Implantació d'un Sistema de Integració Continua

Memòria

Daniel García Magané

Enginyeria en Informàtica

Miquel Colobran Huguet

Curs 2017-18 (1er semestre)

DEDICATORIA I AGRAÏMENTS

En primer lloc, agrair a la meva dona Jenny per ser el suport que necessitava, animant-me cada dia, per a dur a terme aquest PFC. Li dono les gràcies per la seva paciència en tots aquests anys i caps de setmana de PACs tancats a casa sense poder sortir. Gràcies per intentar fer de mi una millor persona cada dia. Gràcies de tot cor.

En segon lloc, a les meves dues filles bessones Abril i Sira, us demano perdó per haver estat desaparegut aquestes últims mesos, ho sento. Us dono les gràcies per fer-me sentir millor en els pitjors moments amb el vostre somriure i alegria infinits.

Als meus pares per donar-me l'oportunitat d'estudiar i recolzar-me sempre que els he necessitat. Gràcies.

I finalment al meu professor de PFC, en Miquel Colobran, per la seva gran ajuda sense la qual no hauria sigut possible realitzar aquest treball. Ha sigut un plaer treballar amb tu.

Moltes gràcies.

Descripció del Treball

Avui en dia en les empreses dedicades al desenvolupament de software, un gran avantatge competitiu és la capacitat de lliurar els seus productes i serveis de forma ràpida, segura, lliure d'errors i respectant els ajustats temps de mercat (*time to market*).

El desenvolupament de software amb metodologies Àgils es caracteritza pel lliurament constant de productes MVP (*Minimum Value Product*). La capacitat de poder oferir ràpidament als clients les noves funcionalitats desitjades i detectar l'abans possible qualsevol error és l'objectiu de qualsevol organització.

Disposar d'eines per a gestionar les diferents versions del producte així com administrar els entorns de desenvolupament i producció de la forma més automatitzada possible, donarà sense dubte un gran avantatge i noves oportunitats de negoci a les empreses.

En aquest PFC es vol presentar una proposta de treball completa per descriure el procés de transformació d'una empresa tipus. Es partirà des d'un entorn sense un correcte control de la gestió del desenvolupament de projectes informàtics cap a un entorn totalment automatitzat en termes de Integració Continua.

Es tractaran aspectes relacionats amb la forma de treballar i de gestionar els projectes informàtics. S'analitzaran entre d'altres, temes importants com l'organització dels repositoris de codi font, les eines disponibles per a validar, compilar i publicar les aplicacions, les metodologies i els procediments de treball. També es consideraran els recursos de xarxa, de software i de maquinari necessaris per a integrar-se dins de l'organització.

S'analitzaran les diferents estratègies i productes del sector i es descriurà una possible implementació.

Índex

Índex de figures 6
1 Introducció 8
1.1 Justificació i Context del projecte 8
1.2 Objectius del projecte 8
1.3 Organització i Planificació del treball 9
1.4 Enfoc i metodologia seguida 15
1.5 Requisits del projecte. 15
1.6 Productes obtinguts 16
1.7 Breu descripció dels capítols de la memòria 16
2 Introducció a la Integració Continua 17
2.1 Què és la Integració Continua ? 17
2.2 Breu antecedents històrics de la IC 17
2.3 Principals components d'un sistema de IC 18
2.4 Principals fluxos i interaccions en un sistema d'IC 20
2.5 Característiques i pràctiques d'un sistema d'IC 21
2.6 Altres pràctiques relacionades amb la IC 27
3 Anàlisis de productes IC en el mercat 30
3.1 Jenkins 30
3.2 Bamboo 35
3.3 Team City 42
3.4 CircleCI 47
3.5 GoCD 52
3.6 Resum i comparativa de les eines 56
3.7 Justificació del producte triat 58
4 Implementació d'un Sistema d'Integració Continua 61
4.1 Descripció i anàlisis de l'entorn inicial 61
4.2 Anàlisis d'entorns 62
4.3 Anàlisis de l'estructura del codi font 65
4.4 Anàlisis de recursos necessaris 73
4.5 Instal·lació d'un repositori de Codi Font 73
4.6 Instal·lació de l'eina d'Integració Continua 76
4.7 Metodologia desenvolupament 85
4.8 Formació d'equips 89
4.9 Monitoratge 90

5 Conclusions 92
6 Bibliografía 93

Índex de figures

Figura 1: Diagrama de Gantt resumit 10
Figura 2: Diagrama de Gantt complet 11
Figura 3: Components i interaccions bàsiques a un sistema de IC (font) 21
Figura 4: Relacions i dependències dins el flux de IC (font) 25
Figura 5: Exemple de flux de treball i notificacions....26
Figura 6: Altres pràctiques relacionades (font)....27
Figura 7: Integracio Continua Vs Lliurament Continu....28
Figura 8: Continuos Delivery Vs Continuous Deployment....28
Figura 9: Exemple de configuració de tasques en Pipelines....31
Figura 10: Jenkins - Edició i visualització de *pipelines* 32
Figura 11: Jenkins - Visualització d'una execució fallida de les *pipelines* 32
Figura 12: Jenkins - Visualització de l'activitat històrica d'una pipeline 32
Figura 13: Jenkins - Visualització del estat de les branques 33
Figura 14: Exemple escenari plataforma Jenkins 34
Figura 15: Ecosistema de Jenkins i connexió a altres plataformes (font)....35
Figura 16: Model arquitectura bàsica Bamboo....36
Figura 17: Exemple de configuració de jobs en Bamboo (font) 37
Figura 18: Relació entre *stage*, *task* i job a Bamboo....38
Figura 19: Panell d'estat del projecte a Bamboo (font)....38
Figura 20: Bamboo *Wallboard* 39
Figura 21: Flux de pre-validació de TeamCity....44
Figura 22: Arquitectura TeamCity per a pre-validar el codi 44
Figura 23: Arquitectura bàsica CircleCI....48
Figura 24: CircleCI visualització dels detalls d'un *Workflow*....49
Figura 25: CircleCI MacOs licensing....52
Figura 26: Estructura bàsica de GoCD....53
Figura 27: Exemple composició d'una pipeline en GoCD....53
Figura 28: Value Stream Map en GoCD 54
Figura 29: GoCD Dashboard 54
Figura 30: GoCD utilitat comparació de builds 55
Figura 31: Taula resum cost servei de suport de GoCD....56
Figura 32: Conjunt de *sandboxes* – entorns en un entorn de treball Àgil....62
Figura 33: Exemple bàsic estructura projecte en VisualStudio 68
Figura 34: Exemple de descriptor Nuget nuspec 69
Figura 35: Configuració de l'Assembly Information del projecte en VisualStudio....70
Figura 36: Flux de treball amb paquets NuGet....70
Figura 37: Arbre de dependències de llibreries d'un projecte .NET 70
Figura 38: Gestor de repositoris NuGet de VisualStudio 72
Figura 39: NuGet Package solution manager del VisualStudio 72
Figura 40: Taula resum servidor plataforma IC....73
Figura 41: GitLab menú de navegació fitxers d'un projecte. 75
Figura 42: Bamboo menú configuració d'agents....78

