

Desplegament d'aplicacions web mitjançant plataformes Cloud



Quan desenvolupem una aplicació web no tenim per què encarregar-nos de tots els components necessaris perquè funcione. Podem desenvolupar i administrar nosaltres mateixos el servidor, el sistema operatiu, l'entorn d'execució, la base de dades i l'aplicació, o podem delegar part d'estes tasques en proveïdors cloud.

Des del punt de vista d'un programador, la principal diferència entre els diferents models de servicis cloud és quina part de la infraestructura i del programari gestionem nosaltres i quina part proporciona el proveïdor. En esta unitat estudiarem les principals opcions i conceptes necessaris per a comprendre l'ecosistema actual de desplegament en el núvol, i practicarem amb un exemple en alguna d'elles.

1. Models de servicis cloud

Els **servicis cloud** poden classificar-se atés el nivell de responsabilitat que assumix el proveïdor enfront del desenvolupador. La classificació tradicional distinxix tres models principals:

- **IaaS (Infrastructure as a Service)**
- **PaaS (Platform as a Service)**
- **SaaS (Software as a Service)**

Estos models representen diferents nivells d'abstracció i de responsabilitat entre el proveïdor i el desenvolupador. A continuació expliquem les característiques principals de cada un.

1.1 IaaS - Infrastructure as a Service

IaaS o Infraestructura com a Servici consistix a contractar recursos informàtics com a màquines virtuals, emmagatzematge, xarxes o capacitat de processament. El proveïdor s'encarrega de la infraestructura física, però nosaltres continuem havent d'administrar gran part de l'entorn en el qual s'executarà la nostra aplicació.

Per exemple, podem contractar una màquina virtual i instal·lar en ella Linux, Node.js, Express i la nostra aplicació. També haurem d'encarregar-nos de tasques com les actualitzacions, la configuració de la xarxa, la seguretat i el manteniment del sistema. Proporciona, per tant, molt de control, però també molta responsabilitat. Alguns exemples són **Amazon EC2**, **Azure Virtual Machines** o **Google Compute Engine**.

1.2 PaaS - Platform as a Service

PaaS o Plataforma com a Servici proporciona un entorn preparat per a executar les nostres aplicacions. El proveïdor s'encarrega de gran part de la infraestructura i nosaltres ens centrem principalment a desenvolupar, configurar i desplegar el nostre programari. Alguns exemples són **Render**, **Railway**, **Google App Engine** o **Azure App Service**.

Podem connectar un repositori de GitHub amb una plataforma com Render i configurar els comandos necessaris per a construir i executar el nostre backend Node.js + Express. La plataforma s'ocupa de gran part de les tasques relacionades amb el servidor i nosaltres ens centrem en la nostra aplicació. PaaS ofereix, per tant, menys control que IaaS, però simplifica considerablement el desplegament i el manteniment.

1.3 SaaS — Software as a Service

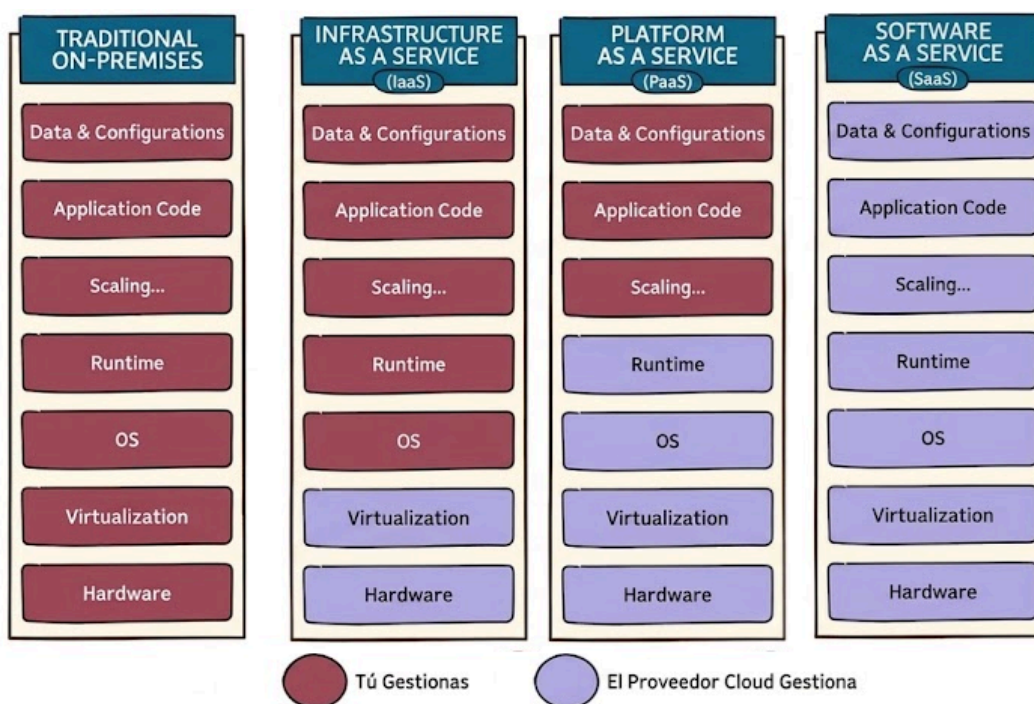
SaaS o Programari com a Servici és el nivell més alt d'abstracció en el núvol. Consistix a utilitzar una aplicació completament desenrotllada, allotjada i mantinguda per un tercer, a la qual solem accedir a través d'un navegador web o mitjançant una API.

