Administració de serveis web

PID_00275597

Alberto José Mateos Bartolomé
Joan Ramon Esteban Grifoll
Javier Panadero Martínez

Temps mínim de dedicació recomanat: 5 hores

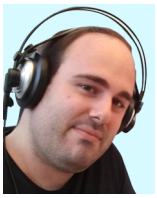
Universitat Oberta de Catalunya

Alberto José Mateos Bartolomé

Enginyer tècnic en Informàtica de Sistemes i enginyer superior en Informàtica per la Universitat Politècnica de Catalunya (UPC). Des de 1999, col·labora amb la Universitat Oberta de Catalunya (UOC), inicialment creant la part pràctica de l'assignatura Xarxes de computadors i, posteriorment, fent les tasques de consultor del laboratori de Sistemes operatius i de l'assignatura Administració de xarxes i sistemes operatius. També ha exercit de tribunal i consultor del TFG en aquest àmbit. Actualment, treballa a l'Escola de Telecomunicacions de la UPC de Barcelona com a analista i desenvolupador de projectes, especialitzat en gestió i millora de processos, metodologies àgils, llenguatges d'entorn web i solucions basades en microserveis i contenidors.

Joan Ramon Esteban Grifoll

Diplomat en Enginyeria Tècnica Industrial en l'especialitat d'Electrònica i Automàtica per la Universitat Politècnica de Catalunya (UPC). Des de 2008, està vinculat a la Universitat Oberta de Catalunya (UOC) com a professor col·laborador en el grau d'Enginyeria Informàtica, impartint docència en l'assignatura d'Administració de xarxes i sistemes operatius (AXSO). Forma part de l'equip de consultors de l'assignatura AXSO i també ha exercit de tribunal i consultor del TFG en aquest àmbit. Amb més de vint anys d'experiència en l'àrea d'informàtica en les especialitats de sistemes operatius, seguretat, comunicacions i programació com a personal de la UPC. Actualment, exerceix com a analista i programador d'aplicacions web a l'Escola Tècnica Superior d'Enginyeria de Telecomunicació de la UPC (Barcelona). Especialitzat en la gestió i millora de processos i la programació d'aplicacions web dinàmiques i adaptables als dispositius mòbils. Especialista en col·laboració i aplicació de metodologies àgils per al desenvolupament en equip (Scrum, Kanban). Defensor del programari i maquinari lliure, i amb coneixements i experiència sobre disseny i implantació de sistemes IoT.



Javier Panadero Martínez

Enginyer informàtic i doctor en Computació d'Altes Prestacions per la Universitat Autònoma de Barcelona (UAB). Des de 2019, és professor dels Estudis d'Informàtica, Multimèdia i Telecomunicació de la Universitat Oberta de Catalunya (UOC). Director del màster universitari en Enginyeria Computacional i Matemàtica. Ha elaborat diversos materials sobre administració de sistemes i programació. Els seus interessos de recerca inclouen la computació paral·lela i distribuïda, l'optimització i simulació de sistemes complexos i els algorismes intel·ligents.

Primera edició: setembre 2020

Av. Tibidabo, 39-43, 08035 Barcelona
Autoria: Alberto José Mateos Bartolomé, Joan Ramon Esteban Grifoll, Javier Panadero Martínez
Producció: FUOC

Índex

Introducció

En aquest mòdul veurem aspectes relacionats amb els serveis web i les tecnologies vinculades a aquests serveis. Els serveis web són aplicacions de client i servidor que es comuniquen per mitjà del protocol de transferència d'hipertext (HTTP) del World Wide Web (WWW). Tal com descriu el World Wide Web Consortium (W3C), els serveis web proporcionen un mitjà estàndard d'interoperabilitat entre les aplicacions de programari que s'executen en diverses plataformes i marcs de referència.

Per iniciar-nos en la temàtica, començarem veient què és un web, com funciona i quins tipus de protocols i arquitectura podem trobar.

Parlarem de les diferents tasques i responsabilitats que hauria de tenir la figura del *webmaster* o administrador web dins l'organització per tal de garantir uns mínims de qualitat a l'hora d'oferir els serveis.

Seguidament, ens endinsarem en aspectes més tècnics sobre l'arquitectura i el disseny de les aplicacions web que, tot i que pot semblar que sigui un aspecte més de programació, també és important conèixer com a administrador de sistemes per tal de poder proporcionar i dimensionar els recursos necessaris i poder protegir, de forma correcta, la infraestructura de l'organització a l'hora d'oferir aplicacions web.

També explicarem què s'entén per servei i de què consta una arquitectura orientada al servei (SOA). Analitzarem en detall totes les capes d'abstracció que disposa aquesta arquitectura i veurem el mecanisme de *web services* (serveis web) que és una de les tecnologies que fa possible implementar una arquitectura orientada al servei.

Finalment, explicarem el que molts consideren una evolució de l'arquitectura SOA: l'arquitectura de microserveis. Aquesta arquitectura implementa un model basat en petits serveis que es comuniquen de forma molt superficial entre ells. Cada microservei executa els seus propis processos i es desplega de forma independent, normalment amb mecanismes automatitzats. Com a peça clau d'aquesta arquitectura, explicarem els contenidors i la plataforma Docker, que s'ha fet molt popular al llarg dels darrers anys.

Per acabar amb el tema dels serveis, també veurem una metodologia de treball molt coneguda anomenada DevOps. L'objectiu principal que es busca amb aquesta metodologia és aconseguir productes millors i més fiables. La idea es

basa a unificar els dos rols que fins ara treballaven aïllats (desenvolupament i operacions) perquè es coordinin i col·laborin fent desplegaments en producció més ràpids i freqüents.

Objectius

Els materials d'aquest mòdul contenen les eines necessàries per tal que l'estudiant assoleixi els objectius següents:

1. Adquirir un coneixement dels principals protocols i tecnologies web.

2. Adquirir la capacitat de dissenyar i implementar infraestructures client/servidor de forma òptima.

3. Conèixer i identificar les tasques i responsabilitats com a administrador en la gestió dels serveis web.

4. Adquirir la capacitat de determinar l'arquitectura i les tecnologies més apropiades per a una determinada aplicació/servei web.

5. Conèixer els patrons de disseny web més adequats per a cada cas.

6. Implementar una arquitectura orientada al servei (SOA) i conèixer exemples dels protocols que es fan servir per desenvolupar-la.

7. Saber què són els microserveis i com els contenidors han ajudat a fer possible aquesta nova arquitectura.

8. Saber com aplicar la metodologia DevOps en una organització amb l'objectiu d'aconseguir productes millors i més fiables.

1. Administració web

1.1. Introducció: què és un servidor web?

Durant la dècada dels noranta, al CERN,[1] Tim Berners-Lee va presentar una proposta per a un sistema de gestió de la informació que permetia compartir coneixements i recursos per mitjà d'una xarxa informàtica. En concret, va proposar utilitzar l'hipertext per enllaçar i accedir a informació de diversos tipus com una xarxa de nodes on l'usuari podia navegar a voluntat. Aquest tipus de compartició d'informació es va anar propagant al que realment es pot dir una World Wide Web, ja que les persones de tot el món l'usen per a una àmplia varietat de propòsits. Originalment es va promoure la World Wide Web com una biblioteca virtual, un sistema de control de documents que permetia compartir recursos d'informació entre els investigadors. L'accés als documents en línia es feia mitjançant una direcció de document única, un localitzador universal de recursos (URL). Aquests documents podien ser referenciats per mitjà d'enllaços d'hipertext, com comentàvem anteriorment.

(1)Laboratori de recerca científica a prop de Ginebra, Suïssa.

L'**hipertext** és un conjunt estructurat de textos, gràfics, etc., units entre si per enllaços i connexions lògiques.

La tecnologia bàsica necessària per tenir una web és relativament simple. Una computadora connectada a internet, amb un servidor web, és el mínim per lliurar documents. Les primeres pàgines web es basaven en un conjunt d'enllaços organitzats de forma jeràrquica i accessible des d'una pàgina d'inici o *webpage*. Al conjunt de pàgines accessible ja se'l considerava com un lloc web (*website*). D'aquí es van establir les primeres diferències entre una pàgina web i un lloc web. Un lloc web és un grup de pàgines web que estan connectades entre si per mitjà d'enllaços d'hipertext. A mesura que anaven creixent els llocs web, es feia més necessari organitzar el seu contingut tant en disseny com en interconnexions entre les pàgines per fer-les comprensibles i de fàcil navegació. Aquesta informació requeria una edició i actualització de continguts inicialment de manera manual, la qual cosa es denomina *continguts estàtics*. A poc a poc, a causa de la complexitat que comportava el manteniment d'aquests continguts, van anar apareixent les primeres tecnologies que permetien la generació de continguts i mostraven la informació continguda en bases de dades, motors de cerca o, fins i tot, altres pàgines web. A aquest tipus de continguts se'ls va passar a denominar *continguts dinàmics*. Posteriorment, aquests continguts dinàmics han anat adquirint més personalització en funció

del client que es connectava sobre la base d'uns paràmetres de sol·licitud. Així doncs, dels denominats *llocs web* es van passar a la creació de les denominades *aplicacions web*.

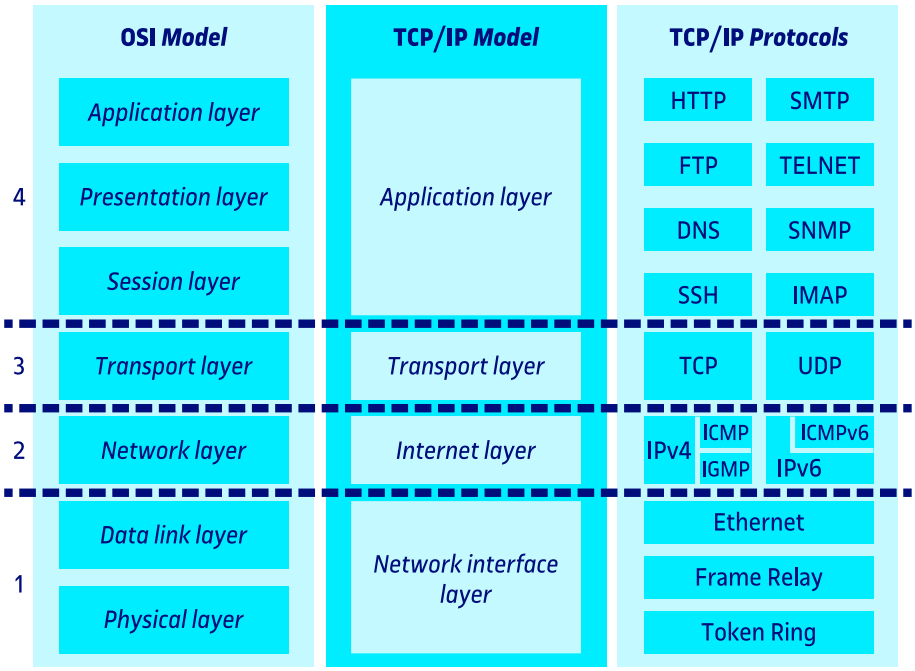
Una aplicació web és una aplicació o eina que presenta continguts adaptats de manera dinàmica sobre la base d'uns paràmetres de sol·licitud, accessible per mitjà d'un client com un navegador web i proporcionats per un servidor mitjançant un servei web normalment per mitjà d'internet.

De la mateixa forma que ha anat evolucionant la manera de mostrar la informació mitjançant pàgines web, llocs web i aplicacions web, també va anar evolucionant la manera de crear els continguts des de l'edició bàsica i manual d'una pàgina, al sistema de gestió de continguts o CMS (*content management system*) i, fins i tot, a l'ús de *frameworks* i llibreries de llenguatges de programació per crear aplicacions. Tot això ho veurem amb més detall en els diferents apartats d'aquest mòdul.

1.2. El protocol

Per entendre què és una web també hem d'entendre com és el seu funcionament tecnològic. La tecnologia web es basa en el conjunt de protocols TCP/IP (*transmission control protocol*, *internet protocol*), el model de funcionament de la qual es basa en quatre capes de forma anàloga al model OSI, tal com es pot veure a la figura 1.

Figura 1. Model OSI respecte al model TCP/IP



Vegem, ràpidament, quin és el significat de cadascuna de les capes:

1) Capa interfície de xarxa: és la capa de nivell de més a sota encarregada de la transmissió de les dades TCP/IP al nivell físic de la xarxa. És independent de qualsevol tecnologia de xarxa, la qual cosa permet connectar diferents tecnologies de xarxa com ara Token Ring, Ethernet, Frame Relay, etc.

2) Capa d'internet: és la responsable de la comunicació entre els sistemes mitjançant funcions d'adreçament, empaquetatge i encaminament del *host*. En aquesta capa s'agrega la capçalera dels paquets, la qual cosa es coneix com a adreça IP. Aquesta pot ser de 32 bits (IPv4) o 128 bits (IPv6). Dins de la capa d'internet, podem trobar els protocols IP (*internet protocol*), ICMP (*internet control message protocol*), IGMP (*internet group management protocol*) i ARP (*address resolution protocol*).

3) Capa de transport: així com la capa anterior és l'encarregada de la comunicació entre dos equips, la capa de transport és la que permet que les aplicacions (programa, tasca, procés, etc.) dels equips remots puguin comunicar-se. Per això, disposa de dos protocol principals que permeten l'intercanvi d'informació independentment del tipus de xarxa: TCP i UDP. El TCP (*transmission control protocol*) és un protocol fiable amb detecció d'errors i orientat a la connexió. En canvi, UDP (*user datagram protocol*) és un protocol sense connexió ni detecció d'errors però molt útil en el cas de petites transmissions que no requereixen la complexitat del protocol TCP en què preval el rendiment i la rapidesa a la fiabilitat, per exemple en transmissions d'àudio/vídeo. Per facilitar la comunicació i la identificació de les aplicacions remotes, s'utilitza un sistema d'enumeració per associar aquestes aplicacions, la qual cosa es coneix com a ports. Algun dels ports més coneguts són: 25 (SMTP), 53 (DNS), 80 (HTTP), etc.

4) Capa d'aplicació: és la capa de nivell més alt i la que permet a les aplicacions accedir als serveis de les altres capes mitjançant la definició dels protocols a utilitzar en la capa de transport, sigui TCP o UDP. Algun dels protocols més coneguts són: HTTP (*hypertext transfer protocol*), FTP (*file transfer protocol*), SMTP (*simple mail transfer protocol*), DNS (*domain name service*), etc.

HTTP és el protocol utilitzat a la capa d'aplicació per al servei web i utilitza el protocol subjacent TCP de la capa de transport per transmetre la informació. L'HTTP basa la seva comunicació en el concepte de sol·licitud/resposta, la qual cosa significa que un programa client HTTP envia un missatge de sol·licitud HTTP a un servidor HTTP, que retorna un missatge de resposta HTTP, del qual parlarem amb més detall més endavant en aquest mateix apartat. Per realitzar aquesta comunicació es va dissenyar un esquema que defineix la sintaxi de funcionament i la identificació del recurs al qual desitja accedir. A aquest esquema o seqüència de caràcters se'l denomina: URL (*uniform resource locator*), en què es defineix cadascuna de les parts de la connexió.

```
scheme://host[:port#]/path/. . ./[?query-string][#*anchor]
```

1) L'***scheme*** defineix el protocol de la capa d'aplicació que s'usarà. Per al cas d'un servidor web s'usarà HTTP o, en la majoria dels casos, HTTPS (HTTP sobre SSL, *secure sockets layer*). Com a esquema també és típic usar altres protocols com ara: ftp, mailto, ldap, file, etc.

2) El ***host*** és el nom o l'adreça IP del servidor web al qual es desitja accedir.

3) El **port** fa referència a la numeració que identifica el servei en la capa de transport i es correspon amb el port d'escolta del servei de destinació. En el cas de la web, aquest port és el 80 per al cas d'HTTP i el 443 per al cas d'HTTPS. Aquest valor sol obviar-se, atès que és establert de forma intrínseca quan s'indica el tipus d'esquema, ja que són valors definits per l'estàndard TCP. Sol establir-se quan s'usen ports alternatius als estàndard.

4) El ***path*** o ruta és el camí que defineix la ubicació del document a partir d'un element «arrel» («/») del servidor fins al document desitjat. Aquesta ruta pot coincidir amb el sistema d'arxius del servidor o, com en el cas de la web, pot ser representada per diferents àlies.

5) La ***query-string*** o cadena de consulta defineix els paràmetres que permeten obtenir una resposta dinàmica sobre la base d'unes condicions. La manera d'indicar els paràmetres és mitjançant un signe d'interrogació «?», seguit del nom del paràmetre i el seu valor separat tots dos per un signe «=». Es poden indicar diversos paràmetres amb els seus valors separant-los mitjançant el signe d'unió «&». Sol utilitzar-se per realitzar sol·licituds que permetin obtenir resultats com ara filtres o cerques.

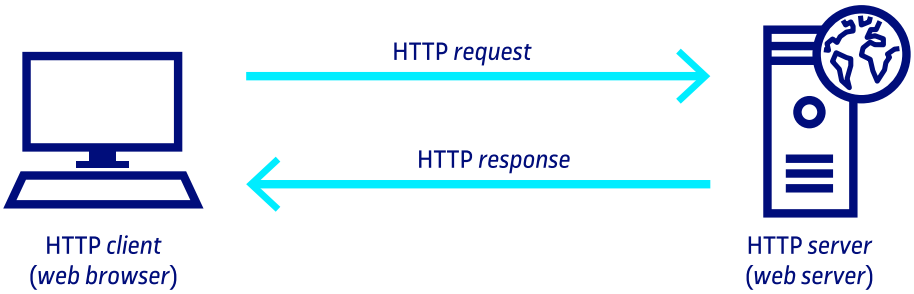
6) L'***anchor*** o àncora indica la referència d'un marcador posicional dins del document sol·licitat. Si s'utilitza, la visualització es desplaça de manera automàtica fins a aquesta posició del document.