Figura 43: Bamboo configuració d'executables i version de Java 79
Figura 44: Bamboo task Wizard 81
Figura 45: Llistat de tasques d'un Job genèric de Release en Bamboo 81
Figura 46: Bamboo tasques pre configurades per a validar tests 82
Figura 47: Exemple configuració NuGet Repositery Demo en Bamboo 83
Figura 48: Bamboo triggers automàtics de deployment 83
Figura 49: Models GitFlow amb feature 86
Figura 50: Resum general model de branques en GitFlow 87
Figura 51: Exemple pantalles plugin GitFlow per a VisualStudio 88

1 Introducció

1.1 Justificació i Context del projecte

Actualment les empreses de desenvolupament de software poden triar moltes eines del mercat per a gestionar el cicle de vida del software des de la seva creació fins al lliurament al client final. El conjunt d'eines triades i com s'orquestren entre si i les metodologies de treball que adopten els equips de desenvolupament determinaran de crucialment la forma i la capacitat de crear i lliurar els seus productes.

Entre molts sistemes i aplicacions de millora que poden fer servir les empreses, la Integració Continua practicada correctament en els camps de: mentalitat, disciplina de treball i implementació tècnica, pot aportar una millora significativa a la productivitat de l'organització.

Hi ha solucions i sistemes de Integració Continua de totes les mides, necessitats i costos que poden permetre d'una forma senzilla i relativament econòmica implantar un sistema de lliurament continu.

Cada cop més, totes les grans empreses del sector aposten per aquesta pràctica, sent fonamental en entorns en què es necessiten temps de resposta molt baixos.

Amb aquest projecte es vol fer una introducció en la temàtica a fi de poder identificar punts en comú que es poden donar en entorns reals. Aplicant els conceptes exposats es podrà tenir una pauta per on començar a treballar.

1.2 Objectius del projecte

Els principals objectius d'aquest treball es basen en conèixer quines són les principals característiques i avantatges de la Integració Continua dins d'un procés de desenvolupament de software.

A la vegada també es volen identificar els problemes clàssics d'una empresa tipus, que no fa servir eines ni metodologies relaciones amb la Integració Continua, que es poden donar en la seva operativa diària i que afecten la seva capacitat productiva. Es mostraran els efectes i impactes que suposa aquesta transformació, així com les principals tasques i activitats necessàries per dur-lo a terme.

Els següent punts sintetitzen els principals objectius del treball:

1.2.1 Objectius generals

- Entendre les necessitats de gestió del cicle de vida de les aplicacions en una organització. Cal veure com es gestionen i s'organitzen els desenvolupadors a fi de treballar tots junts en les diferents solucions de codi font per a aconseguir els objectius.
- Entendre que és Integració Continua.
- Conèixer les diferents eines i plataformes de mercat disponibles. Cada fabricant ofereix la seva versió i implementació pròpia que pot encaixar més o menys depenent de les necessitats reals de cada empresa i del deute tecnològic que arrosseguen.

- Conèixer els avantatges que suposa la implementació d'una plataforma que manegui de forma automatitzada la Integració Continua.
- Identificar els impactes de dur a terme aquesta transformació. Per a poder implementar un sistema d'Integració Continua caldrà canviar molts aspectes tècnics, procediments i infraestructura que en molts casos pot suposar seriosos impactes i problemes en una organització.
- Millorar la productivitat de l'empresa i la qualitat de les aplicacions desenvolupades.

1.2.2 Objectius parcials

Els objectius parcials per assolir-lo són els següents :

- Fer una recerca intensiva dels recursos disponibles al mercat.
- Aplicar el coneixement en informació obtinguts per a la presa de decisions.
- Introduir noves mesures de control addicionals i monitoreig de les noves infraestructures.
- Formació de la majoria del personal involucrat en el procés de desenvolupament del software: des dels propis programadors, com a caps de projecte, àrees de màrqueting, administradors de sistemes, etc.

1.3 Organització i Planificació del treball

En aquest apartat es detalla l'organització del projecte, distribuïda en diferents tasques, així com el calendari i les diferents fites principals.

1.3.1 Descripció de les tasques

La planificació del treball PFC estarà enfocada a assolir els principals objectius del PFC anteriorment esmentats així per a complir amb la temporització de les PACS i fites proposades en l'assignatura. Els principals eixos de la planificació es basaran en:

- Fer una primera recerca de tota la informació necessària per a definir i comprendre que és la Integració Continua.
- Amb la informació obtinguda fer una anàlisi i comparativa de les diferents solucions del mercat per a prendre una decisió.
- Descriure el cas d'una hipotètica implementació amb la solució triada en una empresa tipus.
- Creació del manual de gestió corresponent.
- Conclusions.