En este model, el proveïdor s'encarrega de totes les capes: infraestructura física, sistema operatiu, base de dades, codi de l'aplicació i escalabilitat. Nosaltres únicament consumim el servici i gestionem el nostre propi compte o dades.

Tradicionalment està orientat a usuaris finals (com Google Workspace, Jira o GitHub) però, des de la perspectiva d'un desenvolupador, este concepte s'estén a la integració de servicis especialitzats via API o SDK. En lloc de construir subsistemes complexos des de zero, consumim solucions ja operatives per a:

- Processament de pagaments (ex. Stripe, PayPal).
- Enviaments de correu transaccional (ex. SendGrid, Resend).
- Autenticació i gestió d'usuaris (ex. Auth0, Clerk).
- Anàlítica i monitoratge (ex. PostHog, Sentry).

El principal avantatge és que podem incorporar funcionalitats ràpidament sense haver de desenrotllar i mantindre tota la infraestructura necessària. A canvi, la nostra aplicació passa a dependre del proveïdor, dels seus preus, límits, disponibilitat i API.



1.4. Altres models

Amb l'evolució de la computació en el núvol han aparegut altres models que descriuen formes específiques de consumir servicis cloud, com:

- **FaaS (Function as a Service)**: és un model associat a les arquitectures *serverless*. En lloc de desplegar i mantindre un servidor que execute contínuament la nostra aplicació, desenvolupem funcions individuals que s'executen quan reben una petició o quan ocorre determinat esdeveniment. Resulta especialment interessant per a determinades APIs, tasques automàtiques i aplicacions el trànsit de les quals pot variar molt. Alguns exemples són **AWS Lambda**, **Azure Functions**, **Vercel Functions** o **Netlify Functions**.
- **BaaS (Backend as a Service)**: proporciona funcionalitats de backend ja desenrotllades que podem integrar directament en la nostra aplicació, com a autenticació d'usuaris, bases de dades, emmagatzematge d'arxius, notificacions o APIs. Alguns exemples són **Firebase**, **Supabase** o **Appwrite**.
- **DBaaS (Database as a Service)**: el proveïdor ofereix una base de dades llista per a utilitzar. Nosaltres ens encarreguem de dissenyar i utilitzar la base de dades, mentre que el proveïdor administra gran part de la infraestructura necessària per a executar-la. Un exemple conegut en este curs pot ser **MongoDB Atlas**.

Estos conceptes no han d'entendre's necessàriament com a categories independents al mateix nivell que IaaS, PaaS i SaaS, sinó com a formes especialitzades d'oferir o consumir determinats servicis cloud.

També cal esmentar que estos models no són necessàriament excloents. Una aplicació web moderna pot combinar diferents servicis cloud. Per exemple, podem desenvolupar un *frontend* Angular i un *backend* Node.js + Express, i desplegar-los en una plataforma com Render, i utilitzar MongoDB Atlas com a servici de base de dades. D'esta manera, podem construir una aplicació combinant programari desenvolupat per nosaltres amb infraestructura i servicis proporcionats per tercers.

2. Exemple: Render i MongoDB Atlas

Per a mostrar un exemple de tot això emprem **Render**. En els següents apartats provarem de desplegar una API REST amb *Express* i *Mongoose*, allotjant la base de dades en **MongoDB Atlas**.