1.3. L'arquitectura

La comunicació entre dos equips es realitza mitjançant el paradigma client/servidor usant els protocols HTTP/HTTPS, en què el client sol ser un navegador web o client HTTP i el servidor un equip amb un servei HTTP en execució. El servidor web administra i proporciona accés a un conjunt de recursos. Els recursos poden ser simples arxivaments de text i imatges, o quelcom més complex, com ara una base de dades relacional. Els clients, també coneguts com a agents d'usuari, inicien una transacció enviant una **sol·licitud** a un servidor. El servidor després processa la sol·licitud i envia una **resposta** al client. Aquesta es podria dir que és l'estructura bàsica de qualsevol servei que es desitgi proporcionar i és la que està representada a la figura 2. El client més habitual sol ser el denominat navegador, com els coneguts: Firefox, Edge, Chrome, Sa-

fari, etc., encarregats de mostrar el contingut i interactuar-hi. També es poden trobar en format d'ordres com el lynx o wget. Del costat servidor solem trobar com a programes més habituals: Apache, Nginx, Tomcat, NodeJS, etc.

Figura 2. Comunicació HTTP client/servidor



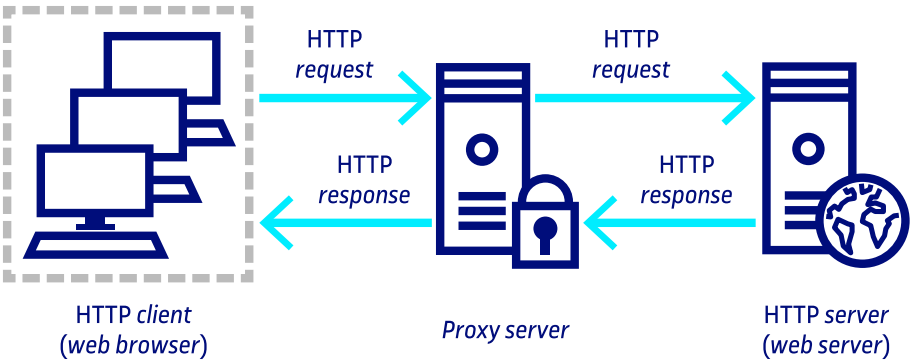
1.3.1. Servidor *proxy*

Quant a l'estructura dels serveis web, una de les variants que es pot trobar és mitjançant la introducció dels denominats servidors *proxy* (vegeu figura 3). Els servidors *proxy* són programes que actuen com a clients i servidor al mateix temps i se situen entre el client HTTP i el servidor HTTP final. El seu funcionament és força bàsic: envia totes les peticions rebudes pels clients HTTP al servidor HTTP. L'objectiu d'aquest tipus de servidors són bàsicament dos:

1) Millora del rendiment: els servidors *proxy* emmagatzemen durant un temps les peticions realitzades pels clients al servidor web, en el que es denomina *memòria cache*. D'aquesta forma, cada vegada que es repeteixi una mateixa petició, aquesta no anirà al servidor HTTP final, sinó que serà el mateix servidor *proxy* qui retorni la informació ja emmagatzemada en la memòria.

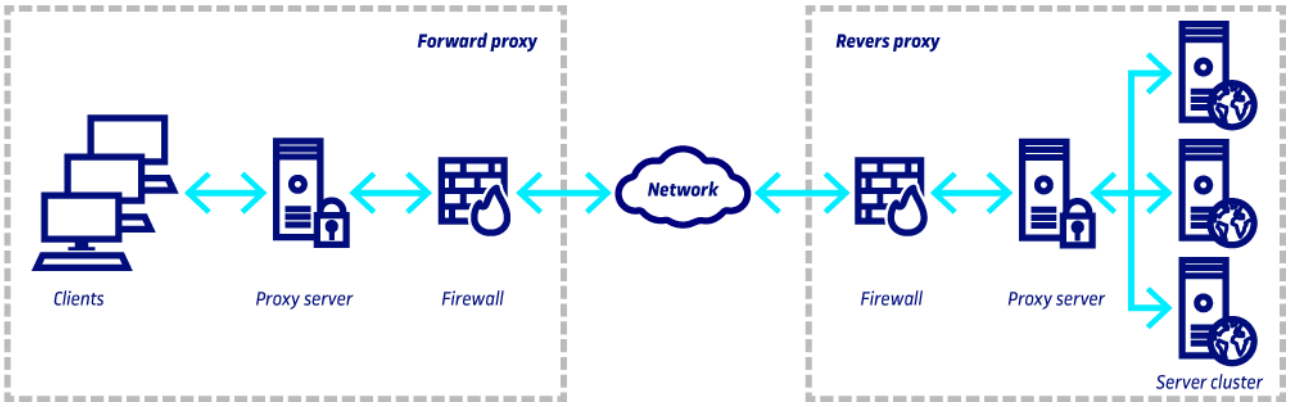
2) Filtrar peticions: un servidor *proxy* també permet filtrar les peticions realitzades pels clients per impedir l'accés a llocs específics. Això pot ser útil, per exemple, a les organitzacions on es vulgui restringir l'accés a llocs d'internet no desitjats.

Figura 3. Servidor *proxy*



Dins de les estructures web amb servidors *proxy* podem trobar dos models d'implementació en funció de la posició on estigui: ***proxy* directe** (*forward proxy*) o ***proxy* invers** (*reverse proxy*). A la figura 4 es pot veure el tipus d'infraestructura i de connexió en funció del tipus de servidor *proxy* implementat.

Figura 4. *Proxy* directe contra *proxy* invers



1) *Proxy* directe: és quan el servidor està entre el client i l'accés a internet. Quan els clients realitzen sol·licituds a llocs i serveis d'internet, el servidor *proxy* intercepta aquestes sol·licituds i després es comunica amb els servidors en nom d'aquests clients, com a intermediari. El seu ús està destinat a:

- Reduir el trànsit a l'exterior mitjançant l'ús de *cache*.
- Bloquejar l'accés a certs continguts.
- Protegir la identitat dels clients.

2) *Proxy* invers: és quan el servidor està entre les sol·licituds entrants d'internet i els recursos dels servidors en una xarxa privada. L'objectiu d'un *reverse proxy* és actuar d'intermediari entre el client extern i els serveis que s'ofereixen. Qualsevol petició de servei és interceptada pel servidor *proxy* que, al seu torn, realitza aquesta petició, si cal, al servidor original i dona la resposta al client com si fos el servidor original. El seu ús està destinat a:

- Reduir la càrrega dels servidors mitjançant l'ús de *cache*. Sol usar-se per servir contingut estàtic. Totes les peticions d'aquest tipus de contingut són proporcionades directament pel servidor *proxy* sense necessitat d'accés al servidor original.

- Protegir l'accés mitjançant connexions HTTPS. Es disposa d'un conjunt de servidors que ofereixen el mateix servei per a la distribució de la càrrega, sol configurar-se el servidor *proxy* d'entrada amb el protocol HTTPS, en comptes d'haver de fer-ho de manera individual.

- Ocultar l'adreça IP real dels servidors. Això permet un nivell més de seguretat per evitar possibles atacs directes als servidors finals.

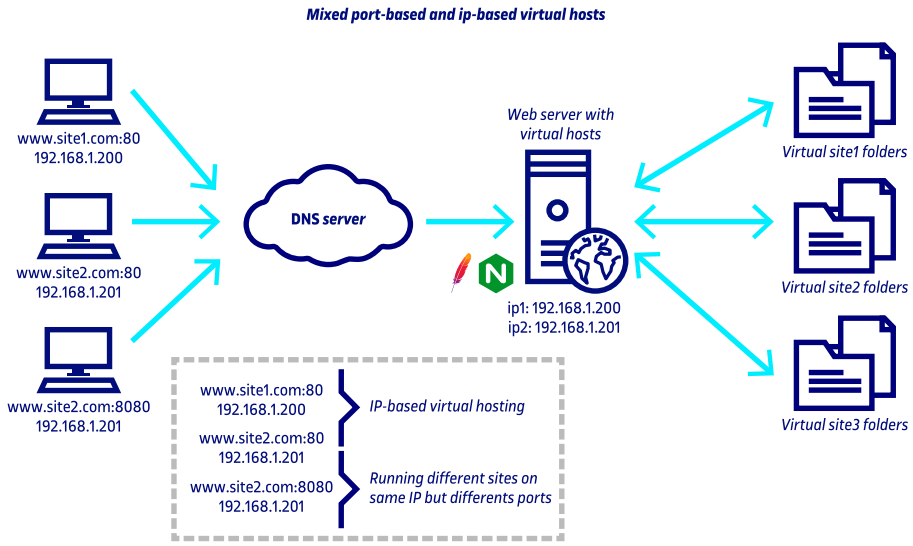
- Pot actuar com a balancejador de la càrrega (*load-balancer*). Encara que no és la seva missió principal, alguns servidors *proxy* compten amb la capacitat de balanceig de la càrrega distribuint les peticions entrants al clúster de servidors que proveeixen el mateix tipus de servei.

1.3.2. Allotjament virtual (*virtual hosting*)

Una altra de les configuracions que se sol implementar a l'hora de servir pàgines web és l'anomenat allotjament virtual o *virtual hosting*. Aquesta metodologia permet allotjar múltiples llocs web (per exemple, el lloc www.site1.com, el lloc www.site2.com, etc.) en una mateixa màquina. Això també implica compartir els recursos de la màquina única com ara la memòria i la CPU. Els recursos es comparteixen i s'utilitzen de manera que s'aconsegueixi la màxima eficiència.

Avui, hi ha diverses formes de configurar un *host* virtual, la majoria de les quals es mostren a continuació:

Figura 5. Estructura allotjament virtual



- **Basat en IP**. És un dels mètodes més senzills i es pot utilitzar per aplicar diferents directives basades en l'adreça IP. A l'allotjament virtual basat en IP, s'utilitzen IP diferents per a cada domini. Aquest allotjament virtual s'aconsegueix mitjançant la creació de diverses adreces IP per al servidor únic. Per exemple, com es veu a la figura 5, el servidor té dues adreces IP (192.168.1.200 i 192.168.1.201) que es resolen respectivament amb els noms www.site1.com i www.site2.com.

- **Basat en ports**. L'allotjament virtual basat en ports és similar a l'allotjament virtual basat en IP. La diferència és que, en lloc d'utilitzar

una adreça IP diferent per a cadascun dels allotjaments virtuals, s'utilitzen diferents ports per a una mateixa IP en què el servidor únic està configurat per respondre a diversos llocs web depenent del port del servidor. Seguint l'exemple de la figura 5, www.site2.com ho resoldrà amb els continguts del lloc virtual 2 o amb els continguts del lloc virtual 3 depenent si el port és el 80 o el 8080, respectivament.

- **Basat en el nom**. L'allotjament virtual basat en el nom és la tècnica d'allotjament virtual més comuna i que s'utilitza més freqüentment. En aquest cas, s'utilitza una adreça IP única per a tots els dominis del servidor. Quan el navegador intenta connectar-se al servidor, envia un missatge al servidor informant sobre el nom del domini al qual intenta connectar-se. Quan es proporciona el nom del domini, el servidor comprova la configuració de l'amfitrió i, per tant, retorna la sol·licitud amb el lloc web correcte.

El servidor web únic pot estar configurat per acceptar una o diverses de les modalitats que s'han comentat o una barreja d'aquestes.

En l'actualitat, un exemple de servidors que accepten aquest tipus de configuració son: Apache (Apache *virtual host documentation* - Apache HTTP *server version* 2.4, 2020) i Nginx (*server block examples*, NGINX, 2020).

2. L'administrador web

L'administrador web o *webmaster* va ser una figura que es va crear per administrar molts dels aspectes que conformen una web: el desenvolupament, el disseny i el manteniment del lloc web. Avui en dia, depenent dels recursos de l'organització, la figura del *webmaster* i les seves tasques s'han anat distribuint entre rols més específics. La necessitat de disposar d'una web ben posicionada als cercadors, atractiva i de ràpid accés, amb continguts actualitzats i accessibles des de dispositius mòbils, fa que es creï la necessitat d'incorporar diferents perfils professionals per tal de facilitar el manteniment dinàmic del lloc web. Això fa que la figura de l'administrador web pugui fer tasques més flexibles com pot ser la coordinació dels equips de desenvolupament, disseny o sistemes per crear projectes o, fins i tot, mantenir els serveis i continguts si fos necessari. Per tant, encara que la figura del *webmaster* sigui creada per a la gestió completa del lloc web, aquesta pot variar en funció de les necessitats de l'organització. Tot i així, la figura del *webmaster* pot continuar existint i les seves tasques i responsabilitats es poden trobar d'entre les que es defineixen en aquest apartat, la qual cosa no vol dir, com s'ha indicat, que aquestes tasques es puguin repartir entre diferents professionals.

Conèixer l'arquitectura de programari del servidor web que s'utilitza és fonamental per al *webmaster*. En general, tots els servidors treballen d'una manera semblant i podem fer un esquema genèric de com són. També són força generals les funcions de l'administrador, encara que és la cultura i la història de l'organització concreta qui marca les tasques i les necessitats reals que ha de cobrir.

El binomi administrador-servidor ha d'estar ben coordinat per obtenir el millor rendiment possible.

L'**administrador** és qui manté el servidor en funcionament, i també qui administra la seguretat i els permisos i, per tant, qui ha de conèixer millor que ningú totes les seves possibilitats.

En concret, l'administrador ha de:

1) Instal·lar i administrar el servidor web. El servidor web és un recurs amb el qual l'organització dona accés a la informació que vol publicar. Aquest servidor web, per exemple, podria ser d'accés restringit o públic, d'accés des d'una xarxa local o des d'internet, adaptat a versions de clients d'escriptori, a dispositius

mòbils o tots dos, o per servir pàgines, llocs o aplicacions. Depenent del servei al qual estigui destinat aquest servidor web, les configuracions i polítiques de disseny variaran.

Un servidor web, des del punt de vista de l'administrador de sistemes, no planteja molts problemes a l'hora d'instal·lar-lo, ja que com a concepte només es tracta de mantenir el servei en funcionament, configurar-lo amb les seves opcions bàsiques com ara indicar on està la informació que s'ha de fer servir i el tipus de seguretat a tenir en compte tant en les connexions TCP/IP entrants com en els permisos dels fitxers. A més, el servidor sol proporcionar eines de control que permeten monitorar i, posteriorment, analitzar el rendiment. Per exemple, és molt comú que un servidor web tingui disponible la quantitat d'accessos simultanis que té i el seu rendiment, i també un registre d'accessos i errors que es puguin anar produint permetent obtenir informació estadística dels tipus de clients, origen i destinació de les connexions, pàgines més visitades, etc., útils per analitzar l'experiència d'usuari i poder aplicar millores en la web en funció d'això.

També serà responsabilitat de l'administrador de sistemes decidir, en funció del servei i el rendiment que calgui oferir, el tipus de servidor més adequat i l'arquitectura que oferirà aquest servei. Podrà ser un servei HTTP únic instal·lat en un servidor, basat en contenidors i microserveis o, fins i tot, un servei al núvol.

2) Conèixer el programari. Hi ha dues classes de programari que l'administrador web ha d'utilitzar. D'una banda, el que fa referència a la instal·lació i configuració del servei que serveix les pàgines web i, de l'altra, l'arquitectura i el disseny de la web en si mateixa. Depenent del tipus d'arquitectura usada, haurem de tenir en compte el dimensionament del servidor i dels recursos necessaris. Tal com veurem en l'apartat següent sobre arquitectures web, podrem trobar-nos amb aplicacions web que s'executen del costat del servidor o del costat del client. Depenent del tipus d'arquitectura triada caldrà un tipus de servidor o un altre i, fins i tot, més o menys recursos. Així doncs, és important prendre consciència dels clients que visitaran la web oferta pel nostre servidor i com volem que la informació es mostri a aquests clients. En cas que el programador hagi desenvolupat una intranet per a la mateixa organització, potser és més fàcil triar el tipus d'arquitectura, ja que es parteix d'un conjunt de clients coneguts i la tecnologia de connexió és comuna a tots ells, per la qual cosa una tecnologia basada en aplicacions del costat del client seria la més idònia i menys costosa.

3) Administrar els directoris. Quan s'ha de dissenyar una nova web, també s'ha de tenir en compte la manera com s'organitzen els diferents recursos. Novament, depenent de l'arquitectura i el disseny de la web, els directoris i els permisos necessaris seran més o menys complexos depenent de la tecnologia i del disseny usat pel programador. El més habitual, i que el mateix programari de desenvolupament de les pàgines ja gestiona, és tenir-los classificats

en carpetes o directoris segons el tipus. És habitual que hi hagi un directori d'imatges o recursos públics al qual totes les pàgines facin referència. Això permet compartir, per exemple, fitxers de logotips de l'organització, ja que no s'han d'incloure a totes les ubicacions on hi hagi documents que els requereixin. Cal ser coneixedor de com està guardada la informació per poder fer còpies de seguretat, per exemple, i també ser capaços de dur a terme migracions o serveis redundants per augmentar la disponibilitat.