1.3.2 Taula de dates resumit

A continuació es descriuen les principals dates del projecte que marcaran el desenvolupament del projecte i dels corresponents lliuraments:

Implementació d'un sistema de Integració Continua			
Setmana	**Data**	**Tipus fita**	**Descripció**
1	20/09/2017	Externa	Inici de curs
1	20/09/2017	Externa	Publicació enunciat PAC 1 (Plat de Treball)
3	02/10/2017	Interna	Lliurament esborrany PAC 1 (Pla de Treball)
3	06/10/2017	Interna	Lliurament PAC 1
8	08/11/2017	Interna	Lliurament esborrany PAC 2

8	10/11/2017	Interna	Lliurament PAC 2 (entre 40%-60% PFC)
12	08/12/2017	Interna	Lliurament esborrany PAC 3
13	15/12/2017	Interna	Lliurament PAC 3 (entre 80%-90% PFC)
16	03/01/2017	Interna	Lliurament Final Memòria + Presentació
16	08/01/2017	Externa	Inici Debat Virtual (per confirmar)
17	14/01/2017	Externa	Fi Debat Virtual (per confirmar)
	XX/01/2017	Externa	Tancament Projecte (per confirmar)

1.3.3 Diagrama de Gantt resumit

Per a realitzar la planificació de les tasques d'elaboració d'aquest PFC s'ha fet servir Microsoft Project 2016. A continuació es pot visualitzar el calendari resumit i tasques per dur-lo a terme:

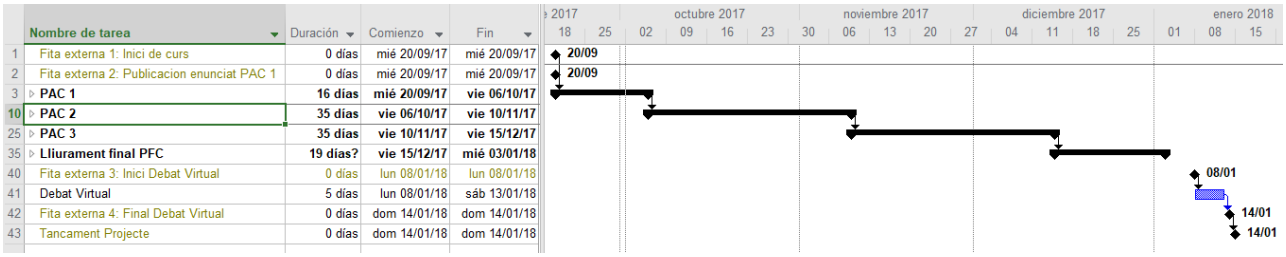


Figura 1: Diagrama de Gantt resumit

Tota la planificació del projecte s'agrupa en activitats corresponents a cadascun dels 4 lliurables de l'assignatura (PAC1, PAC2, PAC3 i Lliurament final). En la següent figura es pot veure el conjunt total de subtasques que componen cadascun dels lliurables.

1.3.4 Diagrama de Gantt complet

Id	Nombre de tarea	Duración	Comienzo	Fin
1	Fita externa 1: Inici de curs	0 días	mié 20/09/17	mié 20/09/17
2	Fita externa 2: Publicacion enunciat PAC 1 (Pla de treba	0 días	mié 20/09/17	mié 20/09/17
3	**PAC 1**	**16 días**	**mié 20/09/17**	**vie 06/10/17**
4	Descàrrega material assignatura i revisió	1 día	mié 20/09/17	jue 21/09/17
5	Valoració inicial contingut i definició abast del PFC	4 días	jue 21/09/17	lun 25/09/17
6	Redacció Pla de Treball	7 días	lun 25/09/17	lun 02/10/17
7	Lliurament esborrany Pla de Treball	0 días	lun 02/10/17	lun 02/10/17
8	Correccions i modificacions Pla de Treball	4 días	lun 02/10/17	vie 06/10/17
9	Lliurament final PAC 1 - Pla de Treball	0 días	vie 06/10/17	vie 06/10/17
10	**PAC 2**	**35 días**	**vie 06/10/17**	**vie 10/11/17**
11	**Anàlisi Integració Continua**	**20 días**	**vie 06/10/17**	**jue 26/10/17**
12	Recerca i recopilació informació de solucions del mercat	8 días	vie 06/10/17	sáb 14/10/17
13	Redacció Introducció Integració Continua	3 días	sáb 14/10/17	mar 17/10/17
14	Anàlisis i comparatives solucions del mercat	8 días	mar 17/10/17	mié 25/10/17
15	Integració en la memòria	1 día	mié 25/10/17	jue 26/10/17
16	**Implementació solució triada**	**15 días**	**jue 26/10/17**	**vie 10/11/17**
17	Descripció entorn de partida	1 día	jue 26/10/17	vie 27/10/17
18	Identificació tasques a realitzar	2 días	vie 27/10/17	dom 29/10/17
19	Anàlisis Entorns de publicació i desenvolupament	4 días	dom 29/10/17	jue 02/11/17
20	Anàlisis Organització codi i repositoris	4 días	jue 02/11/17	lun 06/11/17
21	Integració en la memòria	2 días	lun 06/11/17	mié 08/11/17
22	Liurament esborrany PAC 2	0 días	mié 08/11/17	mié 08/11/17
23	Correccions i modificacions	2 días	mié 08/11/17	vie 10/11/17
24	Lliurament final PAC 2	0 días	vie 10/11/17	vie 10/11/17
25	**PAC 3**	**35 días**	**vie 10/11/17**	**vie 15/12/17**
26	**Implementació solució triada 2**	**13 días**	**vie 10/11/17**	**jue 23/11/17**
27	Anàlisis eina de compilació i deployment	8 días	vie 10/11/17	sáb 18/11/17
28	Anàlisis Formació equip tècnic	3 días	sáb 18/11/17	mar 21/11/17
29	Anàlisis monitoreig	2 días	mar 21/11/17	jue 23/11/17
30	Redacció Manual de Gestió del Producte	10 días	jue 23/11/17	dom 03/12/17
31	Integració en la memòria	5 días	dom 03/12/17	vie 08/12/17
32	Liurament esborrany PAC 3	0 días	vie 08/12/17	vie 08/12/17
33	Correcions	5 días	vie 08/12/17	mié 13/12/17
34	Lliurament final PAC 3	0 días	vie 15/12/17	vie 15/12/17
35	**Lliurament final PFC**	**19 días?**	**vie 15/12/17**	**mié 03/01/18**
36	Redacció conclusions	1 día?	vie 15/12/17	sáb 16/12/17
37	Elaboració Presentació virtual	12 días	sáb 16/12/17	jue 28/12/17
38	Consolidació i redacció final memòria	5 días	jue 28/12/17	mar 02/01/18
39	Liurament final Memoria + Presentació	0 días	mié 03/01/18	mié 03/01/18
40	Fita externa 3: Inici Debat Virtual	0 días	lun 08/01/18	lun 08/01/18
41	Debat Virtual	5 días	lun 08/01/18	sáb 13/01/18
42	Fita externa 4: Final Debat Virtual	0 días	dom 14/01/18	dom 14/01/18
43	Tancament Projecte	0 días	dom 14/01/18	dom 14/01/18