2.1. Característiques principals de Render

Com hem comentat anteriorment, Render entra dins de la categoria de *PaaS (Platform as a Service)*, i ens proporciona un entorn completament configurat per a executar les nostres aplicacions web. És especialment interessant per al nostre cas perquè permet desplegar actualment servicis web Node.js/Express directament des d'un repositori GitHub o similar, i disposa d'una modalitat gratuïta per a experimentar ([web oficial](#))

En Render podem desplegar tant servidors *backend* persistents (desenrotllats en Node.js, Python, Go o qualsevol tecnologia empaquetada amb Docker), com a llocs web *frontend* en HTML/CSS/JS servits de manera ràpida i gratuïta a través d'una xarxa CDN. A més, permet allotjar i gestionar bases de dades PostgreSQL i Redis, i també connectar amb altres bases de dades remotes.

Com a punt fort, permet mantindre processos en execució contínua (com a servidors Express o WebSockets) i ofereix xarxes privades per a connectar de manera segura el nostre backend amb la base de dades sense

exposar-la a Internet. En la seua modalitat gratuïta, els servicis web se suspenden després d'uns 15 minuts sense trànsit, per la qual cosa la primera petició després de la inactivitat pot tardar uns segons a respondre (arrancada en fred).

Si ho comparem amb un VPS (*Servidor Privat Virtual*), Render ofereix una sèrie d'avantatges:

- **Menor complexitat operativa:** no hem d'instal·lar ni mantindre el sistema operatiu, les dependències del servidor ni les actualitzacions de seguretat.
- **Certificats SSL automàtics:** la plataforma genera i renova els certificats HTTPS de manera transparent sense que hàgem de configurar res.
- **Gestió senzilla de variables d'entorn:** podem definir claus secretes, ports o cadenes de connexió a la base de dades directament des del panell de control web.
- **Escalabilitat immediata:** si la nostra aplicació necessita més recursos (més memòria RAM o més instàncies per a repartir la càrrega), podem augmentar-los amb un simple clic.

En canvi, en un VPS tenim un control total del sistema i l'entorn, però hem d'encarregar-nos d'instal·lar i administrar tot: sistema operatiu, servidors, connexions remotes, etc.

A més, una de les característiques més potents de les plataformes *PaaS* modernes és la seua **integració directa amb repositoris** de GitHub o GitLab. Això ens permet implementar un flux de treball automatitzat i continu:

- **Detecció de canvis:** connectem el nostre repositori a la plataforma. Cada vegada que fem un `git push` a la branca principal (*main*), el *PaaS* detecta automàticament la nova versió.
- **Fase de construcció:** la plataforma descarrega el nostre codi, instal·la les dependències necessàries (per exemple, executant `npm install`) i prepara els arxius per a producció.
- **Fase d'arrancada:** la plataforma inicia el nostre servidor executant el comando configurat (per exemple, `npm run start` o `node server.js`).
- **Desplegament sense caigudes:** el nou contenidor o versió de l'aplicació s'arranca en segon pla. Fins que no comprova que respon correctament, la plataforma no substitueix la versió anterior, evitant que els usuaris experimenten caigudes del servici durant l'actualització.

2.2. Preparant el projecte

Prendrem com a base el projecte *ContactesREST_v2* que es va desenrotllar en [este document](#). En ell vam construir una API REST per a gestionar un model senzill de contactes.

2.2.1. Canvis respecte al projecte Node local

[Ací](#) pots descarregar el codi del projecte que usarem, des d'un repositori GitHub. Si li dones una ullada, hi ha alguns xicotets canvis sobre l'original d'aquell document:

- Un canvi important és que s'utilitza un **fitxer d'entorn** per a emmagatzemar valors de configuració, com ara la URL de connexió a la base de dades, o el port pel qual llançar l'aplicació. En l'arrel del projecte veuràs un fitxer `.env.example` amb dades de prova. Hauràs de copiar-ho a un altre anomenat `.env`

i canviar en ell les dades pels de la teua pròpia connexió. Per a gestionar les variables d'entorn instal·lem la llibreria `dotenv` des de `package.json`, juntament amb `express` i `mongoose`.

```
PORT=3000
MONGODB_URI=mongodb+srv://usuari:password@url_cluster/nomBD?parametres
```

- Un altre canvi a destacar és la inclusió d'un **fitxer `.gitignore`** per a indicar què no volem que es pugui al repositori GitHub que crearem. En el nostre cas, descartarem tant la carpeta `node_modules` com el fitxer `.env`, que pot tindre dades sensibles per a proves en local (només es puja `.env.example`, amb dades de prova inofensives).

```
node_modules/
.env
```