2.1. Responsabilitats genèriques de l'administrador

Un servidor web proporciona una porta d'accés a una gran quantitat d'informació d'una manera molt còmoda per al client i, encara que la instal·lació del servidor sol ser senzilla, sí que s'ha de tenir en compte com administrar cadascuna de les parts que el componen i quins recursos són necessaris per garantir un accés continu a la informació, ja que és una responsabilitat que recau en mans de l'administrador.

Podem destacar les responsabilitats de l'administrador següents:

1) Responsabilitat del servidor. Un servidor web no és excessivament complex d'administrar. Com tot servei d'internet que desitgem oferir, s'ha de garantir la seva seguretat, disponibilitat i escalabilitat. Per això, es disposen de les mateixes tècniques i recursos que s'ha vist en mòduls anteriors quant a la necessitat de disposar d'un tallafocs, sistemes d'alimentació ininterrompuda, redundància de serveis, etc. Com que és un servei en què es preveu un nombre important de connexions entrants, caldrà garantir una amplada de banda mínima per evitar la caiguda del servidor per pics importants de connexions. Avui hi ha diferents tècniques que permeten garantir aquesta disponibilitat i controlar el nivell de càrrega, i també aprofitar els recursos de la manera més òptima. Tal com s'ha vist en apartats anteriors, disposem de tècniques com ara el *virtual hosting* (allotjament virtual) que ens permet aprofitar els recursos físics del servidor mitjançant la possibilitat d'assignar múltiples llocs web i noms de domini a una sola adreça IP, o tècniques de *caching* (emmagatzematge en memòria cau) per a la millora en eficiència a l'hora de servir continguts mitjançant l'emmagatzematge en memòria cau o *cache* dels resultats intermedis per augmentar la velocitat en les respostes. Tot i així, la disponibilitat va més enllà de la responsabilitat dels servidors HTTP, ja que continuen sent responsabilitat de les aplicacions o altres tècniques que veurem més endavant com ara l'arquitectura basada en microserveis, que permeten dividir les aplicacions en processos o serveis més petits i independents però que tots junts duen a terme la mateixa tasca, la qual cosa permet gestionar els recursos de forma més òptima.

Vegeu també

Sobre l'allotjament virtual, vegeu el subapartat «Allotjament virtual (*virtual hosting*)».

Sobre l'emmagatzematge en memòria cau, vegeu el subapartat «Servidor *proxy*».

L'**allotjament virtual** és la capacitat d'assignar múltiples servidors i noms de domini a una sola adreça IP.

2) Responsabilitat dels continguts. Pel que fa a l'administració de les pàgines, normalment aquestes queden fora de l'abast de l'administrador web ja que, com veurem més endavant, depenent del tipus de web que s'estigui oferint: pàgines estàtiques, dinàmiques, serveis web, etc., hi haurà un gestor de continguts (CMS) o desenvolupador que les mantingui, sigui amb contingut estàtic o obtingut de manera dinàmica. Així doncs, l'administrador web haurà de donar el suport necessari quant a la gestió dels sistemes i serveis que allotjaran els recursos necessaris per poder oferir aplicacions i serveis web.

És important adonar-se de dos aspectes del contingut en un servei web:

a) Ha d'haver-hi un control de quines dades es proporcionen en cada instant, de manera que no hi hagi continguts antiquats, repetits, sensibles o que no siguin coherents entre ells. Aquesta tasca pot recaure en l'administrador web o, cada vegada més, sobre un responsable de continguts com pot ser un *social manager* per al cas dels continguts de les xarxes socials o *product manager* per al cas de les aplicacions web.

b) En relació amb l'accés a aquestes dades, ha d'haver-hi un control de com es gestiona, consulta i modifica la base de dades a la qual pot estar accedint el servidor web.

Avui en dia, la gestió dels continguts estàtics d'una web es fa mitjançant **gestors de continguts** (CMS).

3) Responsabilitat de l'homogeneïtat. Un aspecte més de disseny que pot recaure en l'administrador web, és el control de la imatge que reflecteix la web. Com que es tracta d'una porta oberta a molts usuaris potencials, que poden no ser coneguts, s'ha de cuidar la forma com es presenta la informació. Així, és important tenir en compte que a totes les pàgines ha d'haver-hi un estil constant. Aquest disseny també pot recaure en un dissenyador que ens faci una maqueta o imatge de com hauria de ser la web i, seguidament, el *webmaster* o programador s'encarregaria d'adaptar-lo als llenguatges web (HTML, CSS). Tal com comentàvem anteriorment, l'ús de CMS també facilita la gestió de continguts en què el mateix CMS incorporaria les seves pròpies plantilles de disseny i solament s'hauria de preocupar dels continguts.

4) Optimització per a motors de cerca. Perquè un lloc web sigui fàcilment localitzable i pugui millorar la seva visibilitat, ha d'estar ben posicionat en els resultats dels diferents motors de cerca. Aquest concepte es pot trobar amb el nom de SEO (*search engine optimization*). Els motors de cerca es basen, bàsicament, en dos factors:

a) Rellevància: la relació del contingut de la pàgina respecte als termes indicats en la cerca. Cal optimitzar la web perquè el motor de cerca sigui capaç d'entendre el contingut mitjançant l'ús de paraules clau o *keywords*, el temps de càrrega, l'experiència de l'usuari, l'optimització del codi i el format de les URL.

b) Autoritat: bàsicament és la popularitat d'una web. Els motors de cerca usen l'experiència d'usuari i el nivell de compartició d'aquest contingut per determinar la utilitat de la web per als usuaris. Com més popular sigui, millor es posicionarà la web.

Evidentment, un bon posicionament del lloc web pot ser molt útil en concepte de màrqueting i publicitat per guanyar clients i, en conseqüència, obtenir més beneficis per a l'organització.

5) Generació d'estadístiques. Segons les necessitats de cada organització, aquesta tasca pot ser més o menys important. En general, cal tenir un registre dels que han visitat la web, tant dels que l'han vist, com quan i des d'on ho han fet. A part, hi ha altres tipus d'estadístiques molt interessants com, per exemple, a quina velocitat s'ofereixen els documents o la quantitat de peticions que hi ha de les diferents pàgines i el temps total que un client es passa navegant per tota la web. La recollida d'aquests indicadors permet analitzar quin és l'ús que s'està donant del lloc web, l'experiència de l'usuari i, per tant, ens permet actuar d'una forma més directa en la millora dels continguts i optimitzar la web per oferir uns continguts més ajustats a les necessitats del client.

2.2. L'administrador i la creació de pàgines

Tal com s'indicava al principi, actualment el paper de l'administrador web sol estar dividit en diferents rols. L'administrador web ha d'assegurar el correcte funcionament de la web, que els aspectes de seguretat siguin els que necessita l'usuari i l'organització, que se segueixin les directrius que marca l'organització i que el contingut global sigui homogeni.

No obstant això, pot ser que no s'encarregui de cuidar els continguts. La tendència és que les pàgines les mantingui directament la mateixa font d'informació. Per això, es proporcionen eines i formació a les persones que han d'encarregar-se d'aquests continguts. Per tant, l'administrador web ha de poder tenir dues classes d'usuaris:

1) Consumidors del material de la web. Els consumidors del material de la web són els usuaris finals, els que es connecten per mitjà d'un navegador i consulten la web per obtenir informació sobre l'organització o tenir accés als serveis que s'ofereixen.

2) Creadors de contingut. Ja s'ha comentat que, sobre l'administrador web, poden recaure tasques de supervisió o coordinació dels continguts, però normalment no els manté. Pot aconsellar sobre com fer-los, donar plantilles, donar eines, explicar les possibilitats de la web i facilitar els mecanismes del servidor web perquè els creadors del material de la web puguin fer la pàgina de la millor manera possible.

En els seus inicis, el creador de pàgines web era una persona amb un perfil bàsicament artístic o de disseny. No tenia la necessitat de tenir coneixements tècnics, però sí que sabia bé com combinar elements per crear efectes vistosos. El *webmaster* s'encarregava de proporcionar els coneixements dels elements que integren les webs: HTML, Java, JavaScript, etc., per poder adaptar el disseny a les pàgines web.

Podríem considerar que l'administrador web hauria de seguir **pautes** com les que es detallen a continuació:

a) Com que hi ha uns propietaris que tenen la responsabilitat de la informació que el servidor web està servint, se'ls ha d'involucrar en la manera com aquesta informació es presenta. Així, és important que l'administrador s'encarregui d'orientar-los i introduir-los al món de l'HTML, que els ensenyi a utilitzar escàners, a convertir formats de fitxers, a buscar per internet o a utilitzar les propietats del navegador.

b) Per facilitar el manteniment dels continguts estàtics a usuaris sense coneixements tècnics, ajuda molt disposar d'un gestor de continguts (CMS) com es comentava anteriorment. Si la part de disseny recau en el programador, com que aquestes pàgines són dinàmiques, l'ús de llibreries gràfiques és un bon mecanisme per mantenir la homogeneïtat.

c) Cal tenir una política d'ús del servidor web. Així, una vegada s'ha acceptat la política, no hi haurà malentesos en la generació dels nous continguts.

d) Verificar els enllaços. Normalment, a les webs de continguts estàtics, en què la informació es manté de manera manual, és possible que els documents es moguin constantment d'un lloc a un altre fent que l'adreça d'accés canviï i que, per tant, qualsevol referència a aquesta falli. És important que un servidor web no ofereixi enllaços «caducats» o erronis, ja que això afectaria la imatge que l'organització ofereix amb vista a l'exterior.

L'**administrador** és qui promou la utilització dels recursos de la web entre els creadors de pàgines. Els proporciona tots els recursos i coneixements necessaris perquè puguin fer pàgines web i mantenir els continguts.

2.3. Eines d'ajuda per a la generació de contingut

La importància de l'ús de la web com a mitjà de comunicació i promoció per a les corporacions fa que es creï la necessitat de disposar d'un departament o recursos destinats a dur a terme la gestió i manteniment d'aquesta web. Per això, s'ha de disposar d'un administrador web i de personal amb amplis coneixements tecnològics, tant en la part de sistemes encarregada de l'administració del servei com per al desenvolupament de codi encarregat de la generació de continguts, requerint així fer una certa inversió en recursos tant en personal com econòmics que no sempre és possible dur a terme per a algunes corporacions petites o mitjanes. Per facilitar la creació i la gestió de la web per part de personal amb menys coneixements tècnics i més orientat al màrqueting i publicitat, han anat apareixent aplicacions d'ajuda a l'administració i generació de continguts, els coneguts sistemes de gestió de continguts o CMS.

Un **sistema de gestió de continguts (CMS)** és un programa que permet crear una estructura de suport per a la creació i administració de continguts per part dels participants. És una aplicació de bases de dades que automatitza el procés d'administrar i mantenir la publicació del contingut.

Un sistema de gestió de continguts consisteix en una interfície que controla una o diverses bases de dades on s'allotja el contingut (textos, imatges, etc.) que veurem al lloc web. El sistema permet manejar de manera independent el contingut i el disseny. Així, és possible conservar el contingut i canviar en qualsevol moment el disseny sense haver de modificar el format de tot el contingut. A més, permet la publicació fàcil i controlada al lloc web. Un exemple clàssic és el d'editors que carreguen el contingut al sistema i un altre de nivell superior que permet que aquests continguts siguin visibles per a tot el públic.

Un CMS té dues funcions principals: facilitar la creació de continguts i presentar aquests continguts. Quant a la primera, proveeix una sèrie d'eines perquè publicar el contingut sigui tan fàcil com emplenar un formulari i, a més, hi hagi una sola font. Pel que fa a la segona, facilita la publicació de continguts en múltiples formats a partir d'una sola font i els afegeix metadades per facilitar la navegació.

Un CMS proveeix les eines necessàries per gestionar el cicle de vida dels continguts: creació, gestió, presentació, manteniment i actualització.

Els **avantatges** dels CMS són els següents:

1) Manteniment descentralitzat. L'actualització del contingut es pot dur a terme des de qualsevol navegador. Això permet fer les tasques des de qualsevol estació de treball, en qualsevol moment i per diversos editors simultàniament.

2) Autors de contingut amb pocs coneixements tècnics. Els autors només necessiten un coneixement estàndard de les aplicacions de processador de text per crear continguts. No cal cap coneixement d'HTML o eines addicionals. S'utilitzen editors WYSIWYG, *what you see is what you get* (el que veus és el que obtens), cosa que permet una edició com si es tractés d'un processador de text.

3) Restriccions d'accés configurables. S'assignen rols/permisos als diferents usuaris. D'aquesta manera, es protegeix el contingut de ser modificat només per als elements autoritzats.

4) Consistència del disseny. Com que se separa el contingut del disseny, aquest últim s'aconsegueix preservar molt millor. El contingut de tots els autors es presenta en un estil predefinit, conservant la unitat, la identitat i la consistència de la informació.

5) Vincles, menús i navegació generats automàticament. Els menús són generats automàticament i es basen en el contingut de la base de dades. Els vincles també, i s'eviten els vincles erronis o que no enllacen a cap lloc.

6) Reutilització de contingut. L'emmagatzematge central i independent de la presentació es tradueix en la possibilitat d'usar el mateix contingut en diferents parts del web.

7) Disseny *responsive* o adaptatiu al mitjà. El disseny de les pàgines ja està preparat perquè sigui accessible i visible adaptant el seu contingut al format més adequat segons el client des del qual s'accedeix (client escriptori, mòbils, tauleta, TV, etc.).

8) Contingut escalable. Les extensions com per exemple: fòrums, enquestes, aplicacions de compra, cerca o notícies, poden afegir-se de manera modular. Fins i tot, el CMS permet la creació d'extensions a mesura que es poden afegir de manera modular.

9) Control de versions. Simplifica les actualitzacions, registra els canvis i l'activitat feta per cada autor (*log*) i proporciona eines de col·laboració i treball en grup entre els autors.

10) Visualització del contingut segons les preferències. El contingut pot tenir una vigència per ser publicat o requerir una contrasenya. Mentrestant, pot estar amagat perquè només sigui vist pels editors.

11) Internacionalització i localització i18n. Incorpora mecanismes d'ajuda en la traducció de continguts, i també l'adaptació a diferents regions.

Moltes de les funcionalitats del CMS dependran de l'eina triada. Una de les tasques de l'administrador web serà l'elecció de l'eina que més s'adeqüi a les necessitats de l'entitat on es pretengui implantar.

No es pot (ni s'ha de fer) oferir un únic programari com el més adequat per a qualsevol projecte.

Per això, s'han de valorar aspectes com:

- Recursos disponibles per a la seva implantació.
- Analitzar el projecte, les funcions, els requeriments, etc.
- Possibilitats d'evolucions futures.
- Revisar l'origen i l'evolució de l'eina: quina empresa hi ha darrere, la temporització de les actualitzacions, la comunitat de desenvolupadors que hi ha al darrere, el suport tècnic, la documentació, etc.

En funció de les valoracions es podrà triar l'eina més òptima per al cas d'estudi i aportar decisions com:

- Tipus d'implantació. Possibilitat d'instal·lar l'eina al sistema local de l'organització o utilitzar l'eina a servidors externs o al núvol.
- Determinar si cal una versió comercial del producte o l'ús d'eines de codi obert, *open source*.

3. Arquitectura i disseny de les aplicacions web

Per a un administrador de sistemes o per a un administrador web és important conèixer els elements que componen l'arquitectura sobre la qual està constituïda la web per poder donar el millor suport possible tant a la capa de sistemes com a la capa de programació, sobretot en aquelles infraestructures on la implantació es farà als sistemes locals de l'organització. Tal com s'ha comentat, l'arquitectura web es basa en el paradigma de traspàs de la informació entre el client i el servidor mitjançant el protocol HTTP. Per això, s'ha de tenir en compte els diferents elements que intervenen en el seu funcionament tant pel que fa a l'arquitectura física com a la lògica de disseny.

3.1. Arquitectura lògica contra arquitectura física

El disseny de l'**arquitectura física** se centra en l'elecció dels elements físics (*hardware*), la xarxa i els programes necessaris per a la posada en marxa del sistema per atendre de la forma més òptima les necessitats de l'aplicació i, al mateix temps, respectar les restriccions tècniques i econòmiques del projecte.

De l'arquitectura física s'obtindran els requisits que afectin el funcionament del sistema. Per tant, s'hauran de tenir en compte tots aquells factors que afecten el dimensionament de la infraestructura: rendiment, escalabilitat, disponibilitat, seguretat, cost, etc., i també la ubicació dels sistemes: local, *cloud*, *housing* en funció de la càrrega que suportarà el lloc web.