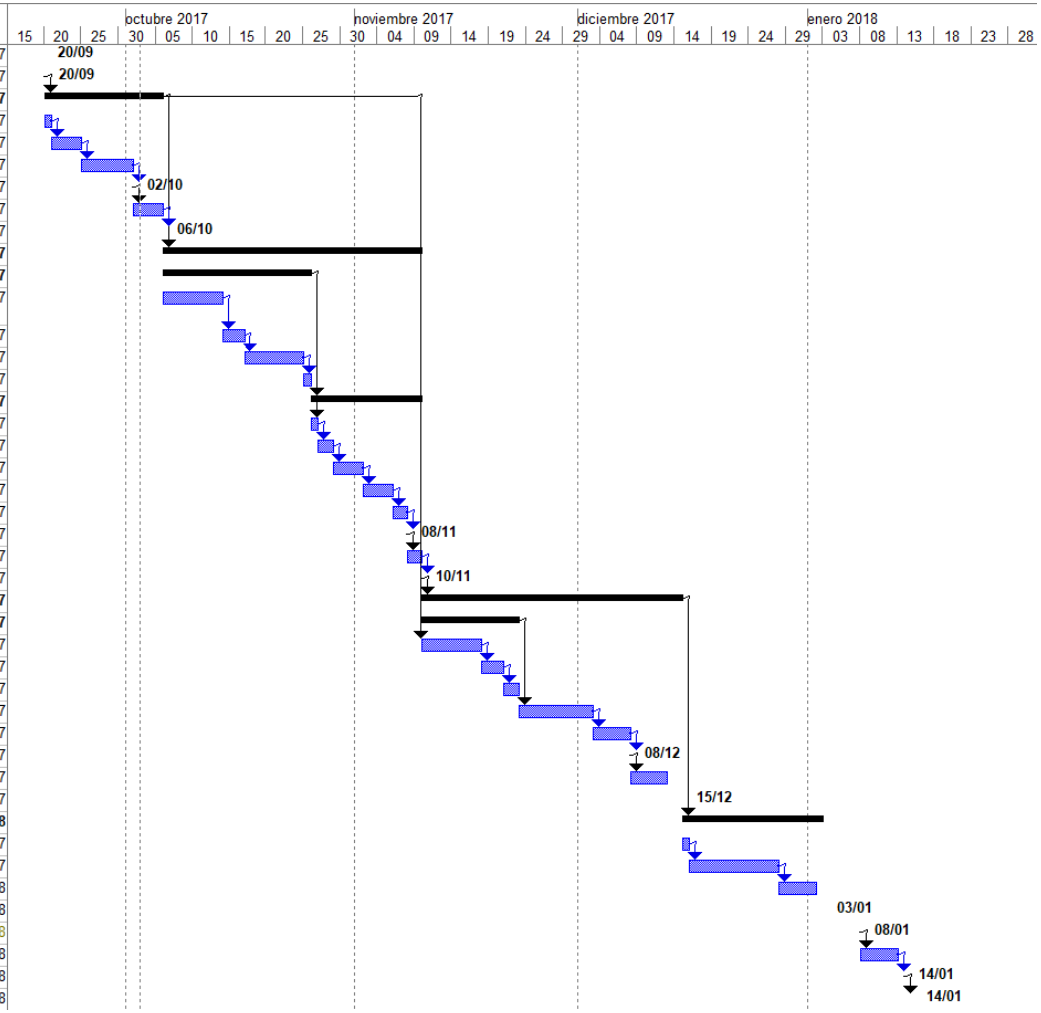


Figura 2: Diagrama de Gantt complet

1.3.5 Descripció de les tasques

En els següents punts, es detallarà per cada tasca les dates d'inici i final planificades, i una petita descripció del que es tractarà en cada tasca. Al igual que la planificació es seguirà una classificació enfocada en cadascuna de les PACS i Lliuraments finals.

1.3.5.1 Activitats PAC 1 – Pla de treball

Les activitats d'aquest bloc aniran enfocades a entendre que s'ha de fer, com es vol fer, tenir clars els materials disponibles i el més important: realitzar la planificació en tasques del projecte.

Nom tasca	**Descàrrega material assignatura i revisió**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	20/09/2017	21/09/2017	1	PAC 1
Descripció	Descarregar tot el material de l'aula, recursos i resta d'informació. Fer una lectura i classificar tota la informació per a conèixer de quins recursos es disposa des de l'inici del curs.			

Nom tasca	**Valoració inicial contingut i definició abast del PFC**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	21/09/2017	25/09/2017	4	PAC 1
Descripció	Amb tot el material i informació disponible fer la valoració i tria de com serà el projecte, la temàtica específica, format, quins punts es volen tractar, com es volen organitzar, etc per a tenir clar quin serà l'abast que es vol aconseguir al PFC.			

