flexibilitat en les talles i / o relació RAM / CPU dels elements del catàleg per optimitzar l'assignació de recursos i els costos
- **Eines de monitorització al catàleg:** Afegir eines de monitoratge al catàleg de serveis ajudaria a homogeneïtzar l'ús d'aquestes i a més asseguraria que les eines que es facin servir estiguin aprovades per CTTI.
- **Serveis d'integració:** Ídem anterior.



 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 56 / 58

- **Serveis de cache de memòria:** Ídem anterior.
- **Serverless en arquitectures de contenidors en cloud privat:** Permetria potenciar i homogeneïtzar l'ús d'aquesta forma de desplegament / execució

## 4.2 Serveis potencialment comuns


Dins el disseny conceptual (veure apartat 2.4.1) hi ha serveis que, per la seva naturalesa, es poden plantejar com a comuns d'arquitectura. S'ha definit uns criteris a partir dels quals s'identifiquen quins serveis són potencialment comuns:

- **Explotació informació arquitectura:** La transversalització del servei pot permetre a la unitat d'arquitectura o a les corresponents d'operacions treure informació valuosa resultant de la consolidació, en un únic servei, de totes les solucions.
- **Cost molt elevat:** Costos molt elevats per ser assumit per una única solució (costos de llicenciament, infraestructura, etc.)
- **Accés a informació compartida:** Quan les diferents solucions poden usar el servei per compartir informació.
- **Inter/Intra solució:** El servei aporta capacitats que tant poden ser usades dins d'una solució com ser compartides i usades conjuntament per diverses solucions, per exemple, serveis de missatgeria que tant poden ser usats per comunicar microserveis d'una mateixa solució com per comunicar dues solucions diferents.


Partint d'aquests criteris, s'identifiquen els següents blocs com a potencialment reutilitzables:

- **Arquitectura d'execució:**
  - Serveis d'integració:
    - (Inter/Intra solució) Els serveis poden transportar tant missatges interns entre micro-serveis com entre diferents solucions
    - (Explotació informació arquitectura) La visibilitat dels missatges, especialment, inter- aplicacions pot ser valuosa i dona opcions d'arquitecturar la missatgeria, és a dir, d'establir, per exemple, unes capçaleres comuns que permetin fer algun tipus d'anàlisi, diagnòstic de problemes d'integració, etc.
  - Memory cache
    - (Inter/Intra solució) Els serveis poden servir per intercanvi d'informació entre solucions
  - Serveis d'autenticació i autorització:
    - (Accés a informació compartida) Els serveis usen el directori corporatiu per autenticar-se
- **Arquitectura d'operació:**
  - Tots els serveis de l'arquitectura d'operació:
    - (Explotació informació arquitectura) L'explotació de tots els serveis de l'arquitectura d'operació poden aportar valor des d'una visió agregada de múltiples solucions.
    - (Cost molt elevat) El desplegament de tota l'arquitectura d'operació per cada solució pot implicar costos molt elevats.



 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	<i>Arquitectura de referència microserveis</i>	N. revisió doc.: <i>0.7</i>
	<b>Descripció de l'arquitectura de referència</b>	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 57 / 58

- **Arquitectura de desenvolupament:** Actualment ja està, majoritàriament, com a serveis comuns.

 Generalitat de Catalunya Centre de Telecomunicacions i Tecnologies de la Informació	Arquitectura de referència microserveis	N. revisió doc.: 0.7
	Descripció de l'arquitectura de referència	
	N. versió arquitectura referència : 0.7 Build: 0	Pàg. 58 / 58

## 5. Referències

[AZ21] Microservices on Azure – What Is Microservices | Microsoft Azure [Internet]. [citat 18 març 2021]. Disponible a: <https://azure.microsoft.com/en-us/solutions/microservice-applications/>

[AZU21] Improving observability of your Kubernetes deployments with Azure Monitor for containers [Internet]. [citat 18 març 2021]. Disponible a: <https://azure.microsoft.com/en-us/blog/improving-observability-of-your-kubernetes-deployments-with-azure-monitor-for-containers/>

[BRA13] Brandolini A. Introducing Event Storming [Internet]. Ziobrando's Lair. 2013 [citat 18 març 2021]. Disponible a: <http://ziobrando.blogspot.com/2013/11/introducing-event-storming.html>

[CTT21] Principis d'arquitectura de sistemes d'informació [Internet]. [citat 18 març 2021]. Disponible a: [https://canigo.ctti.gencat.cat/arqctti/principis\\_arq/](https://canigo.ctti.gencat.cat/arqctti/principis_arq/)

[EVA03] Evans E. Domain-driven design: tackling complexity in the heart of software. Boston, MA: Addison-Wesley; 2003.

[HAR21] Harbor [Internet]. [citat 18 març 2021]. Disponible a: <https://goharbor.io/>

[HON21] The Microservices Observability Problem [Internet]. Honeycomb. [citat 18 març 2021]. Disponible a: <https://www.honeycomb.io/microservices/>

[VER13] Vernon V. Implementing domain-driven design. Upper Saddle River, NJ; Mexico City: Addison-Wesley; 2013.

[YOU79] Yourdon E, Constantine LL. Structured design: fundamentals of a discipline of computer program and systems design. Englewood Cliffs, N.J.: Yourdon Press; 1979.