Per això, disposarem de diferents solucions possibles en funció de les necessitats (vegeu figura 6) i, encara que algunes d'aquestes ja s'han comentat en altres mòduls de l'assignatura, en farem un breu recordatori:

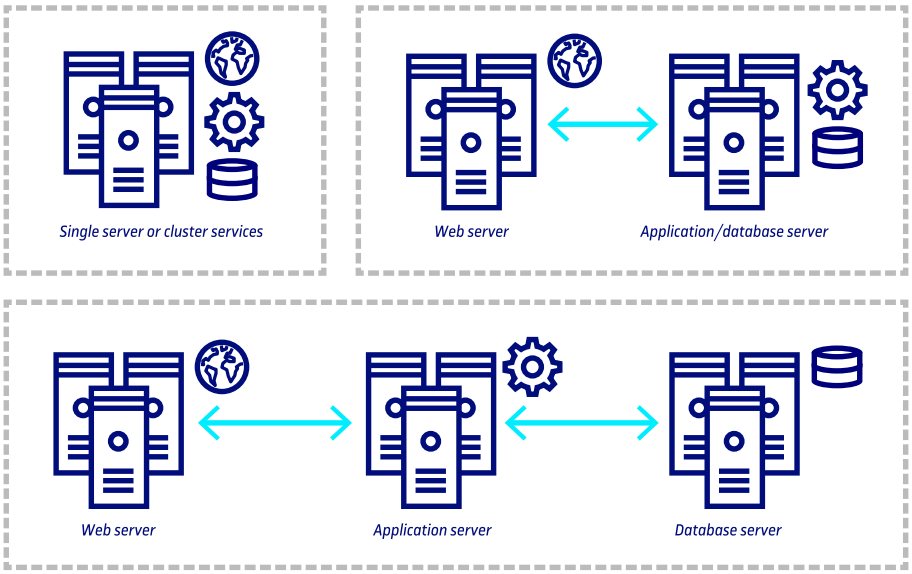
- **Servidor únic**: és el model més simple i el de menor cost, encara que el de menys de confiança. Tots els serveis operen al mateix servidor de manera que el rendiment dependrà de les característiques de RAM i CPU de la màquina. L'escalabilitat dependrà del tipus de màquina utilitzada i la disponibilitat és baixa ja que, davant una caiguda del servidor, tant la base de dades com els serveis web deixaran de funcionar.

- **Servidors web replicats**: mitjançant una configuració del servidor en clúster es millora el rendiment mitjançant la possibilitat de realitzar un balanceig de la càrrega entre els servidors i que sigui visible com una sola entitat encara que cadascun d'aquests ofereixi els serveis de manera individual. En cas de fallada d'un d'aquests, la càrrega de treball queda distribuïda als altres nodes del clúster.

- **Separació de la base de dades**: afegint una o més màquines per allotjar la base de dades, permet dimensionar-les en funció del programa a utilitzar, millorant així el rendiment i adaptant les característiques de cadascuna d'aquestes segons les necessitats. Permet escalar de manera independent cadascun dels nivells i millora la seguretat, ja que l'accés a la base de dades es limita només al propi servidor web.

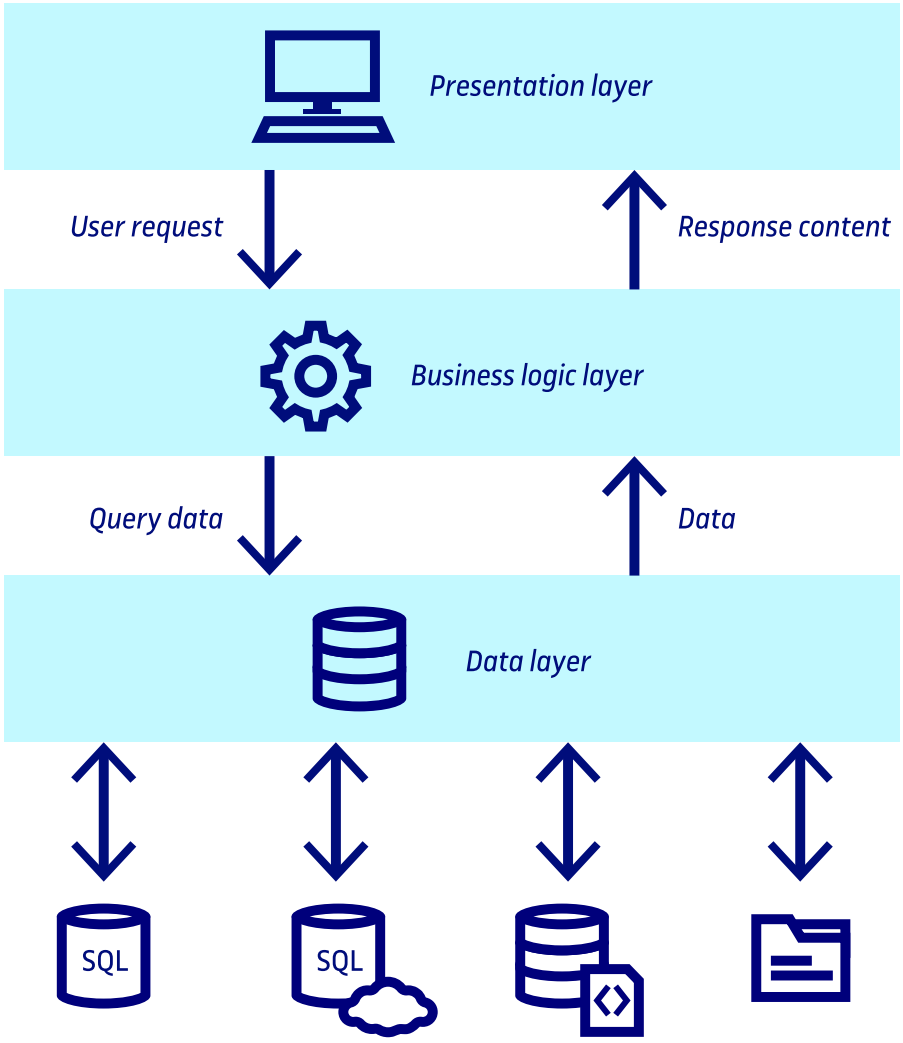
- **Separació del servidor d'aplicacions**: el servidor d'aplicacions és la part del programa separada del servidor web que s'encarrega d'executar els processos a la capa de negoci que és l'encarregada de generar els continguts dinàmics. Separant els servidors d'aplicacions web millorem novament el rendiment dels servidors a utilitzar i la seva escalabilitat.

Figura 6. Arquitectura física del web



L'**arquitectura lògica** fa referència al model que representa el disseny estructural sense limitar la tecnologia ni l'arquitectura a un entorn en particular i determina com es comuniquen els components de l'aplicació entre si. En el cas de les aplicacions web, determina com es comuniquen el client i el servidor a la capa funcional i d'informació i va lligada a l'arquitectura física vista anteriorment. L'arquitectura de les aplicacions web sol dividir-se en diferents capes o nivells permetent així poder actualitzar-les o substituir-les de manera individual. D'aquest patró arquitectònic se'n diu arquitectura de tres o múltiples nivells, del qual podem veure una representació a la figura 7.

Figura 7. Arquitectura lògica del web



Bàsicament les capes o nivells que componen l'arquitectura lògica d'una aplicació web són:

1) Capa de presentació (*presentation layer*): és la capa més externa del model i s'ocupa de la presentació del contingut i de la interacció amb l'usuari. Es pot anomenar vista, presentació, UI. En aquesta capa, l'aplicació mostra a l'usuari el que necessita veure i ofereix les eines per a la interacció. Les tecnologies generalment involucrades en aquesta capa, en el context de desenvolupament web, són principalment el llenguatge HTML encarregat de mostrar la informació, l'estil de la pàgina (CSS) i els *scripts* d'execució al client (JavaScript, etc.). Els aspectes de visualització com ara l'adaptació de la vista al dispositiu s'executen en aquest nivell. Aquesta capa es comunica amb la capa de negoci mitjançant l'enviament de peticions que l'usuari fa a l'espera que aquesta resolgui la petició enviant de nou el contingut a mostrar.

2) Capa de negoci (*business logic layer*): és la capa que s'ocupa de la lògica del programa. Rep dades del nivell superior i les transforma, utilitzant les lògiques internes de l'aplicació. També recupera dades del nivell de dades més profund

i les usa per a la lògica. I, quan s'integren aquests dos processos, també pot fer modificacions a tots dos nivells. Alguna de les tasques que es realitzen en aquesta capa són:

- gestió del flux de l'aplicació,
- realitzar tots els càlculs i les validacions requerits,
- identificació de l'usuari, o
- administrar tot l'accés a les dades per a la capa de presentació.

La capa lògica de negocis generalment s'implementa dins d'un servidor d'aplicacions. Les eines utilitzades en aquest nivell solen ser llenguatges de programació del costat del servidor com: PHP, Perl, Python, Java, JavaScript amb node.js, Ruby, etc.

3) Capa de dades (*data layer*): és la capa de nivell més profunda i s'ocupa de la recuperació de dades de les seves fonts en les peticions realitzades des de la capa de negoci i retorna a aquesta les dades obtingudes després de la petició. Un bon disseny en aquesta capa permet canviar la seva lògica interna o l'origen de les dades sense afectar la capa de negoci si es continuen retornant les mateixes dades. Les tecnologies utilitzades en aquesta capa són sistemes de gestió de bases de dades com ara MySQL o PostgreSQL per a bases de dades relacionals, sistemes de gestió NoSQL com MongoDB o, fins i tot, arxius XML o arxius de text.

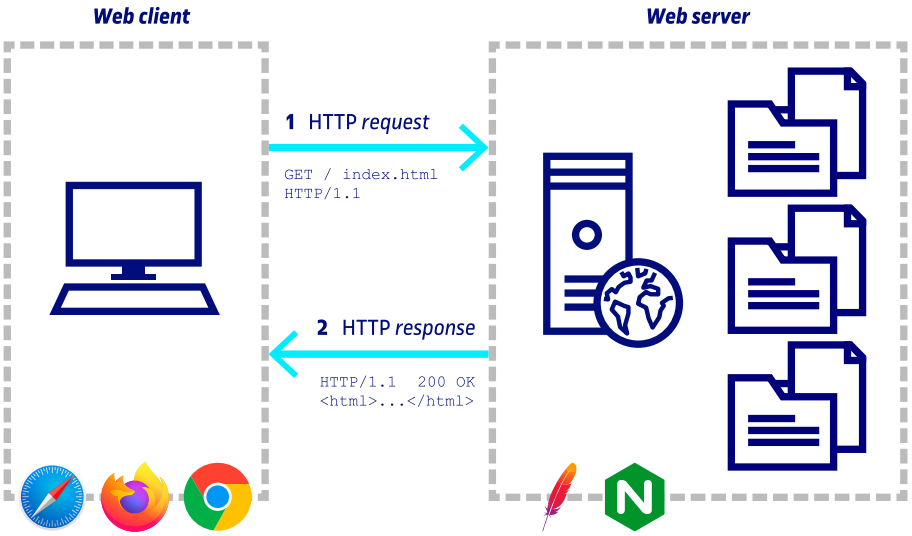
3.2. Pàgines estàtiques i dinàmiques

Tal com hem vist, els llocs webs es caracteritzen per mostrar informació per mitjà de la resposta de contingut d'un servidor web sobre la base d'una petició d'un client, com pot ser un navegador, la comunicació del qual transcorre mitjançant un protocol HTTP o HTTPS. En el cas d'un client web, aquests continguts seran retornats en llenguatges interpretables pel mateix client: HTML (el text que veiem escrit a les pàgines web), CSS (fulls d'estil en cascada), que són els estils i dissenys aplicats a aquestes pàgines, i JavaScript, un llenguatge de programació que defineix el seu comportament (per exemple, fondre's i desaparèixer, efectes de desplaçament, etc.). Aquesta resposta de continguts la podem distingir en dues tipologies en funció de com han estat generats: pàgines dinàmiques i pàgines estàtiques.

En el cas de les **pàgines estàtiques** (vegeu figura 8), la resposta a la sol·licitud del client mitjançant el navegador web es realitza sense cap modificació dels continguts i es mostren tal com van ser escrits i emmagatzemats, com a arxius simples al servidor. Aquests continguts romanen sempre visibles de la mateixa manera, tant en disseny com en contingut, per a tots els clients fins que es modifiquin els fitxers de manera manual. Sol utilitzar-se per a petits llocs web en què el contingut rarament serà modificat i no es requereix cap mena de

complexitat. La resposta del servidor davant la petició de pàgines estàtiques és molt ràpida, ja que aquestes són de petita grandària i no calen modificacions prèvies.

Figura 8. Generació de pàgines estàtiques o dinàmiques



En el cas dels llocs web amb **pàgines dinàmiques**, la resposta a la sol·licitud del client és igualment una pàgina HTML interpretable pel navegador encara que aquesta conté continguts generats de manera automàtica mitjançant llenguatges de programació que obtenen la informació a mostrar a partir d'una font de dades. Aquesta font de dades sol ser una base de dades relacional encara que poden obtenir-se d'altres tipus de fonts com poden ser: bases de dades NoSQL, fitxers, API, serveis al núvol, etc. Per a aquestes, s'usen llenguatges de programació web addicionals al mateix HTML que ajuden a generar més contingut d'aquest tipus. Alguns dels llenguatges més usats per a aquest propòsit solen ser: PHP, Python, Perl, Ruby, etc.

Les pàgines dinàmiques poden ser generades tant del costat client (*client-side*) com del costat servidor (*server-side*) o una combinació de tots dos.

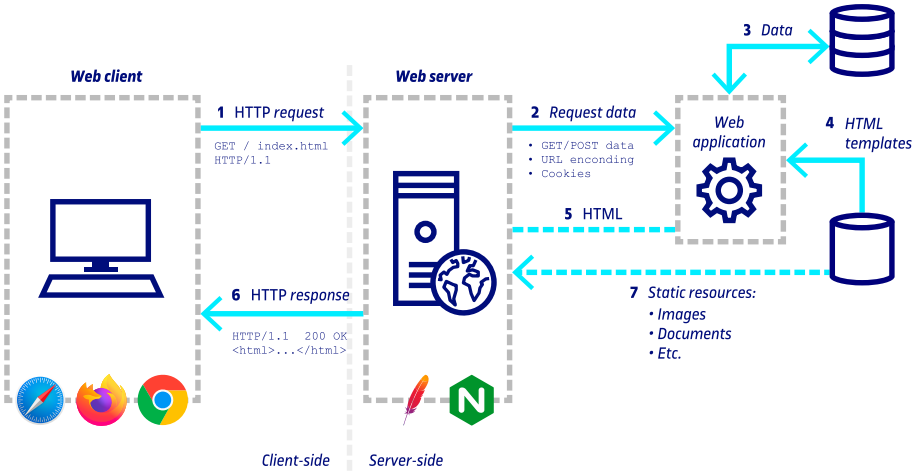
3.3. Aplicacions del costat del servidor

Dins del dinamisme de les pàgines web, es poden definir dos mètodes de generació de continguts en funció d'on s'executa el codi de l'aplicació.

Quan es fa referència a la generació de continguts web del costat del servidor, generalment es refereix a una aplicació o lloc web que utilitza un llenguatge de programació que **s'executa en un servidor**. És al servidor on es generen les pàgines web amb tot el contingut necessari donant com a sortida el codi HTML complet per ser reproduït al navegador, on la pàgina es mostra de la mateixa manera que es rep, com si es tractés d'un contingut estàtic. Normalment, com que el servidor web disposa de millors prestacions que el client, fa que la resposta en la generació de continguts sigui ràpida i eficient. En contrapartida, per a cada acció de l'usuari, es genera una nova petició per URL al servidor que

fa que hagi de realitzar tot el treball de nou: interpretar el codi, recollir dades, compilar i generar el codi HTML i retornar-lo al client juntament amb els fulls d'estil CSS, si fos necessari. A la figura 9 es pot veure un exemple del model a seguir en cas de generació de pàgines del costat del servidor.

Figura 9. Aplicacions web al servidor



Els principals **avantatges** són:

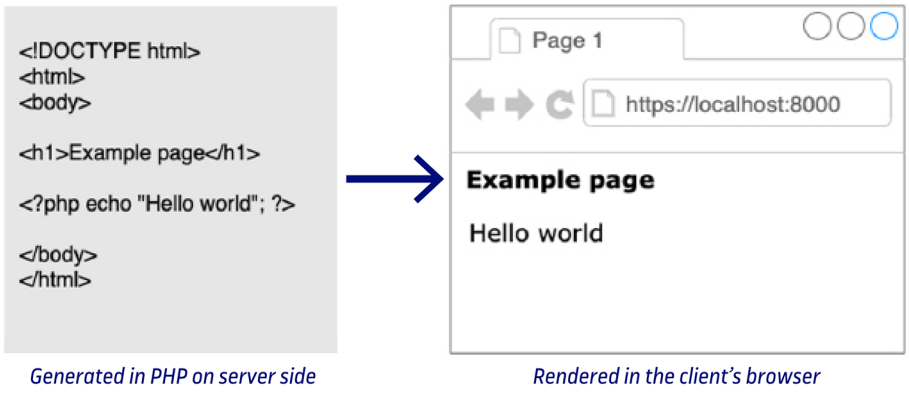
1) Útil per a llocs web amb poca interactivitat amb l'usuari.
2) Alta velocitat en la càrrega de la pàgina.
3) Els motors de cerca poden rastrejar el lloc obtenint millors resultats.

Com a **inconvenients** podem trobar:

1) Peticions freqüents al servidor, podent sobrecarregar el servidor.
2) Les pàgines han de recarregar-se de manera completa per petits canvis que hi hagi.
3) Les interaccions de l'usuari amb l'aplicació són lentes.

Molts dels llenguatges de programació estan dissenyats per treballar en un dels dos costats del lloc web, del costat del servidor o del costat del client. Del costat del servidor podem trobar llenguatges com: PHP, Python, Java o JSP, Ruby, Perl, etc. A la figura 10 podem veure un exemple de com es mostra una pàgina generada al servidor.