Nom tasca	**Redacció Pla de Treball**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	25/09/2017	02/10/2017	7	PAC 1
Descripció	Fer la redacció del Pla de Treball, seguint les indicacions i el material i formats proporcionats a l'aula. Fer l'estructuració clara de les tasques a realitzar i decidir la distribució temporal de cadascuna d'elles. Aquest serà el punt de partida clau per a fer una bona planificació del PFC			

Nom tasca	**Correccions i modificacions Pla de Treball**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	02/10/2017	06/10/2017	4	PAC 1
Descripció	Un cop lliurat l'esborrany del Pla de Treball, disposar de marge per a possibles correccions i modificacions necessàries per al correcte lliurament final del Pla de Treball en la PAC 1			

1.3.5.2 Activitats PAC 2

En la PAC 2, es tractarà el gruix d'analitzar a fons tota la informació sobre la Integració Continua i quines solucions hi ha al mercat. Es començaran a dur a terme els anàlisis necessaris en el cas específic que el vol implementar i s'afegiran a la memòria les primeres propostes concretes.

Nom tasca	**Anàlisi Integració Continua / Recerca i recopilació informació de solucions del mercat**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	06/10/2017	14/10/2017	8	PAC 2
Descripció	Fer una recerca en tots els recursos web possibles amb la finalitat de recollir la màxima informació relativa a la Integració Continua com les principals tècniques i implementacions. Recollir informació més específica sobre com implementen aquestes tècniques productes específics del mercat i identificar els possibles estàndards al respecte. S'intentarà avançar al màxim aquesta activitat si és possible durant els dies previs.			

Nom tasca	**Anàlisi Integració Continua / Redacció Introducció Integració Continua**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	14/10/2017	17/10/2017	3	PAC 2

Descripció	Elaborar la part del primer capítol de la memòria dedicat a explicar i introduir amb les explicacions corresponents que és la Integració Continua, per tal de poder tindre la referència i context necessaris per a la resta del PFC.

Nom tasca	**Anàlisi Integració Continua / Anàlisis i comparatives solucions del mercat**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	17/10/2017	25/10/2017	8	PAC 2
Descripció	Amb la informació recollida en les tasques anteriors redactar l'informe de com implementa la Integració Continua cadascun dels productes triats en la recerca. Fer anàlisis dels punts positius i negatius de cadascun, quins encaixen millor depenent del tipus de necessitat i entorn/tecnologies de les empreses, així com les justificacions corresponents que calguin.			

Nom tasca	**Anàlisi Integració Continua / Integració en la memòria.**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	25/10/2017	26/10/2017	1	PAC 2
Descripció	Arribats a aquest punt d'estar prop de lliura la PAC2, cal revisar acuradament que tots els continguts elaborats fins ara estan correctament integrats al document de la memòria.			

Nom tasca	**Implementació solució triada / Descripció entorn de partida**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	26/10/2017	27/10/2017	1	PAC 2
Descripció	Fer la descripció completa de l'entorn inicial en què es troba una empresa tipus. Descriure com gestionen el desenvolupament de software, el lliurament i la seva infraestructura. Donar els detalls necessaris per a que es puguin després identificar les accions a realitzar per tal d'aplicar les millores objectiu d'aquest PFC.			

Nom tasca	**Implementació solució triada / Identificació tasques a realitzar**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	27/10/2017	29/10/2017	2	PAC 2
Descripció	Conegut l'entorn inicial i definits els objectius, es farà un llistat de les principals accions i tasques a dur a terme en l'empresa tipus amb la finalitat d'aconseguir els objectius.			

Nom tasca	**Implementació solució triada / Anàlisis Entorns de publicació i desenvolupament**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	29/10/2017	02/11/2017	4	PAC 2
Descripció	En aquest tasca s'analitzaran aspectes concrets de com haurien de ser els diferents entorns productius, pre-productius i de desenvolupament que es necessitaran. Es farà una proposta completa d'arquitectura i organització. S'introduirà com han d'integrar-se després aquests entorns en les eines de gestió dels repositoris i de compilació/deployment.			

Nom tasca	**Implementació solució triada / Anàlisis Organització codi i repositoris**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	02/11/2017	06/11/2017	4	PAC 2
Descripció	En aquest tasca s'analitzarà com s'ha de controlar el codi font, des de la pròpia estructura interna com a projecte, les relacions i referències a altres projectes i l'organització i publicació en repositori de codi font. Es proposarà un model de gestió i eines necessàries per a gestionar el codi font i les llibreries externes que calguin.			

Nom tasca	**Implementació solució triada / Integració en la memòria.**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	06/11/2017	08/11/2017	2	PAC 2
Descripció	Revisar i integrar tots els documents obtinguts en els punts anteriors i així lliurar en la PAC2 un document correctament estructurat i integrat en el format correcte de memòria.			

Nom tasca	**Implementació solució triada / Correccions i modificacions**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	08/11/2017	10/11/2017	2	PAC 2
Descripció	Fer si escau correccions i modificacions a partir dels possible comentaris del consultor.			

1.3.5.3 Activitats PAC 3

En la PAC 3, es tractarà el tema més important sobre l'eina central que gestionarà la compilació, test i publicació dels projectes. Es tractarà també la redacció del manual de gestió del producte.

Nom tasca	Implementació solució triada 2 / Anàlisis eina de compilació i deployment			
Informació tasca	Data inici	Data fi	Dies	Lliurable
	10/11/2017	18/11/2017	8	PAC 3
Descripció	En aquesta tasca s'analitzarà el gruix de l'eina que permetrà verificar el codi font fen les compilacions i execucions de tests corresponent, que s'encarregarà de definir tots els plans de compilació i publicació en els diferents entorns de producció/pre-producció, etc i d'administració del versionatge i control de totes les versions de les aplicacions creades. Es farà una possible proposta de configuració i procediment de treball.			

Nom tasca	Implementació solució triada 2 / Anàlisis formació equip tècnic			
Informació tasca	Data inici	Data fi	Dies	Lliurable
	18/11/2017	21/11/2017	3	PAC 3
Descripció	La introducció de tots els nous aplicatius i tecnologies implicarà canvis en la forma de treballar de gran part del personal tècnic. Cal identificar els rols específics involucrats en el procés i quins coneixements cal incorporar. Altre objectiu de la tasca és valorar els impactes de formació que altres àrees no tècniques puguin necessitar.			