- També trobaràs un **fitxer `db_seed.js`** en l'arrel del projecte, que conté codi per a inserir en la base de dades una sèrie de contactes, restaurants i mascotes de prova, perquè no estiga buida. Este fitxer l'executarem després per a poblar la base de dades a l'inici.
- Addicionalment, s'han afegit uns **scripts** en la secció `scripts` de `package.json` per a llançar l'aplicació amb el comando `npm` (encara que també es pot llançar amb `node index.js`). Des de Render es podrà triar el comando a executar per a posar-la en marxa.

```
"scripts": {
  "start": "node index.js",
  "dev": "node --watch index.js",
  "seed": "node db_seed.js"
},
```

2.2.2. Accés a MongoDB Atlas

Com a segon pas important haurem de tindre un compte creat en [MongoDB Atlas](#), i haver creat un clúster seguint els passos indicats en [este document](#).

En el fitxer `.env` del nostre projecte hem de tindre emmagatzemada la URL de connexió a la base de dades, amb el format ja conegut de `mongodb+srv://usuari:password@url_cluster/nomBD?parametres`. Dins dels paràmetres, poden vindre bé un parell d'ells:

```
mongodb+srv://usuari:password@url_cluster/nomBD?retryWrites=true&w=majority
```

- El paràmetre `retryWrites=true` permet que, si una operació d'escriptura (insert, update, delete) falla per un error temporal de xarxa o una reorganització del clúster (per exemple, un canvi de node principal), el driver de Node.js la reintente automàticament una vegada abans de llançar un error a la teua aplicació. Evita que l'API falle davant talls momentanis de connexió.
- El paràmetre `w=majority` estableix el nivell de confirmació d'escriptura. En MongoDB Atlas les dades es repliquen en diversos servidors (un conjunt de rèpliques, usualment 3 nodes). Amb este paràmetre indiquem a MongoDB que no retorne la confirmació d'èxit a l'API fins que la dada haja sigut escrit físicament en la majoria dels nodes del clúster (almenys 2 dels 3). Garantix que les dades no es perden si un dels servidors patix una caiguda just després de rebre la petició.

2.3. Configuració de Render i repositori GitHub

Ara que ja tenim configurat el projecte Node de manera adequada, i el nostre *clúster* en MongoDB Atlas, configurarem un repositori GitHub on allotjarem el nostre projecte, i el sincronitzarem amb Render perquè desplegue l'aplicació.

2.3.1. Creació del repositori en GitHub

Crearem el nostre repositori públic en GitHub on emmagatzemar este projecte. Suposarem, per a este pas, que tens compte creat en [GitHub](#) i que has instal·lat la ferramenta [Git](#) en el teu sistema. Si no és així, completa estos dos passos abans de continuar.