Figura 10. Exemple de pàgina generada al servidor



3.4. Aplicacions del costat del client

Quan parlem de la representació del costat del client, bàsicament es tracta de representar el contingut al navegador mitjançant l'ús del llenguatge JavaScript, interpretable per la majoria dels navegadors web que **s'executen al client**. D'aquesta forma, en lloc d'obtenir tot el contingut del mateix document HTML, es rep un document HTML bàsic amb codi o un arxiu JavaScript en la càrrega inicial, que és el que permet generar la resta del contingut utilitzant el navegador. El principi bàsic per renderitzar les pàgines al navegador es basa en la manipulació del DOM (*document object model*).

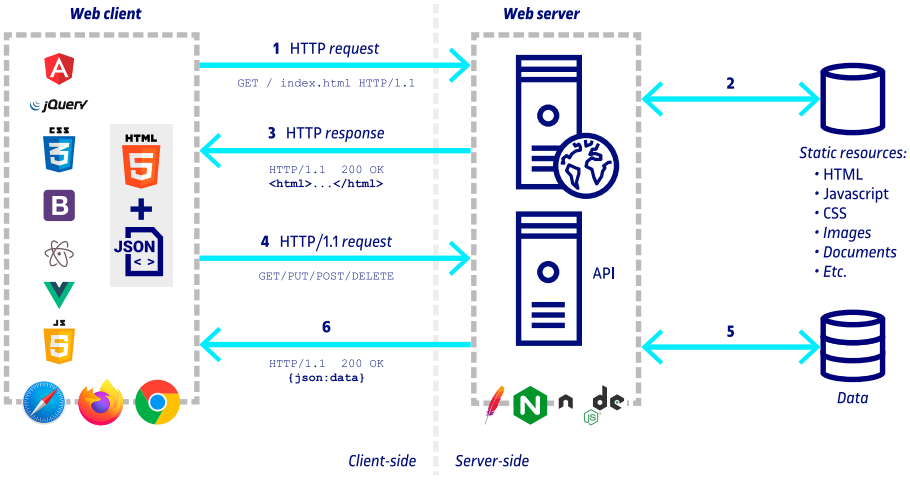
DOM

El model d'objectes del document (DOM) és una interfície de programació d'aplicacions (API) per a documents vàlids HTML i ben construïts XML. Defineix l'estructura lògica dels documents i la manera com s'hi accedeix i manipula.

Font: W3.org, 2020.

En termes generals, el servidor exposa una API (sovint RESTful, es veurà més endavant en aquest mòdul) que retorna dades (generalment JSON). Aquesta vegada és tasca del client construir la pàgina: necessita obtenir les dades del servidor i inserir les dades en l'HTML utilitzant la manipulació DG. Aquí és on entra en joc JavaScript.

Figura 11. Aplicacions web al client



Tal com podem veure a la figura 11, quan un usuari navega a un lloc web amb una arquitectura web del costat del client, el navegador primer sol·licita l'HTML del servidor. Contràriament a l'arquitectura del costat del servidor, l'HTML no conté cap dada. És simplement una plantilla estàtica en la qual les dades s'injectaran més tard. En aquest punt, el navegador ja pot mostrar l'HTML sense format (disseny general, etiquetes, imatges, etc.).

A continuació, el navegador realitza una o més sol·licituds GET asíncrones a l'API del servidor per obtenir les dades. El servidor recupera les dades de la base de dades i les retorna al navegador. Finalment, el navegador insereix les dades a l'HTML (manipulant el DOM) i formata la pàgina amb el contingut.

D'aquesta manera, a diferència de l'arquitectura del costat del servidor, on per visualitzar els canvis calia recarregar tota la pàgina, el navegador només canvia una part d'aquesta a partir de les dades obtingudes per l'API. A aquesta tècnica d'actualització de continguts de manera asíncrona també se la denomina AJAX (*asynchronous JavaScript and XML*).

Aquesta arquitectura ofereix els **avantatges** següents:

1) Interaccions d'usuari ràpides mitjançant components dinàmics.

2) Ús eficient de la xarxa. Només es carrega una part de la pàgina, millorant així l'accés en cas de connexions amb poca amplada de banda.

3) Ús eficient de la base de dades. Només es consulten les dades específiques del moment.

4) Els llenguatges del costat del client redueixen la càrrega del servidor.

5) Representació ràpida del lloc web després de la càrrega inicial.

6) El servidor API pot ser un servidor independent al de la web i, fins i tot, pot ser un servei al núvol.

7) Reutilització de l'API. Les crides i resultats poden reutilitzar-se en altres parts de la web o en aplicacions diferents.

8) Ideal per a aplicacions web.

9) Gran repositori de llibreries JavaScript disponible per a execucions del costat client.

Com a **inconvenients** podem trobar:

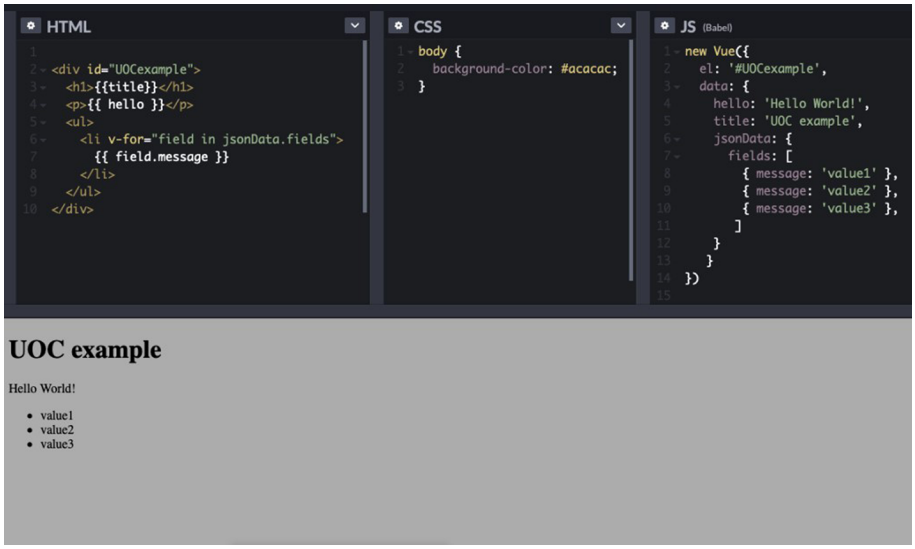
1) Baixa optimització per als motors de cerca si no es programa correctament.

2) La càrrega inicial pot necessitar més temps.

3) Alta dependència en el tipus de navegador i les seves versions.

4) Com que s'executa del costat del client, els recursos dependran del tipus de màquina des de la qual es connecti.

Figura 12. Exemple de pàgina generada del costat client



Els llenguatges utilitzats del costat del client per a les aplicacions web són tots aquells que siguin compatibles amb els navegadors, majoritàriament: JavaScript, HTML (per estructura), CSS (per estils) i JSON (per dades). A la figura 12 es pot veure un exemple de com s'obté la visualització d'una pàgina generada des del client a partir d'una plantilla HTML. El JavaScript és l'encarregat de modificar el DOM d'aquest document HTML i l'aplicació dels estils mitjançant el fitxer CSS.

Inicialment es van crear llibreries JavaScript d'ajuda per al control de les interaccions, el dinamisme i les peticions AJAX al servidor i per a la representació gràfica dels elements en la interfície d'usuari (UI), com per exemple: jQuery (JavaScript), Vue i Bootstrap (elements gràfics i CSS), però a mesura que passi el temps en poden aparèixer de noves.

3.5. Arquitectura model-vista-controlador (MVC)

En els inicis del desenvolupament web, amb les tecnologies existents, la generació de pàgines dinàmiques es basava a barrejar codi de llenguatge de marca (HTML), amb codi de programació en un mateix fitxer per a la generació de la pàgina web. Fins i tot es podia separar la capa d'obtenció de dades i la de generació d'HTML en diferents arxius per facilitar el seguiment del codi però, a mesura que anava creixent l'aplicació i, per tant, el nombre d'arxius, es feia molt difícil d'identificar cadascuna de les parts de l'aplicació i, en conseqüència, el manteniment d'aquesta. És per això que es va crear un patró per orga-

nitzar el codi, la qual cosa es coneix com a arquitectura model-vista-controlador (MVC). El patró MVC es basa a **separar el model de negoci de la capa de visualització**. Amb això s'aconsegueixen diversos **avantatges**:

- Múltiples desenvolupadors poden col·laborar en diferents capes del model de manera independent.
- Aplicacions fàcilment actualitzables.
- Compartició de dades per a múltiples vistes.
- Les modificacions no afecten tot el model.
- Facilita el desenvolupament de grans aplicacions.

L'arquitectura MVC està composta per **tres capes**:

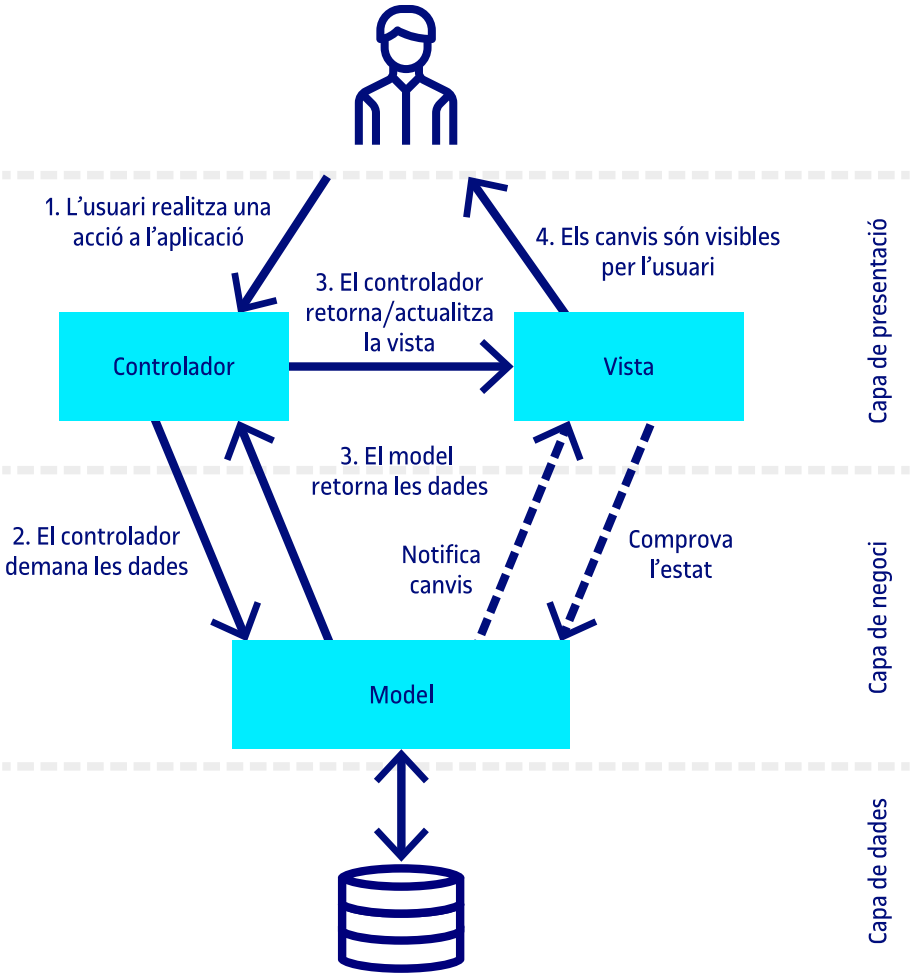
1) Model: és l'encarregada d'emmagatzemar les dades de l'aplicació i gestionar la lògica relacionada.

2) Vista: és la part de l'aplicació que mostra la representació de les dades. Normalment correspondrà al llenguatge HTML interpretable pel navegador i es generarà a partir de les dades obtingudes en el model.

3) Controlador: és la capa responsable de respondre a les accions de l'usuari. El controlador actualitza el model i porta a terme les accions necessàries quant a tractament de les dades per a, posteriorment, actualitzar la vista.

A la figura 13 es pot veure un exemple del cicle que segueix la interacció de l'usuari amb una estructura web basada en el model MVC.

Figura 13. Cicle interacció en MVC



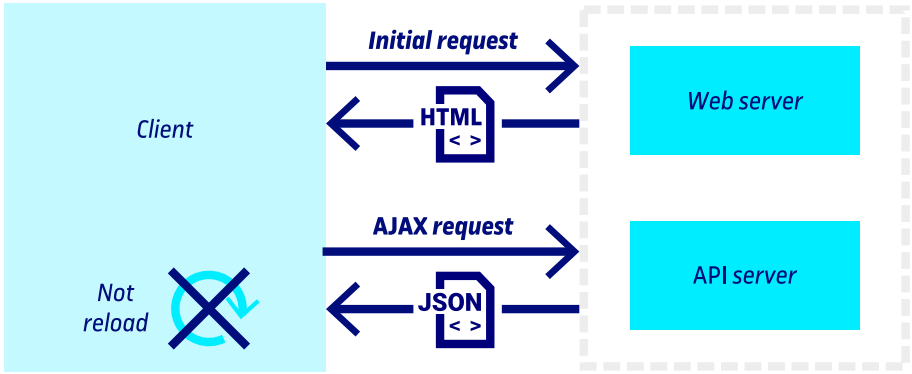
Actualment, hi ha diferents *frameworks* que faciliten aquest tipus de programació, ja que es basen en el patró MVC. Algun dels *frameworks* actuals més importants poden ser: Angular, Vue, Re-act, Ionic, Meteor, Django, Ruby on Rails, etc., encara que amb el pas del temps poden aparèixer nous entorns que ofereixin aquest tipus d'arquitectura de programació.

3.6. Arquitectura de pàgina única (SPA)

Dins de l'arquitectura d'aplicacions web del costat del client, en què s'ha pogut veure com el contingut de les pàgines és actualitzat de manera asíncrona i interactiva sense necessitat de recarregar tot el contingut de la pàgina, hi ha una altra modalitat en l'arquitectura del costat client que permet mantenir aquest dinamisme no solament dins de la pàgina actual obtinguda del servidor, sinó de tot el conjunt de l'aplicació web. És a dir, de la mateixa manera que a les webs estàtiques o en l'arquitectura basada en el servidor, qualsevol petició HTTP generada a partir d'una URL diferent necessita una recàrrega de pàgina total del costat client quan rep la resposta HTTP amb el contingut HTML del servidor, en algunes aplicacions del costat del client, malgrat que té el dinamisme AJAX de la pàgina, pot requerir la mateixa recàrrega del costat del client quan canvia l'URL i, per tant, rebre com a resposta una nou contingut HTML i obtenir noves dades de l'API. En moltes de les aplicacions web

d'aquest estil, el contingut era moltes vegades repetit, fins i tot amb vistes diferents. Per millorar el dinamisme de les aplicacions web i la seva usabilitat es va crear l'arquitectura anomenada SPA (*single page architecture*) o arquitectura de pàgina única. El cicle d'execució (vegeu figura 14) és el mateix que s'ha vist en el cas de l'arquitectura de clients, però **sense la necessitat de recarregar la pàgina** quan es canvia de ruta o URL.

Figura 14. Cicle d'execució d'una arquitectura SPA



Per facilitar el desenvolupament d'aquest tipus d'aplicacions més complexes, se solen utilitzar entorns de treball amb un conjunt de llibreries i eines anomenades *frameworks* que estan destinades a generar contingut seguint el model MVC, la tecnologia AJAX/API i el model d'arquitectura SPA. Algun exemples dels *frameworks* de JavaScript més utilitzats en l'actualitat són: React, Angular, Ember.js, Meteor, Node.js.

4. Arquitectura orientada als serveis (SOA)

Des del principi, els models de desenvolupament han anat evolucionant amb el temps. Si mirem enrere, al començament dels anys vuitanta va començar la programació orientada als objectes, cap als noranta va fer-se popular el model-vista-controlador (MVC) i els models basats en components. Ara, la tendència més actual és fer servir models orientats als serveis.

El model orientat als serveis ofereix una nova capa, anomenada *capa de serveis*, en què la resta de components del sistema exposen les seves funcions (serveis) perquè puguin ser utilitzades per tercers que anomenarem consumidors.

Un **servei** és una funció bàsica que un proveïdor exposa perquè diferents consumidors n'obtinguin un resultat.

Un servei, dintre d'una arquitectura SOA, ha de tenir les característiques següents:

- Ha de tenir una interfície ben definida (o un contracte de servei). Ha de comptar amb una descripció de com ha de ser consumit el servei (nom, paràmetres, resultat i ubicació).

- No ha de guardar cap tipus d'estat. El servei ha de rebre tota la informació que necessita per poder donar una resposta.

- Ha de poder ser reutilitzable. El servei ha de poder ser cridat o consumit per més d'un consumidor.

- Ha de tenir un baix acoblament amb la resta de serveis. Idealment, ha d'oferir la funcionalitat sense dependre d'altres serveis.

Per acabar d'entendre el concepte de servei us posarem un exemple. Imagineu-vos la implementació del procés que fa possible una transferència bancària. Les passes necessàries per poder assolir aquest procés serien: saber la quantitat que es vol transferir, consultar el saldo de l'emissor (per saber, si el client disposa del capital a transferir), comprovar si el compte destí existeix, fer la transferència al compte destí i informar del resultat de la transferència.

D'aquest exemple podem destacar el servei consulta de saldo. Fixeu-vos que aquest servei no depèn de cap altre i pot ser utilitzat per altres processos com pot ser treure de diners d'un caixer automàtic.