Nom tasca	Implementació solució triada 2 / Anàlisis monitoreig			
Informació tasca	Data inici	Data fi	Dies	Lliurable
	21/11/2017	23/11/2017	2	PAC 3
Descripció	Fer l'anàlisi i valoració sobre quins possibles requisits des del punt de vista del monitoreig pot tenir el nou sistema d'Integració Continua implantat. Si calgués es podrà fer alguna proposta de punts a monitoritzar i eina a fer servir.			

Nom tasca	Redacció Manual Gestió del Producte			
Informació tasca	Data inici	Data fi	Dies	Lliurable
	23/11/2017	03/12/2017	10	PAC 3
Descripció	Redactar un petit manual de que es podria fer amb les eines utilitzades i una breu descripció d'exemples de configuració i us pràctic de les funcionalitats. La idea no és redactar un possible manual complet de com funciona al detall totes les funcions, sinó fer una introducció lo suficientment completa per a que es pugui tindre una idea clara del que es pot fer.			

Nom tasca	Integració en la memòria			
Informació tasca	Data inici	Data fi	Dies	Lliurable
	03/12/2017	08/12/2017	5	PAC 3
Descripció	Donat el gran volum de documentació aportada en aquest moment al PFC i el proper lliurament de la PAC 3 amb gran part dels continguts del PFC ja elaborats, assignar un dies específics per a integrar correctament tota la informació i revisar l'estructura i formats de la memòria.			

Nom tasca	Correccions			
Informació tasca	Data inici	Data fi	Dies	Lliurable
	08/12/2017	13/12/2017	5	PAC 3
Descripció	Realitzar les possible correccions necessàries obtingudes a partir dels comentaris del consultor.			

1.3.5.4 Activitats Lliurament Final

En aquest bloc bàsicament es treballarà la redacció final de la memòria completa i l'elaboració de la presentació virtual.

Nom tasca	Redacció conclusions			
Informació tasca	Data inici	Data fi	Dies	Lliurable
	15/12/2017	16/12/2017	1	Lliurament final
Descripció	Redactar les conclusions finals del PFC un cop ja assolits tots els objectius del qual.			

Nom tasca	Elaboració Presentació virtual			
Informació tasca	Data inici	Data fi	Dies	Lliurable
	16/12/2017	28/12/2017	12	Lliurament final

Descripció	Part important del PFC serà la correcta elaboració de la Presentació Virtual. Seguint els materials i indicacions elaborar la presentació per tal d'exposar al tribunal del PFC i dels seus resultats

Nom tasca	**Consolidació i redacció final memòria**			
Informació tasca	**Data inici**	**Data fi**	**Dies**	**Lliurable**
	28/12/2017	02/01/2018	5	Lliurament final
Descripció	Realitzar les possible correccions necessàries d'última hora en la memòria com acabar de quadrar els últims punts que puguin sorgir. Generar els fitxers en el format correcte i fer el lliurament final.			

1.4 Enfoc i metodologia seguida.

Per a l'elaboració d'aquest PFC es partirà amb la descripció d'un possible escenari habitual que es dona en moltes organitzacions. En aquesta situació l'absència de qualsevol mecanisme de Integració Continua, procediments de test dels projectes i mecanismes de publicació en els diferents entorns suposa un problema seriós d'operacions i de qualitat en l'organització.

L'opció de desenvolupar un projecte propi per a cobrir aquestes necessitats és de molt difícil aplicació real. És un procés complex que requereix una gran inversió en el disseny i programació de les eines i scripts que puguin d'alguna forma encaixar en aquestes necessitats. La millor opció passarà per fer servir eines ja existents en el mercat que encaixin en el context actual de l'organització i en els requisits que es volen satisfer.

Es farà l'anàlisi d'algunes de les eines disponibles en el mercat que gestionen la Integració Continua i en funció de les seves característiques i el context de l'organització es triarà una possible solució.

La presa de decisions es basarà no sols en el cost econòmic de l'eina sinó que també es consideraran aspectes com: l'ús d'eines àmpliament reconegudes pel sector, que siguin totalment funcionals i que disposin d'un bon servei de suport. Aquest últim punt ens ajudarà a garantir la correcta instal·lació i posada en marxa del sistema i solucionar qualsevol incidència que pugui succeir.

1.5 Requisits del projecte.

Per tal d'elaborar aquest PFC i la redacció de la seva memòria i presentació es necessitaran els següents requisits informàtics i documentals.

1.5.1 Requisits de maquinari

Les característiques del maquinari que es farà servir són les següents:

- Processador Intel® Core(TM) i7-4810MQ CPU @ 2.80GHz.
- Memòria RAM 8 GB DDR3L SDRAM.
- Disc dur intern SSD de 500GB.
- Disc dur extern USB 3.0 WD de 2TB per a *backups*.
- Connexió Internet ADSL 10 Mbps.

1.5.2 Requisits de programari

Les característiques del programari que es farà servir són les següents:

- Sistema Operatiu Windows 10 Pro.
- Microsoft Office Professional 2013 suite amb Microsoft Word,Microsoft Excel i Microsoft Power Point.
- Microsoft Project 2016.
- Microsoft Visio.
- Cute PDF Writer.
- Adobe Acrobat Reader DC.
- Els necessaris en funció de la proposta triada.

1.5.3 Requisits recursos de documentació

També es faran servir les següents fonts d'informació

- Recursos propis de la UOC com:
  - Biblioteca http://biblioteca.uoc.edu/.
  - Materials propis aula sobre presentació, elaboració i redacció de textos científics.
- Recursos documentació Web principals fabricants:
  - Microsoft Docs https://docs.microsoft.com/.
  - Amazon AWS, recursos DEVOPS https://aws.amazon.com/es/devops/.
  - Atlassian Documentation https://confluence.atlassian.com/alldoc/.
  - Terminologia informàtica en català https://www.softcatala.org/recull-termes-softcatala/.