El primer que farem serà inicialitzar un repositori local en la nostra carpeta de projecte Node. Podem fer-ho des del terminal integrat en VS Code, dins del projecte, escrivint estos comandos:

```
# Inicialitzem repositori
git init

# Afegim tot el contingut inicialment (el fitxer .gitignore exclou el que correspon)
git add .
git commit -m "Commit inicial: projecte Express i Mongoose configurat"

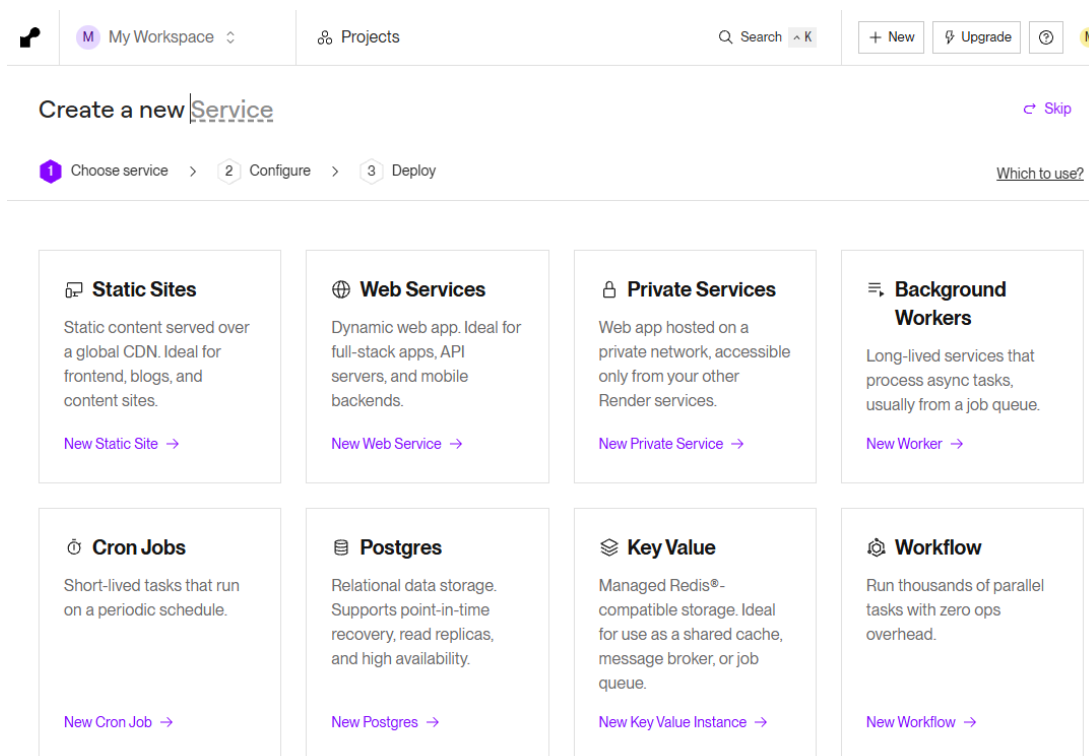
# Canviem de nom la branca principal a "main" (opcional, encara que recomanable)
git branch -M main
```

Ara hem d'accedir a [GitHub](#), crear un nou repositori buit (podem anomenar-ho com vulguem. Per exemple, `ContactesREST`) i quedar-nos amb la seua adreça (per exemple, `https://github.com/el-teu-usuari/contactesrest.git`). Ara connectarem el nostre repositori local amb eixe repositori GitHub:

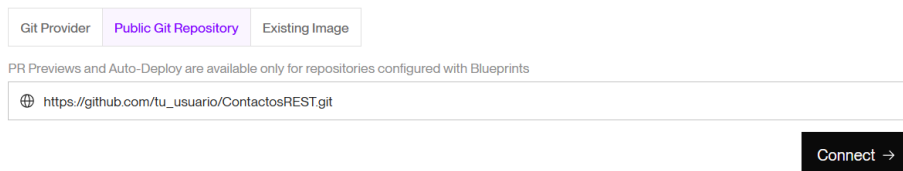
```
git remote add origin https://github.com/el-teu-usuari/el-teu-repositori.git
git push -o origin main
```

2.3.2. Creació de projecte en Render i connexió amb GitHub

Com a últim pas, crearem un nou projecte en el nostre compte de [Render](#) (si no tens compte, crea un). Des del nostre panell (*Dashboard*) triem crear un nou *Web Service*.



Després connectem amb el repositori en qüestió



El següent pas és configurar el servici. En el formulari ens demanen que posem un nom, el llenguatge o framework de desenvolupament (Node), la branca principal del repositori, la regió on desplegar-lo...

Name A unique name for your web service.	ContactosREST
Language Choose the runtime environment for this service.	Node
Branch The Git branch to build and deploy.	main
Region Your services in the same region can communicate over a private network .	Frankfurt (EU Central)
Root Directory Optional If set, Render runs commands from this directory instead of the repository root. Additionally, code changes outside of this directory do not trigger an auto-deploy. Most commonly used with a monorepo .	e.g. src

També indiquem el comando per a construir el projecte (en el nostre cas, `npm install`) i per a posar-lo en marxa (podem triar `node index.js` o bé `npm run start` dels nostres *scripts*). Finalment, podem especificar la tarifa (gratuïta en el nostre cas) i les variables d'entorn que s'utilitzaran. Podem afegir-les a mà o bé copiar-les/pegar-les del nostre fitxer `.env` (recorda que este fitxer NO es puja a GitHub).