4.1. Rols de l'arquitectura SOA

Generalment, en una arquitectura SOA podem trobar **tres rols** diferenciats.

1) Consumidor de serveis: és qui demana la funcionalitat proporcionada per un servei i l'encarregat de fer la cerca en el registre de serveis per tal de trobar la funcionalitat adient.

2) Proveïdor de serveis: és el que proporciona la funcionalitat al servei i també l'encarregat de publicar la definició i descripció del servei en el registre perquè els consumidors puguin descobrir i accedir al servei.

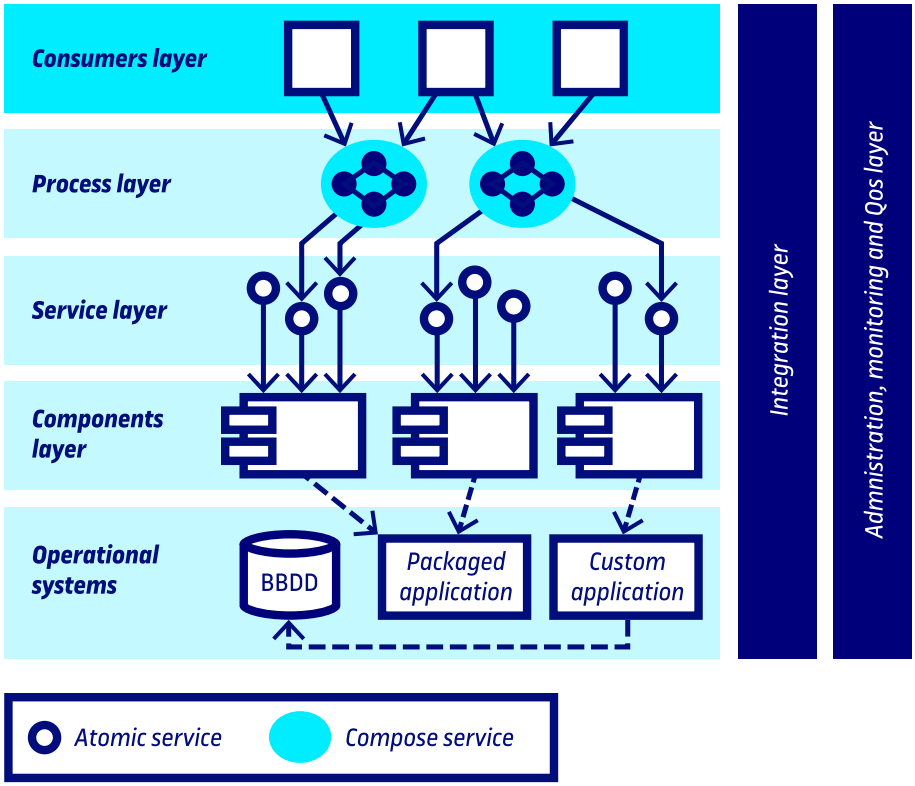
3) Registre de serveis: és un directori on hi ha totes les definicions i descripcions del serveis que hi ha registrats. Dona resposta als consumidors respecte a la localització del servei indicant el proveïdor concret al qual han d'accedir.

4.2. Capes de l'arquitectura SOA

Normalment una organització disposa de diferents aplicacions heterogènies que es fa molt difícil d'interconnectar-les. L'objectiu principal d'aplicar un model d'arquitectura SOA en una organització és bàsicament poder integrar les diferents aplicacions i bases de dades i facilitar la comunicació entre si.

S'acostumen a definir diferents capes d'abstracció per representar una arquitectura SOA, les quals estan alineades amb els diferents processos d'una organització. Tot això ho podem veure amb detall a la figura 15.

Figura 15. Capes de l'arquitectura SOA



1) Les capes horitzontals

a) Capa de sistemes operacionals: representa la capa on anirien totes les aplicacions de l'organització (sistemes *legacy*)[2] que caldria integrar-les mitjançant SOA. Per exemple, aplicacions orientades a objectes, ERP,[3] etc.

(2)Sistema que ha quedat antiquat però encara es fa servir.

(3)Acrònim d'*enterprise resource planning*.

b) Capa de components: aquí trobem els components encarregats d'implementar la funcionalitat que exposen els serveis. Aquests components els podem trobar implementats amb diferents tecnologies.

c) Capa de serveis: és la capa on hi ha exposats els serveis que s'ofereixen. Normalment cada servei acostuma a tenir un contracte en què s'indica la funcionalitat que realitza aquest servei. Per tal d'oferir la seva funcionalitat, el servei acaba cridant els components que necessita de la capa inferior.

d) Capa de processos: és la capa on es duu a terme l'orquestració dels serveis i és l'encarregada de seqüenciar els serveis i dotar de la lògica de negoci necessària per processar la informació.

e) Capa de consumidors: és la capa des d'on es criden els serveis o les orquestracions de la capa anterior. Aquí hi hauria els usuaris que volen obtenir la funcionalitat que desitgen consumir.

2) Les capes verticals

a) Capa d'integració: permet integrar tots els serveis definits a l'arquitectura i facilita la seva interoperabilitat. Això ho aconsegueix implementant algunes capacitats com ara l'enrutament, la mediació i les transformacions de protocols. També permet no tenir lligat el consumidor a la localització del proveïdor del servei.

b) Capa d'administració, monitoratge i QoS: aquesta capa defineix el que és conegut com a governança SOA. Aporta la capacitat de monitoratge i l'alta disponibilitat perquè la comunicació sigui confiable. Aquí també trobem les capacitats d'autenticació, d'autorització i d'encriptació de la comunicació.

La implementació d'aquestes capes verticals es fa mitjançant el que anomenem ***bus de servei empresarial*** (ESB).

El bus de servei empresarial (ESB)

Es tracta de la capa més externa i és la que bàsicament facilita la intercomunicació de tots els serveis heterogenis que una organització vol tenir integrats. Sense aquesta capa, els consumidors i els proveïdors dels serveis haurien de realitzar una connexió punt a punt i això fa que dificulti molt la gestió si disposem d'un nombre elevat de serveis.

Què aporta la implementació d'aquest mecanisme en una organització?

- **Desacoblament** entre els clients i els proveïdors, ja que totes les connexions les gestiona el mateix bus.

- Integra de forma transparent diferents protocols de transport (HTTP, JMS, FTP, SMTP, etc.). Per tant, facilita la **mediació entre els diferents protocols** fent les transformacions adients.

- Tal com passa amb els protocols de transport, també permet integrar **diferents formats de missatges**.

- Proporciona un **encaminament complex** dels missatges. El bus és l'encarregat de saber la destinació dels missatges sense que el client s'hagi de preocupar. Podríem dir que aporta virtualització a la ubicació dels serveis.

- Aporta una **garantia de lliurament dels missatges**. Gràcies als mecanismes d'encuament i l'ús de polítiques de reintent d'enviament dels missatges permet eliminar el risc de tenir indisponible el sistema de destí.

- Capaç d'implementar **mecanismes de seguretat** com ara l'autenticació, l'autorització i l'encriptació tant a l'entrada com a la sortida dels missatges.

- Ens facilita l'**administració** i el **monitoratge** per poder fer posteriors auditories.

4.3. Els serveis web i SOA

Bona part de l'arquitectura SOA es pot implementar amb el que s'anomenen *serveis web* (WS).[4] Cal deixar clar que no és l'única forma de fer-ho, es pot implementar fent servir altres tecnologies. Tenir implementats serveis web en un sistema tampoc no implica tenir una arquitectura orientada al servei.

(4)De l'anglès *web services*.

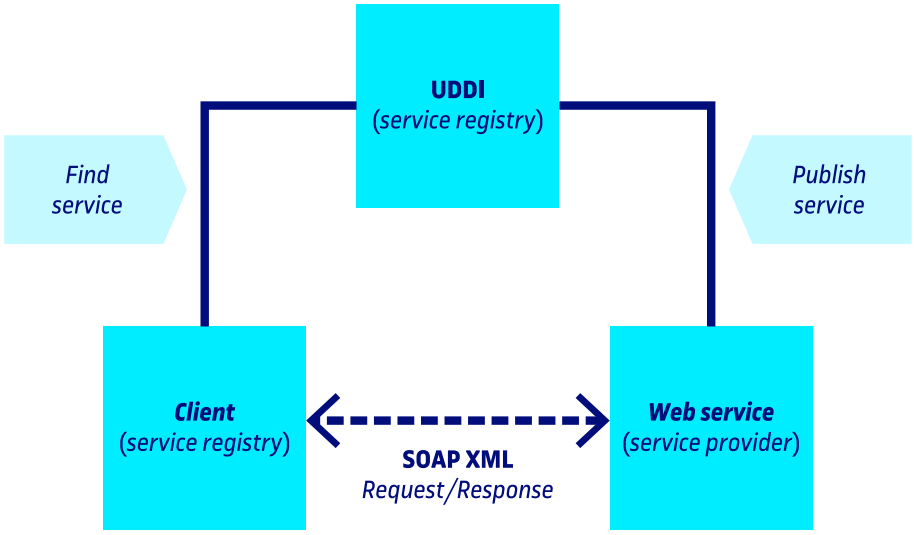
Un **servei web** és una tecnologia que fa servir un conjunt de protocols i estàndards amb l'objectiu d'intercanviar dades de diferents aplicacions.

Els responsables de definir aquesta arquitectura i els seus reglaments són les organitzacions OASIS i W3C.

El funcionament d'aquesta tecnologia és el següent. Primer, el proveïdor d'un servei ha de publicar en el registrador la definició i descripció d'aquest servei fent servir el llenguatge WSDL. D'aquesta manera, acaba publicant el servei en un catàleg o directori d'interfícies de serveis web anomenat UDDI. Per altra

banda, el consumidor del servei contacta amb el registrador per saber quin es el proveïdor d'aquest servei. Posteriorment, el consumidor contacta amb el proveïdor fent servir el protocol SOAP i aquest valida la petició retornant les dades que el consumidor havia demanat. Podeu veure aquest tipus de funcionament a la figura 16.

Figura 16. Flux de funcionament d'un servei web



Tots aquests protocols fan ús del format XML i són transportats mitjançant el protocol HTTP.

4.3.1. WDSL (*web services description language*)

És el llenguatge que es fa servir per descriure un servei. Gràcies a aquest fitxer, el consumidor pot saber quina interfície té el servei que representa i els tipus de dades necessàries per a la seva utilització. El format d'aquest fitxer és XML i segueix l'estructura següent:

```
<definitions>
<types>...</types>
<message>...</message>
<portType>...</portType>
<binding>...</binding>
<service>...</service>
</definitions>
```

1) Tipus de dades `<types>`: és la secció on es declaren tots els tipus de dades que es faran servir. Per definir els tipus de dades, normalment es fa servir el format XML Schema.

2) Tipus de missatges `<message>`: aquí es declaren tots els missatges d'entrada i sortida de les operacions. Per a cada missatge, cal declarar el nom i el tipus de dades.

3) Tipus de port `<portType>`: conté les diferents operacions. Una operació és una agrupació lògica d'un conjunt de missatges intercanviats. Cada operació pot definir com a molt un tipus de missatge d'entrada i un altre de sortida.

4) Lligadures `<bindings>`: és on es defineix el format del missatge i els detalls del protocol per a cada tipus de port.

5) Servei `<service>`: un servei representa un tipus de port, amb una lligadura concreta accessible mitjançant una adreça donada.

4.3.2. UDDI (*universal description, discovery and integration*)

Inicialment, tres empreses (Microsoft, IBM i SAP) van crear aquesta iniciativa de catàleg global. L'objectiu era donar-se a conèixer les unes a les altres compartint informació mitjançant aquest catàleg.

UDDI és un registre que permet emmagatzemar informació de forma estructurada amb l'objectiu de donar a conèixer els serveis que una organització vol oferir.

UDDI permet emmagatzemar tres tipus d'informacions:

1) Pàgines blanques: és on s'enregistra el nom, la descripció, el tipus d'organització, el contacte, etc.

2) Pàgines grogues: detalls sobre la taxonomia de l'organització. Aquí es detalla el sector industrial al qual pertany, els codis de producte, el codi de negoci, etc.

3) Pàgines verdes: és on es detallen tots els temes tècnics dels serveis web que s'ofereixen, perquè els clients tinguin els detalls per poder invocar-los.

Així doncs, per fer possible una entrada en un registre UDDI, cal facilitar la informació següent:

- Nom de l'empresa, amb la seva descripció, si pot ser en diferents idiomes i el contacte dels responsables que han implementat els serveis web.

- Els serveis web que l'organització ofereix i els arxius WSDL amb la seva descripció.

- Les categories i identificacions adients per a l'organització o servei.

El model de dades UDDI es compon dels elements següents:

1) **businessEntity**: aquí es detalla la informació del proveïdor que publica el servei.

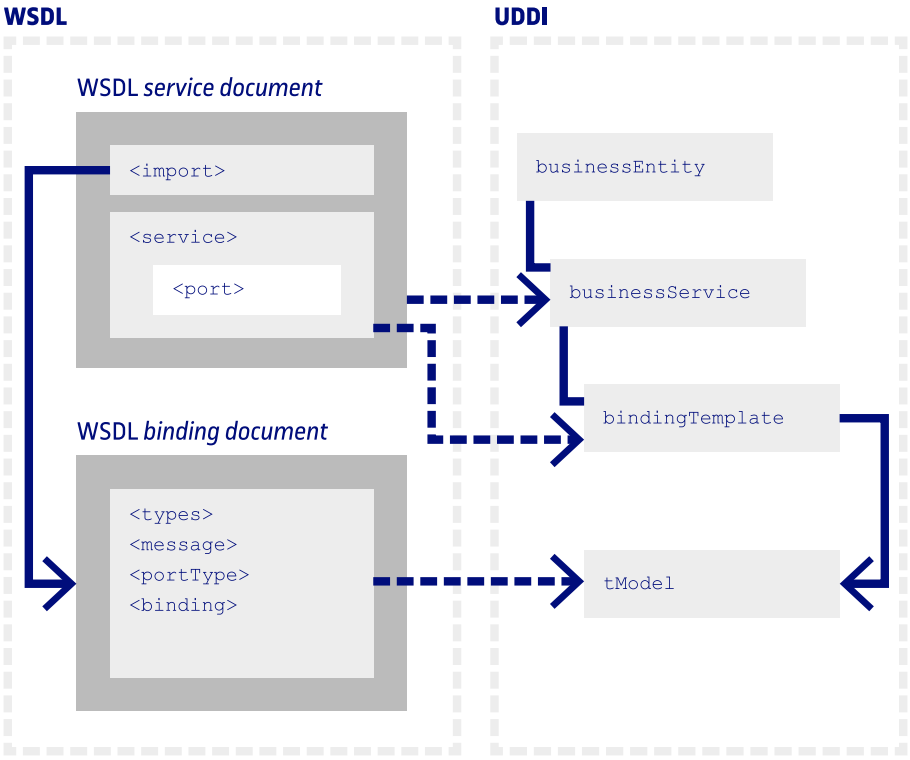
2) **businessService**: descripció de la funció que implementa el servei.

3) **bindingTemplate**: detalls tècnics de la implementació del servei. Ha d'incloure l'URL que el client ha de fer servir per invocar el servei.

4) **tModel**: inclou atributs o metadades sobre les diferents especificacions representades per un servei web determinat.

A la figura 17 podem trobar les relacions que hi ha entre WSDL i UDDI.

Figura 17. Relació entre WSDL i UDDI



El sistema UDDI no es va estendre tant com els seus creadors haguessin volgut. Des de gener de 2006, cap de les tres empreses ja no fa servir aquest catàleg. Actualment, els sistemes UDDI es poden trobar en algunes organitzacions i es fan servir per lligar els sistemes dels clients a les implementacions. També es fa servir entre organitzacions que vulguin col·laborar entre elles.

4.3.3. SOAP (*simple object acces protocol*)

És el protocol de missatgeria que es fa servir per comunicar el client amb el proveïdor del servei. És un mecanisme simple i lleuger i fa servir el format XML.

L'especificació SOAP conté tres parts:

1) Sobre (*envelope*): és on es descriu el que hi ha d'haver en el missatge i com s'ha de processar. Té dos elements fills: les capçaleres (*headers*) i el cos (*body*).

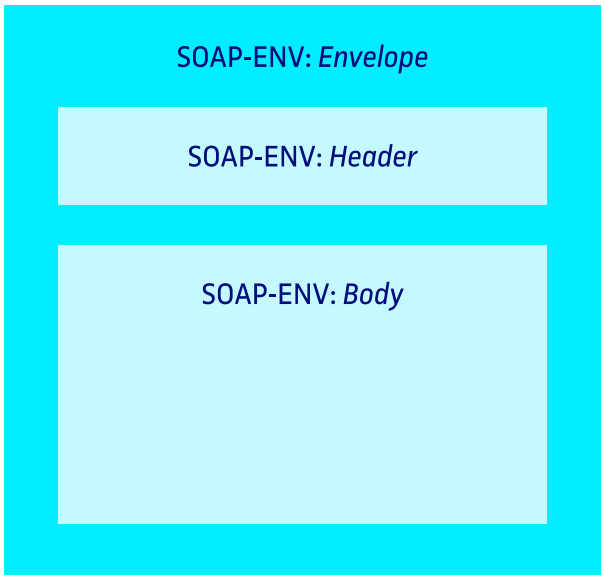
2) Regles de comunicació: aquí s'expressen les instàncies dels tipus de dades. Les definicions d'aquests tipus de dades són independents del llenguatge de programació, ja que estan basades en XSD. Poden ser simples (*int*, *string* i *date*) o compostos (*structs* i *arrays*).