1.6 Productes obtinguts

Els productes obtinguts en aquest projecte a més dels propis requerits per l'elaboració i presentació del PFC, seran els següents:

- Document memòria del PFC en format PDF.
- Presentació virtual del projecte.
- Anàlisis teòric d'un servei d'IC amb les seves característiques i components principals.
- Anàlisis de diferents eines de Integració Continua.
- Manual de gestió del producte implantat.

1.7 Breu descripció dels capítols de la memòria

La memòria està estructurada en els següents capítols:

Capítol 1. Introducció a la temàtica del PFC i descripció de la planificació a seguir, amb els corresponent detall de tasques i requisits.

Capítol 2. Introducció a la Integració Continua. Breu recerca històrica i explicació dels principals components i característiques d'un sistema d'IC.

Capítol 3. Anàlisis d'eines i productes IC del mercat. Breu descripció de les seves característiques principals i comparació. Es justificarà també l'eina triada.

Capítol 4. Treball Pràctic. Plantejament , anàlisis i desenvolupament d'un sistema de Integració Continua en una empresa tipus.

2 Introducció a la Integració Continua

En aquest capítol es donarà una descripció general sobre què és un sistema d'Integració Continua (*Continuous Integration*). Es tractaran, entre d'altres, conceptes com la seva definició, la seva historia, quins són els principals components que intervenen en el sistema i altres pràctiques relacionades amb la IC.

2.1 Què és la Integració Continua ?

La IC és una pràctica de gestió del desenvolupament de projectes informàtics en el que els equips de programadors col·laboren conjuntament, integrant el més freqüent possible els seus canvis i aportacions en el codi font, fent servir un repositori central. Es pretén principalment integrar el codi i validar-lo automàticament per a detectar el més aviat possible qualsevol error.

Aquesta pràctica pren especial rellevància quan es fa servir una metodologia de desenvolupament **Àgil** en la que es desenvolupen i integren les noves funcionalitats introduïdes de forma rutinària i habitual. En molts casos fins i tot diversos cops al dia.

En aquest procés d'integració l'objectiu màxim és assegurar-se que el codi integrat és correcte tant des del punt de vista de compilació com funcional. S'executaran recurrentment els *tests* que verifiquen que els canvis introduïts no han provocat errors de funcionament de l'aplicació.

Inicialment abans de la introducció de metodologies àgils o de mecanismes automàtics d'integració, molts dels programadors treballaven desenvolupant en les seves pròpies màquines. Es feien servir només els seus compiladors i recursos locals de tal forma que s'aconseguia que "allò fet" funcionés en aquella màquina. Això no era cap garantia de que a l'executar la mateixa aplicació en altres entorns (preproducció/producció) o a l'ajuntar mòduls i codis creats per altres equips, l'aplicació continués funcionant lliure d'errors i amb el comportament desitjat.

Aquest procés d'integració, encara que es fes servir un repositori compartit de codi, es caracteritzava en què la freqüència amb la que es lliurava i compartia el codi era molt baixa. Molts cops es tractava de dies o setmanes i no es feia fins tenir tot el producte o la nova funcionalitat acabats. En aquest context, la capacitat de detectar errors estava força reduïda i el més complicat era saber on és trobava l'error. El problema es donava principalment perquè passava massa temps i s'havien fet masses canvis en el codi font entre els diferents lliuraments i integracions. Això feia molt difícil identificar la causa. És més fàcil solucionar un error produït tot just fa uns minuts o hores, que no pas un error causat per canvis introduïts fa dies o setmanes.

En definitiva la manca de mecanismes automàtics en totes aquestes fases dificulta la capacitat de compartir i integrar ràpidament petites peces de codi funcionals i de detectar els errors executant els tests necessaris sobre l'aplicació.

Amb la intenció de resoldre aquests problemes i de millorar la capacitat, moltes empreses del sector s'han adaptat i han implementat un sistema de Integració Continua.

2.2 Breu antecedents històrics de la IC

Des dels principis del desenvolupament del software recent, sempre hi ha hagut procediments i/o pràctiques per a compilar i integrar d'algun mode el software.

Inicialment, i encara avui en dia en ús a alguns sectors, la pràctica de córrer processos automàtics de compilació i integració de forma diària o cada cert temps normalment per les nits, ha sigut suficient quan la dimensió dels projectes en termes de recursos, fitxers i nombre de desenvolupadors involucrats era petita.

A mida que la complexitat i la demanda del mercat de lliurar amb més freqüència els projectes ha anat creixent amb el temps, s'ha començat a introduir la necessitat i el concepte de la IC en el sector de desenvolupament de software. Potser inicialment d'una forma diferent a com s'entén avui en dia la IC, però amb certes similituds, que poc a poc va evolucionar fins a l'actual pràctica.

Per exemple, el primer a citar-ho va ser **Grady Booch** [1] (destacat arquitecte de software Americà) al principi dels anys 90. En el seu llibre *Object Solutions: Managing the Object-Oriented Project* [2] suggereix que: *en el procés del desenvolupament orientat a objectes , el procés de Integració Continua executat regularment produeix executables que creixen en funcionalitat en cada lliurament. A través d'aquest procés es poden gestionar i mesurar de forma progressiva el progrés i la qualitat de tal forma que es poden anticipar i identificar i resoldre els diferents problemes i riscos de forma continua.* No és una referencia a fer servir la Integració Continua de tot el projecte com a tal, sinó un dels avantatges de fer-ho servir com a mecanisme de gestió de riscos i de la qualitat.

En altres contextos, com els descrits per Steve McConnel, es van emprar pràctiques similars com "Daily Build and Smoke Test" com a tècniques emprades a Microsoft durant els anys 90 [3]. Llavors el concepte més rellevant era més la freqüència que no pas encara l'automatització.

Altres autors com Michael A. Cusumano i Richard W. Selby al llibre Microsoft Secrets, fan referències i discussions a aquest tipus de pràctiques pioneres en els processos de "*daily builds*" de Microsoft.