Compute For more power and to get the most out of Render, we recommend using one of our paid compute plans. All paid compute plans support: - Zero Downtime - SSH Access - Scaling - One-off jobs - Support for persistent disks	Selected: \$0 / month 0.1 CPU 512 MB RAM <i>Free instances spin down after periods of inactivity. They do not</i> <table> <tr> <td>☒</td> <td>\$0 / month</td> <td>0.1 CPU</td> </tr> <tr> <td>☐</td> <td>\$7 / month</td> <td>0.5 CPU</td> </tr> <tr> <td>☐</td> <td>\$25 / month</td> <td>1 CPU</td> </tr> <tr> <td>☐</td> <td>\$85 / month</td> <td>2 CPU</td> </tr> </table> Show all 15 compute plans	☒	\$0 / month	0.1 CPU	☐	\$7 / month	0.5 CPU	☐	\$25 / month	1 CPU	☐	\$85 / month	2 CPU
☒	\$0 / month	0.1 CPU											
☐	\$7 / month	0.5 CPU											
☐	\$25 / month	1 CPU											
☐	\$85 / month	2 CPU											
Environment Variables Set environment-specific config and secrets (such as API keys), then read those values from your code. Learn more .													
<table> <thead> <tr> <th>KEY</th> <th>VALUE</th> </tr> </thead> <tbody> <tr> <td>MONGODB_URI</td> <td>value</td> </tr> </tbody> </table>	KEY	VALUE	MONGODB_URI	value									
KEY	VALUE												
MONGODB_URI	value												

En la secció d'opcions avançades podem especificar algun detall més. Per exemple (per a opcions de pagament) podem indicar un *pre-deploy Command*, que vindria bé per a executar `npm run seed` i poblar la base de dades a l'inici. Com està deshabilitat per a l'opció gratuïta, haurem d'executar este comando des del nostre ordinador local per a emplenar la base de dades de MongoDB Atlas.

Una vegada emplenades les opcions de configuració, podem desplegar el servici i, si tot és correcte, després d'uns segons per a fer el *build* i instal·lar les llibreries necessàries, el tindrem disponible en una direcció per defecte dins del domini `xxxxx.onrender.com`, on `xxxxx` serà el nom que li hàgem donat al projecte en Render.

2.4. Alternatives a Render

Actualment existixen moltes plataformes *PaaS* i, encara que totes ens permeten connectar un repositori de GitHub per a automatitzar el desplegament, cada una està pensada per a resoldre necessitats distintes, des de frontends estàtics optimitzats en CDN fins a backends persistents i bases de dades gestionades. A continuació nomenem algunes amb les seues particulars característiques.

- **Vercel**: és la plataforma de referència per a desplegar projectes frontend moderns i aplicacions creades amb frameworks de renderitzat híbrid. En ella podem desplegar tant aplicacions *frontend* fetes amb frameworks moderns (React, Vue, Astro...) com a arquitectures *serverless*. No obstant això, no està dissenyada per a allotjar servidors backend tradicionals d'execució contínua.
- **Netlify**: plataforma que va popularitzar el concepte Jamstack (JavaScript, APIs i Markup). Igual que Vercel, està molt enfocada en la capa frontend. Permet desplegar llocs web estàtics, aplicacions SPA (*Single Page Applications*) i funcions *serverless*.
- **Railway**: dissenyada per a desplegar arquitectures full-stack i microservicis mitjançant una interfície visual molt intuïtiva. Permet desplegar backends en qualsevol llenguatge, contenidors Docker i múltiples bases de dades amb un sol clic (PostgreSQL, MySQL, Redis i MongoDB). No obstant això, el seu model d'ús està més enfocat al consum per recursos que a un pla gratuït tradicional permanent.

Ací veiem una anàlisi comparativa de les plataformes comentades:

Resumen técnico de características y selección de herramientas

Característica	Render	Vercel	Netlify	Railway
Tipo de proyecto ideal	Backends y Full-Stack	Frontends y Next.js	Frontends y Jamstack	Microservicios y Full-Stack
Servidores persistentes (Express)	SÍ (Nativo)	NO (Serverless)	NO (Serverless)	SÍ (Nativo)
Bases de datos integradas	PostgreSQL y Redis	Integración externa	Integración externa	PostgreSQL, MySQL, Mongo, Redis
Soporte para Docker	SÍ (Nativo)	NO	NO	SÍ (Nixpacks / Docker)
WebSockets / Conexión continua	SÍ (Soportado)	LIMITADO (Serverless)	LIMITADO (Serverless)	SÍ (Soportado)
Despliegues automáticos con Git	SÍ (Build & Deploy)	SÍ (Deploy Previews)	SÍ (Deploy Previews)	SÍ (Entornos efímeros)
Plan gratuito para aprender	SÍ (con suspensión)	SÍ (muy generoso)	SÍ (muy generoso)	Créditos de prueba