3) Estils de comunicació: poden estar basats en RPC (crides a procediments remots) o orientats als missatges (document).

És un protocol força popular, ja que fa servir el transport HTTP i garanteix que qualsevol client amb un navegador amb internet pugui contactar amb un servidor remot.

Com ja hem explicat, el sobre conté les capçaleres i el cos (vegeu figura 18). Què podem incloure dins d'aquestes parts?

Figura 18. Especificació SOAP



a) Sobre (*envelope*): és l'element que trobem a l'arrel i identifica el missatge. Jeràrquicament, està a un nivell per sobre de la capçalera i el cos.

b) Capçaleres (*headers*): és un element opcional. Es fa servir per dotar d'extensions al missatge per informar com s'ha d'enviar el missatge. L'estructura de la informació inclosa és lliure i l'ha de poder entendre tant l'emissor com el receptor del missatge.

c) Cos (*body*): aquí trobarem pròpiament les dades concretes del missatge que volem enviar. En alguns casos, podem trobar com a fill l'element *fault* destinat a informar d'algun possible error en el cas que es produeixi.

A continuació, mostrem un exemple de missatgeria SOAP on es pot veure que un client demana a un proveïdor el temps actual d'una ciutat.

Petició

```
POST /Weather HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

    <soap:Body xmlns:m="http://www.example.org/weather">
        <m:GetCurrentWeather>
            <m:CityName>Barcelona - Spain</m:CityName>
        </m:GetCurrentWeather>
    </soap:Body>

</soap:Envelope>
```

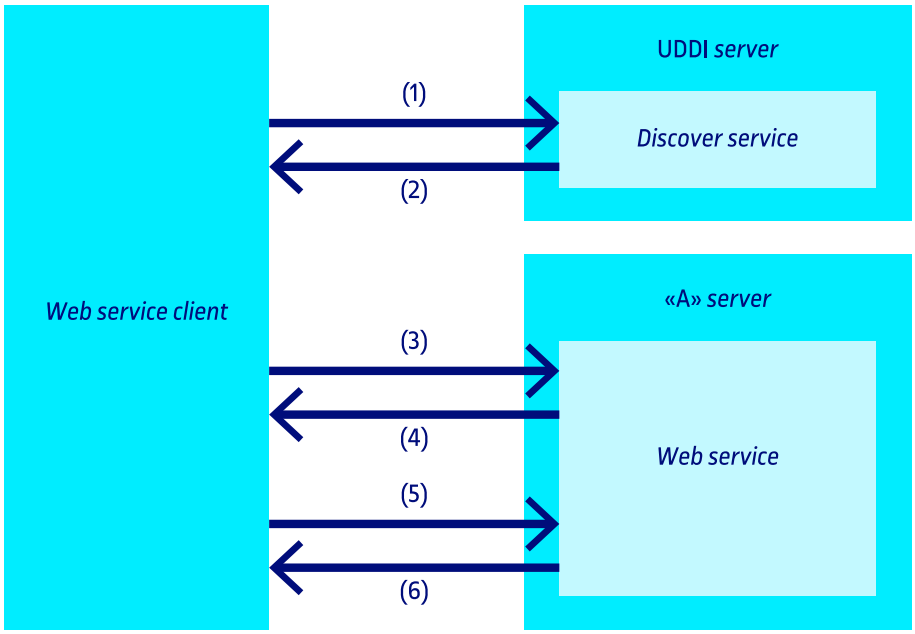
Resposta

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

    <soap:Body xmlns:m="http://www.example.org/weather">
        <m:GetCurrentWeatherResponse>
            <m:temp>19.8</m:temp>
            <m:main>Clear</m:main>
            <m:description>Clear Sky</m:description>
            <m:windSpeed>1.5</m:windSpeed>
        </m:GetCurrentWeatherResponse>
    </soap:Body>

</soap:Envelope>
```

4.3.4. Flux de funcionament d'un servei web amb SOAP

Continuant amb l'exemple anterior i a mode de resum, detallarem les diferents accions que s'executen quan una aplicació vol consultar un servei web d'un proveïdor. L'exemple que farem servir és el del client que vol saber el temps actual de la ciutat de Barcelona. Es pot veure amb detall a la figura 19.

Figura 19. Flux de funcionament d'un servei web amb SOAP



1) El client pregunta al servidor UDDI on pot trobar un servei del temps.

2) El servidor li respon indicant quin proveïdor té aquest servei.

3) El client contacta amb el proveïdor per saber com li ha de preguntar.

4) El proveïdor envia al client un arxiu WSDL amb la descripció del servei que necessita.

5) El client ara ja pot fer la petició via SOAP invocant a la funció `GetCurrentWeather` amb el paràmetre `Barcelona - Spain`.

6) El proveïdor respon amb un missatge SOAP de resposta amb la informació demanada. El camp on té la resposta és `GetCurrentWeatherResponse`.

4.3.5. REST (*representational state transfer*)

Una nova tendència que va néixer a partir de l'any 2000, que apareix motivada per la complexitat que sovint incorpora la pila d'especificació dels serveis web, és l'anomenada arquitectura REST.

REST és qualsevol interfície entre sistemes que funciona sobre el protocol HTTP i permet obtenir dades i generar operacions en diferents formats (XML, JSON, etc.).

Val a dir que és una arquitectura orientada més al recurs i no al servei. Actualment i gràcies a la seva senzillesa i eficiència, es fa servir sobretot en entorns IoT, dispositius mòbils i *frameworks* JS, ja que permet l'intercanvi d'informació en format JSON.

Entre les seves **característiques**, trobem les següents:

- **Client/Servidor**. Està basat en l'ús d'aquesta arquitectura. El servidor conté els recursos i el client pot fer operacions amb aquests.

- **Sense estat**. El client ha d'enviar tota la informació necessària perquè el servidor pugui processar la petició.

- **Recurs identificat universalment**. En un sistema REST, cada recurs és identificat amb una única URI.

- **Operacions ben definides**. Com que el protocol de transport que fa servir és l'HTTP, REST fa ús de les operacions que ja té definides aquest protocol per interactuar amb el recurs. Aquestes operacions són **POST** (crear), **GET** (llegir i consultar), **PUT** (editar) i **DELETE** (eliminar).

- **Hipermèdia**. En alguns casos, quan el client fa una petició al servidor, aquest pot retornar hipervincles de navegació a altres recursos. Aquest principi es coneix amb el nom d'HATEOAS (*hypermedia as the engine of application state*).

Seguint amb l'exemple del temps d'una ciutat que hem explicat amb SOAP, farem la representació mitjançant REST:

Petició

```
GET /api/v1/weather/locations/es/barcelona HTTP/1.1
Host: www.example.org
```

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: nnn
{ temp: 19.8, main: "Clear", description: "Clear Sky", wind_speed: 1.5 }
```

Com podeu veure, l'ús d'una API REST és força simple i facilita molt la separació entre el client i el servidor. Això comporta tenir més flexibilitat i escalabilitat quan haguem de fer qualsevol canvi. Per altra banda, SOAP avantatja REST en temes de seguretat, estandardització i extensibilitat.

L'ús d'un o l'altre dependrà de l'aplicació que caldrà implementar. Actualment, SOAP es fa servir més en aplicacions empresarials, aplicacions d'alta seguretat, entorns distribuïts, sector financer, en definitiva, organitzacions més grans que necessiten una alta seguretat en què no importa sacrificar una mi-

ca l'eficiència. En canvi, l'arquitectura REST s'està fent servir més en entorns IoT, xarxes socials, aplicacions de dispositius mòbils i en organitzacions més petites.

5. Arquitectura de microserveis

L'arquitectura de microserveis és considerada una evolució de l'arquitectura orientada als serveis (SOA). Molts també la consideren una forma més d'implementar SOA. En aquest apartat veurem quins avantatges aporta una arquitectura de microserveis respecte d'una SOA i les diferències més essencials entre els dos tipus d'arquitectura.

Una arquitectura de microserveis implementa un model basat en petits serveis que es comuniquen de forma molt superficial entre ells. Cada microservei executa els seus propis processos i es desplega de forma independent, normalment, amb mecanismes automatitzats.

Com podem veure, són arquitectures molt semblants però que difereixen d'alguns detalls que explicarem a continuació.

5.1. Arquitectura SOA contra arquitectura de microserveis

Les diferències més rellevants respecte d'una arquitectura SOA tradicional són:

- Els microserveis estan **completament desacoblats** entre si. Un microservei ha de poder funcionar de forma autònoma i independent de la resta, sense importar si la resta de serveis estan operant o no. En l'arquitectura SOA, en canvi, es disposa d'un cert acoblament, tot i que normalment és molt baix.

- Un microservei ha de disposar d'un **emmagatzematge independent** de la resta, oposadament a l'arquitectura SOA que normalment els serveis accedeixen al mateix emmagatzematge.

- En SOA els serveis s'exposaven mitjançant esquemes i contractes (WSDL), en canvi, els microserveis acostumen a exposar-se mitjançant una **API RESTful**.

- Els microserveis es poden **escalar per separat**. Amb SOA, calia escalar tota l'aplicació per poder assignar més recursos.

- Normalment en SOA, per oferir serveis més complexes, hi ha un component encarregat d'orquestrar i coordinar els múltiples serveis individuals. En canvi, en l'arquitectura de microserveis, en lloc d'haver-hi orquestració, s'acostuma a parlar de **coreografia de serveis**, en què cada servei sap com actuar en cada moment sense tenir cap component que els coordini.

- L'arquitectura de microserveis és més **tolerant a fallades**, si un microservei cau, la resta no ha de veure's afectat. Amb l'arquitectura SOA, tenim un punt d'errada únic: el bus de servei empresarial (ESB), un mal funcionament afecta tot el sistema.

Figura 20. Arquitectura SOA

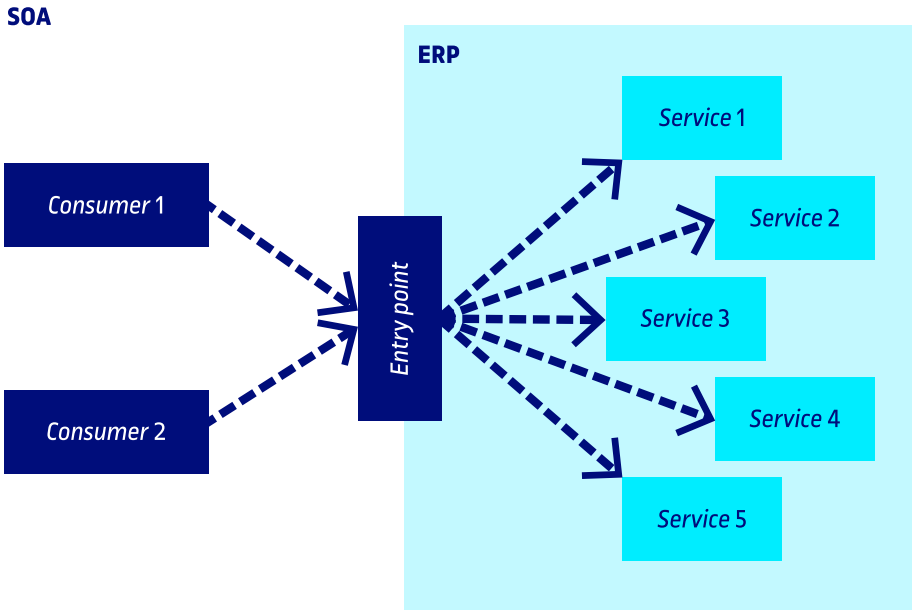
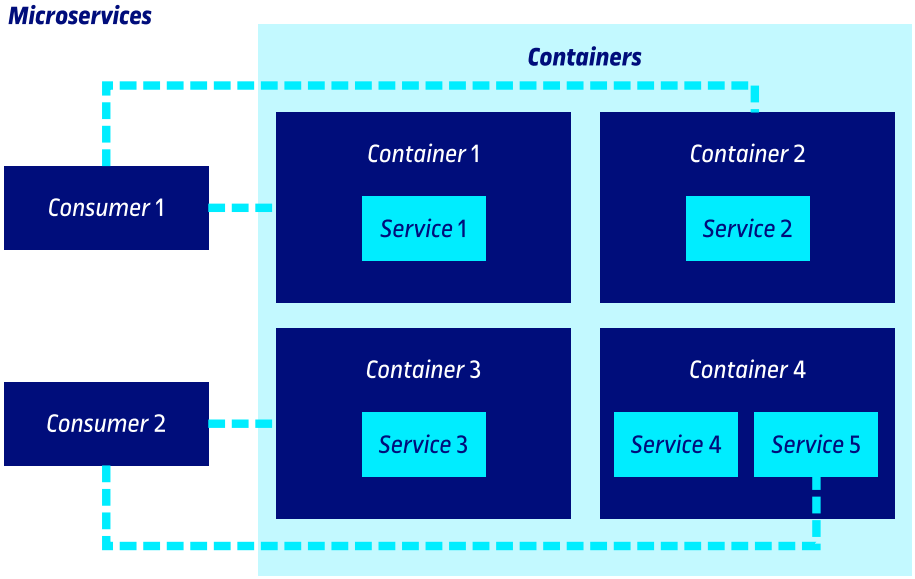


Figura 21. Arquitectura de microserveis



A les figures 20 i 21 es pot veure la diferència entre les dues arquitectures. En l'arquitectura de microserveis, un contenidor representa un entorn d'execució independent on s'executa cada servei.

5.2. Avantatges i desavantatges dels microserveis

Què ens aporta aquest nou model?

- Els microserveis es poden desplegar, mantenir i monitoritzar de forma independent. Cada microservei el pot portar un equip de programadors diferent. Aquest equip no cal que conegui com funcionen la resta de microserveis. És fàcil actualitzar aquest component a una nova versió i poder revertir a la versió anterior si alguna cosa no va bé.

- Cada microservei es pot implementar amb un llenguatge de programació o tecnologia diferent que s'adapti millor a la funció que representa. Això permet tenir diversitat tecnològica i més versatilitat.

- Es pot actualitzar només un microservei sense afectar la resta. Això és gràcies al fet que la comunicació entre els serveis es fa via API i no comparteixen cap estructura de dades.

- Es pot escalar només al microservei que es necessiti. Aquest model permet dotar de més recursos només al microservei que calgui.

- El manteniment acostuma a ser més fàcil i rentable, ja que els canvis només es produeixen en un component i no a tota l'estructura com passava amb els sistemes monolítics.

En l'actualitat podem trobar molts casos d'èxit a organitzacions com ara: Netflix, eBay, Amazon, Twitter, PayPal i molts altres llocs web que han anat evolucionant des de l'arquitectura monolítica a l'arquitectura de microserveis.

Tot i així, també podem mostrar una sèrie de **desavantatges** que caldrà tenir en compte a l'hora de fer servir l'arquitectura de microserveis.

- Com que normalment es realitza en desplegaments distribuïts, a vegades, hi ha més dificultat a l'hora de gestionar les proves que es vulguin fer.

- Si es disposa d'un nombre elevat de microserveis, es pot complicar la integració de tots ells. També haurem de disposar d'un servei per poder localitzar la resta de microserveis.

- El consum de recursos és més alt, ja que cada microservei conté la seva base de dades i els seus propis recursos.

- El cos d'implementació és més elevat. Sobretot a causa dels costos de les llicències dels productes o aplicacions de tercers.

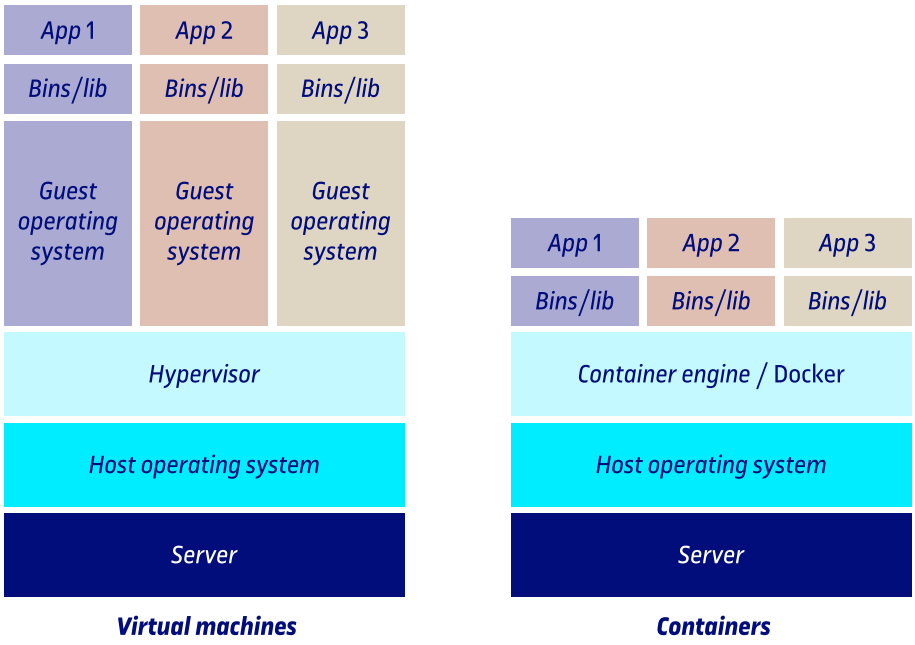
5.3. Els contenidors

Com hem vist en una figura 21, quan volem implementar una arquitectura de microserveis, s'acostuma a fer ús d'una tecnologia que ens facilita molt la feina: els contenidors. Els contenidors van ser dissenyats per poder executar funcionalitats mitjançant peces petites i això casa directament amb l'arquitectura de microserveis.

Un contenidor ens permetrà encapsular una aplicació amb tot el necessari per poder executar-la: codi, eines, biblioteques, configuracions, etc. S'executa com un procés aïllat i comparteix el sistema operatiu amb altres contenidors. A més, la seva execució serà fiable i consistent, independentment de l'entorn en què es realitzi.

Els contenidors són diferents al concepte de virtualització tradicional. A la figura 22 podem veure les diferències:

Figura 22. Màquines virtuals contra contenidors



Com es pot veure a la figura 22, si en un entorn virtual tradicional cal virtualitzar el sistema operatiu Guest per sobre del sistema Host mitjançant l'hipervisor, en un entorn de contenidors, no cal. El contenidor s'executa en el sistema operatiu Host sense la necessitat de tenir un sistema operatiu propi, podríem dir que els contenidors comparteixen els recursos del sistema operatiu mitjançant el motor de contenidors.

Aquest nou model fa que els contenidors siguin molt més lleugers respecte del model tradicional de màquines virtuals.

Contenidors

Exemples d'aquestes tecnologies poden ser VmWare o VirtualBox com a hipervisors i Docker com a motor de contenidors.

Actualment l'ús de contenidors s'està estenent amb força no solament per desplegar aplicacions en producció sinó també per desplegar aplicacions en entorns de desenvolupament. Així, cada membre de l'equip de desenvolupadors pot disposar ràpidament de l'entorn per programar, cosa que permetrà fer un traspàs ràpid a entorns de proves i producció.

5.3.1. Docker

Actualment, la plataforma Docker és la més coneguda per implementar una arquitectura de contenidors. El seu codi va ser alliberat cap al març de 2013 i de mica en mica va anar adquirint popularitat. De les organitzacions que més han contribuït en el projecte trobem Microsoft, IBM, Google, Cisco i Amadeus IT Group.

En el motor de contenidors de Docker podem trobar tres components:

1) Servidor (*daemon*): és l'encarregat de fer tota la feina de construir i administrar tot el que envolta els contenidors. En definitiva, gestiona els objectes que es poden definir com ara imatges, contenidors, volums i xarxes. Representa el *daemon docker*.

2) API rest: atén les peticions que arriben al servidor i les executa sobre aquest.

3) Client: és qui fa les peticions al servidor. Normalment representa la línia de comandes amb la comanda *docker*.

El servidor i el client acostumen a estar en el mateix sistema. Tot i així, també podem trobar arquitectures separades en què el client accedeix al servidor de forma remota.

Un element important que també trobem en aquesta arquitectura és el registre. El registre pot ser públic i serveix per emmagatzemar les imatges que faran servir els contenidors. El registre públic més conegut s'anomena *Docker Hub* i conté una gran quantitat d'imatges, moltes de les quals oficials (certificades per Docker). També podem crear un registre privat per fer un ús propi, sense que sigui públic.

Per entendre tot aquest ecosistema, és important tenir clars els diferents objectes que el componen:

- **Imatge**: és una plantilla de només lectura amb una sèrie de funcionalitats que la defineixen. Contindrà totes les dependències i llibreries que posteriorment necessitarà el contenidor. Moltes imatges són creades a par-

tir d'altres imatges, partint d'una imatge base li anem afegint més capes a sobre.

- **Contenidor**: es crea en el moment que instanciem una imatge, és a dir, quan l'executem. El contenidor es defineix per la imatge que es fa servir i per les opcions de configuració que li indiquem quan s'inicia la seva execució.

- **Volum**: quan es destrueix un contenidor, també es perden totes les dades associades que té. Els volums permeten que una certa part de les dades del contenidor puguin ser persistents, tot i haver eliminat el contenidor.

- **Xarxa**: també podem definir diferents tipologies de xarxa que ens permetran interconnectar o aïllar les comunicacions contenidor-a-contenidor o contenidor-a-*host*.

Finalment, a l'hora de desplegar els nostres contenidors és important una capa que ens permeti escalar i monitoritzar la disponibilitat dels serveis que tinguem en execució. Aquesta eina rep el nom d'**orquestrador** i concretament ens permet:

- Automatitzar la tasca de desplegament a l'hora d'aixecar els serveis.
- Balanceig de la càrrega.
- Autoescalat i autoreinici de contenidors.
- Interconnexió de dades entre els diferents contenidors.
- Monitoratge de la disponibilitat dels serveis que ofereixen els contenidors.

Orquestradors

Com a exemples més coneguts d'orquestradors trobem Docker Swarm o Kubernetes.

5.4. DevOps

La paraula DevOps ve dels acrònims en anglès *development* (desenvolupament) i *operations* (operacions). Es tracta d'una metodologia de treball que va sorgir dels equips informàtics que van començar a fer servir arquitectures de microserveis.

L'objectiu principal és aconseguir productes millors i més fiables. La idea d'aquesta metodologia es basa a **unificar els dos rols** que fins ara treballaven aïllats (desenvolupament i operacions) perquè es coordinin i col·laborin fent desplegaments en producció més ràpids i freqüents. Per aconseguir això, cal que els dos equips s'esforcin per adoptar una millor comunicació i més freqüent. També és important que tots dos equips tinguin coneixement de les necessitats dels clients i pensar en com satisfer aquestes necessitats. Sovint, això fa que a vegades cada equip ampliï les seves funcions del que normalment es faria en un equip tradicional.

Un altre concepte important d'aquesta metodologia és trencar una mica amb les barreres que hi ha entre els departaments. Cal fomentar la cultura d'equip donant més transparència al que es fa a cada àrea de negoci. Per exemple, es pot donar més visibilitat al procés de desenvolupament perquè altres àrees puguin veure en quin estat està el producte.

Hem de ser conscients que aquests canvis de mentalitat i formes de fer no sempre són fàcils dur a terme i dependrà molt del tipus d'organització i de com tinguin implementats els diferents processos a l'organització. El canvi cultural és sovint molt marcat.

Altres objectius que persegueix aquesta metodologia són:

- Temps de comercialització inferior.
- Reducció de la taxa d'errors de les aplicacions.
- Tenir versions de producte amb més freqüència.
- Augment de la productivitat dels membres de l'equip.

5.4.1. El cicle de vida de DevOps

Per entendre les particularitats d'aquesta metodologia us mostrarem les diferents **fases** que la componen.

1) Pla: és la primera fase de totes i pretén recollir les aportacions dels usuaris de forma continuada. D'aquesta manera, s'aconsegueixen els requisits necessaris de l'aplicació. També caldrà prioritzar aquests requisits i definir les iteracions que s'aniran implementant.

2) Desenvolupament: és la fase on es genera el codi de l'aplicació. S'acostuma a fer en petits cicles per afavorir el procés d'entrega. També és on es construeix l'entorn de desenvolupament que es replicarà als membres de tot l'equip.

3) Integració contínua: en aquesta fase, s'integra la nova funcionalitat desenvolupada amb la resta de l'aplicació. És el moment de superar les proves i detectar possibles errors abans de desplegar la nova versió en producció. Aquesta integració es fa de forma automàtica i continuada. La idea és que cada cert temps es descarregui d'un repositori de control de versions la darrera versió del codi font, es compili (si cal) i s'executin les proves i tests. Com a sortida obtindrem un informe d'aquest procés automàtic.

4) Desplegament: és el moment de posar en producció la nova funcionalitat. Es fan servir eines que automatitzen el pas a producció. Gràcies a això, tindrem una certa traçabilitat per saber les funcionalitats que es despleguen a cada entrega i podrem tornar a la versió estable anterior de forma fàcil i segura si alguna cosa no ha anat bé.

5) Operació: un cop tenim l'aplicació en producció caldrà fer un monitoratge i seguiment tant de la càrrega del servidor com dels possibles errors que hi puguin haver. També es fan servir eines que automatitzen aquesta fase i permeten generar avisos en el cas de problemes.

Aquest cicle és iteratiu i es va executant contínuament per a cada nova funcionalitat que vulguem implementar en l'aplicació.

Com podeu veure, es basa en una sèrie d'eines que ajuden a automatitzar el cicle de vida de l'aplicació. Fa que l'equip de treball es concentri més a l'hora d'implementar les noves funcionalitats, evitant perdre el temps en tasques repetitives que no aporten beneficis.

5.4.2. Beneficis de la metodologia DevOps

L'ús d'aquesta metodologia aporta certs beneficis a les organitzacions. Els més importants són:

- **Velocitat** a l'hora de desenvolupar i desplegar el producte a producció.
- **Reducció del temps** per resoldre possibles incidències.
- **Augment de la qualitat** del producte gràcies a les automatitzacions i proves abans de fer el pas a producció, reduint el marge d'errors.
- **Resposta ràpida** enfront a nous requeriments de l'aplicació, gràcies a la integració contínua.
- Creació d'**entorns col·laboratius** entre els equips de desenvolupament i operacions.
- Producció d'**aplicacions més segures** gràcies al sistema de proves i tests que ajuden a no desplegar aplicacions que incorporin qualsevol tipus d'error.

Resum

En aquest mòdul, s'han definit algunes de les tecnologies i metodologies més importants que envolten la gestió dels serveis web i les seves aplicacions.

S'ha començat definint els inicis de la tecnologia HTTP, tant en l'àmbit històric com en les diferents fases per les quals han anat evolucionant els servidors de pàgines web, des de les pàgines estàtiques inicials fins a les aplicacions més modernes i actuals basades en la separació de diferents capes que gestionen la vista, el model de negoci i les dades.

S'ha mostrat la part tecnològica i d'arquitectura en la qual es basa el funcionament del servei web, tant pel que fa a la definició de les diferents capes que componen el protocol HTTP com la comunicació entre el client i el servidor.

S'ha tingut en compte la gestió dels serveis web des del punt de vista de l'administrador, amb un rol més de gestió i responsabilitat del web; des de l'àrea dels sistemes, encarregada de l'arquitectura física del servei web, pel que fa a la instal·lació i la configuració dels programaris necessaris per garantir un correcte funcionament del servei (servidor web, servidor Proxy/Cache, *virtual hosting*, base de dades, etc.); i des del punt de vista de l'analista/programador, encarregat del disseny i desenvolupament de les diferents capes que poden compondre l'aplicació web (vista, controlador, API, etc.), tenint en compte l'arquitectura més adient segons la finalitat de l'aplicació (costat del client, costat del servidor, pàgina única, etc.).

El model orientat al servei ofereix una nova capa en què la resta de components del sistema exposen les seves funcions (serveis) perquè puguin ser utilitzades per tercers. Una forma d'implementar aquesta nova capa és mitjançat els *web services*.

Les darreres tendències han portat a evolucionar aquesta arquitectura orientada als serveis cap a una arquitectura de microserveis. En aquest cas, cada microservei executa els seus propis processos i es desplega de forma independent. També es passa a tenir una coreografia de serveis, en què cada servei sap com actuar en cada moment sense tenir cap component que els coordini.

Hi ha una metodologia de treball anomenada DevOps que va sorgir dels equips informàtics que van començar a fer servir les arquitectures de microserveis. La idea d'aquesta metodologia es basa a unificar els dos rols que fins ara treballaven aïllats (desenvolupament i operacions) perquè es coordinin i col·laborin fent desplegaments en producció mes ràpids i freqüents.

Cal tenir en compte que les tecnologies en aquesta àrea són molt canviants, però els coneixements tècnics i metodològics adquirits en aquest mòdul us ajudaran a tenir la base necessària per a la implantació de les noves tecnologies.

Activitats

1. Connecteu-vos a un web i mireu el codi font de la pàgina mitjançant les eines de desenvolupadors incorporades en cada un dels navegadors per veure l'HTML i les crides de xarxa XHR amb què està feta. Vegeu les marques <>...</> del DOM, i també les crides a webs externes per obtenir les dades.

2. Avui dia hi ha molts *frameworks* de JavaScript per desenvolupar les vistes de les aplicacions, els quals incorporen un petit servidor web per visualitzar les pàgines que s'estan desenvolupant. Proveu d'instal·lar-ne un d'aquests *frameworks* (Vue CLI, Quasar, Angular, Ionic, etc.) seguint les indicacions de cada programari. En la majoria dels casos obtindreu una web bàsica basada en una plantilla ja incorporada. Què us sembla començar a crear una aplicació web d'aquesta manera?

3. Tal com s'ha vist en l'apartat «Docker», Docker és una de les plataformes que permet implementar l'arquitectura de contenidors. Tanmateix, disposa d'una plataforma denominada DockerHub (https://hub.docker.com/) amb diversos contenidors ja creats per les mateixes marques o per la comunitat pública. Proveu d'instal·lar un petit servidor web (tipus httpd o node) mitjançat algun dels contenidors existents, i proveu d'obrir en un navegador una petita pàgina estàtica creada per vosaltres mateixos servida per aquest servidor. Si us animeu, podeu intentar substituir aquesta pàgina estàtica per l'aplicació creada en l'activitat anterior, d'aquesta manera ja podríeu tenir una web en producció.

4. Cerqueu a internet una API pública que no requereixi autenticació. És fàcil trobar-ne moltes. Intenteu consultar-la des del mateix navegador d'internet i examineu-ne els resultats.

Glossari

CMS *f* És la sigla de *content management system*. Aplicació per crear i administrar continguts.

CSS *m* És la sigla de *cascading style sheets*. Estàndard proposat el 1996 per W3C.

DHTML *m* És la sigla de *dynamic HTML*. HTML dinàmic.

enllaç *m* Mecanisme específic per accedir a un recurs d'internet. L'utilitzen especialment les pàgines web, en un format definit `<A> <href="URL">text</A>`.
en.: *link*

HTML *m* Vegeu **llenguatge d'etiquetatge d'hipertext**.

hypertext markup language Vegeu **llenguatge d'etiquetatge d'hipertext**.

intranet *f* Xarxa interna d'una organització que pot funcionar amb TCP/IP, i a més pot fer servir els recursos d'internet, com poden ser navegadors, servidors web, etc.

link Vegeu **enllaç**.

linking Consisteix a mostrar continguts d'una pàgina diferent com a propis. Es considera una vulneració de la propietat intel·lectual.

llenguatge d'etiquetatge d'hipertext *m* Llenguatge estàndard per etiquetar documents en format d'hipertext a fi d'indicar a un navegador com ha de visualitzar un document a la pantalla de l'ordinador.
en.: *hypertext markup language*
sigla: HTML

localitzador uniforme de recursos *m* Manera única d'especificar un recurs a internet.
en.: *uniform resource locator*
sigla: URL

model en objectes per a la representació de documents *m* Model orientat a objectes que els programes i *scripts* poden usar per accedir i modificar dinàmicament el contingut, l'estructura i l'estil dels documents HTML.
sigla: DOM

servidor web *m* Aplicació que funciona en un ordinador escoltat per un port i serveix pàgines web sota demanda.

uniform resource locator Vegeu **localitzador uniforme de recursos**.

URL *m* Vegeu **localitzador uniforme de recursos**.

webmaster Responsable del web. Sota aquest nom hi ha la persona que té cura dels continguts (no ha de ser necessàriament l'administrador del web).

webmail Correu basat en web.

W3C *m* World Wide Web Consortium. El consorci que marca l'estàndard d'HTML, entre moltes altres coses. http://www.w3c.org.

XML *m* *eXtensible Markup Language* o llenguatge d'etiquetatge extensible. És un estàndard ajustat a les necessitats d'internet.

Bibliografia

Bibliografia bàsica

Daigneau, R. (2012). *Service design patterns: fundamental design solutions for SOAP/WSDL and RESTful Web services*. Boston: Addison-Wesley.

Loshin, P. (2003). *TCP/IP clearly explained* [en línia] (4a. edició). Amsterdam: Morgan Kaufmann. Disponible a: https://discovery.uoc.edu/iii/encore/record/C__Rb1050399?lang=eng.

Papazoglou, M. P. (2012). *Web services & SOA: principles and technology*. Canada: Pearson Education.

Shklar, L.; Rosen, R. (2006). *Web application architecture*. Chichester: John Wiley & Sons.

Subramanian, H. (2019). *Hands-on RESTful web API design patterns and best practices#: design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs*. Birmingham: Packt Publishing.

Wessels, D. (2001). *Web caching* [en línia]. Beijing: O'Reilly. Disponible a: https://discovery.uoc.edu/iii/encore/record/C__Rb1009104.

Zanon, D. (2017). *Building Serverless Web Applications* [en línia]. Birmingham: Packt Publishing. Disponible a: http://search.ebscohost.com.biblioteca-uoc.idm.oclc.org/login.aspx?direct=true&db=edsvle&AN=edsvle.AH31954922&site=eds-live&scope=site.

Bibliografia complementària

Puškaric, H.; Djordjevic, A.; Stefanovic, M.; Zahar Djordjevic, M. (2019). Development of Web Based Application Using Spa Architecture» [en línia]. *Proceedings on Engineering Sciences* (vol. 2, núm. 1, pàgs. 457–464). Disponible a: https://doi-org.biblioteca-uoc.idm.oclc.org/10.24874/PES01.02.044.

Webgrafia

World Wide Web Consortium (W3C)

Apache *virtual host documentation* - Apache HTTP *server version* 2.4

Server block examples, NGINX