Amb la introducció de pràctiques de desenvolupament de software com *Extreme Programing* (XP) per part de Ron Jeffries i Kent Back en la seva participació en el projecte "Chrysler Comprehensive Compensation (C3)" [4] van pràcticament definir la metodologia XP actual. En aquest projecte es va poder fer un ús pràctic, complet i reconegut d'Integració Continua.

D'altra banda, l'enginyer de Software Martin Fowler [5], també relacionat amb aquest projecte C3 i en major grau amb metodologies Àgils, va escriure un article originalment a l'any 2000 que definia les bases teòriques del que s'entén avui en dia com a Integració Continua. L'article està basat en les seves experiències a la companyia **ThoughtWorks**, on treballava. Una segona versió d'aquest article al 2006 [6] defineix els punts específics que tot sistema de IC ha d'implementar per a ser considerat com a tal. També es citen les pràctiques que ha de seguir l'equip de desenvolupadors per dur a terme la IC.

Un altre punt important és la publicació, inicialment desenvolupat per treballadors de la mateixa companyia ThoughtWorks, de la primera versió d'un Servidor d'Integració Continua anomenat CruiseControl [7]. Aquest servidor implementa funcionalitats completes essencials per a monitoritzar els repositoris de codi font, llençar els processos de compilació (*build*), execució de *tests* i notificació i enregistrament dels resultats. Està escrit en Java i publicat com a *open-source* software amb llicència BSD.

2.3 Principals components d'un sistema de IC

Des del punt de vista funcional i estructural la majoria dels sistemes de IC que podem trobar al mercat, estan formats per una conjunt d'actors i components comuns. Un sistema d'IC es basa en la interacció entre diferents sistemes informàtics i d'un conjunt d'actors involucrats directament en el procés de desenvolupament de software que segueixen una metodologia.

Els principals components els podríem definir de la següent forma:

2.3.1 Equip de desenvolupament.

En aquest grup estarien compresos tots els membres que intervenen en el procés de desenvolupament:

- **Programadors**: treballen amb els seus equips i software de desenvolupament i són els principals encarregats de publicar els seus canvis al repositori de codi font.
- **Caps de projecte, Analistes i altres càrrecs de gestió dins l'equip**: tot i que molts d'ell no interactuen directament en la programació i la pujada de codi als repositoris, són persones que reben informació del servidor d'integració sobre els resultats i necessiten saber en tot moment l'estat de les integracions.
- **Equips informàtics i aplicacions de desenvolupament**: també es consideraran en aquest grup tots els que interactuen amb el sistema de IC.

2.3.2 Repositori de Codi Font

Aquest component, a grans trets, és l'encarregat de:

- Emmagatzemar estructuradament **tots** els recursos necessaris en forma de fitxers, que formen part del projecte.
- Gestionar les diferents branques i versions que composen el projecte.
- Gestionar totes les operacions per a modificar els projectes (*push, pull, merge, clone,* etc).
- Enregistrar tota l'activitat feta per part dels programadors.

Els repositoris poden estar en les pròpies màquines locals de desenvolupament, però en la majoria de casos es troben funcionant de forma centralitzada en servidors remots dedicats.

2.3.3 Servidor Integració Continua

En general el mòdul de servidor de IC pot estar a la vegada composat per diferents submòduls interns. Cadascun d'aquests es poden executar en 1 o diversos servidors a la vegada de l'organització o fins i tot al núvol. En conjunt el servidor i tots els seus submòduls proporcionen les següents funcionalitats:

- **Agent de monitorització del repositori de Codi Font**. Aquest agent és l'encarregat de monitoritzar tota l'activitat que es dona al repositori, com per exemple nous canvis introduïts, noves branques publicades, nous *Tags*, etc. La principal funció és llençar *l'event* intern que dispararà les diferents funcions del servidor d'IC. Per exemple *l'event* d'un nou *push* a la branca *master* dispararà la compilació automàtica del projecte.
- **Agent de compilació i empaquetatge.** El servidor de IC a partir dels *events* rebuts del repositori de codi font, agafarà la versió específica (normalment els

últims canvis) i farà una compilació + *build* (empaquetatge de tots els components de l'aplicació) per a verificar la correctesa del projecte. Aquest procés també es podrà llençar de manualment sota demanda quan sigui necessari. Com a resultat s'obtindran els corresponents *artifacts* de l'aplicació. Si es produeix qualsevol error, la integració del codi s'aturarà i es generarà un informe d'errors. Aquest es notificarà als actors subscrits a rebre aquesta informació.

- **Agent execució dels *tests*.** El servidor de IC executarà els *tests* incorporats en el projecte a fi de validar la correcta execució de l'aplicació. En funció del resultat dels *tests* es considerà correcta o no la integració del codi.
- **Agent notificació dels resultats**. El servidor de IC ha de tenir la capacitat d'enregistrar els resultats de totes les proves d'integració fetes (compilacions, *tests* i altres verificacions) i d'enviar les corresponents notificacions als usuaris que tenen assignats determinats rols i/o perfils que hagin de ser notificats (cap de projecte p.e.). Aquest tipus de notificació no sols ha d'arribar al programador que ha fet el canvi, sinó que també a la resta de l'equip, als managers, etc. Es poden enviar per exemple per correu electrònic, notificacions push mitjançant alguna aplicació, panells d'administració, de *reporting* etc.

2.3.4 Agent de monitorització

El sistema ha de saber en tot moment en quin estat es troben els diferents components que formen la plataforma de IC i quin és l'estat dels diferents entorns allà on treballa. A la vegada el sistema ha der fer una mínima monitorització de l'entorn de producció un cop es publiquen noves versions de codi.

2.3.5 Entorns

Tant els desenvolupadors com el servidor de IC treballen en els diferents entorns disponibles per a l'organització. Aquests poden ser:

- **Desenvolupament**: allà on treballen en local cadascun dels desenvolupadors.
- **Integració**: entorn unificat on el servidor de IC i els desenvolupadors integren els seus desenvolupaments.
- **Pre-producció**: entorn de validació, el més semblant possible al de producció.
- **Producció**: entorn productiu.
- **Altres**: depenent de l'organització i les seves necessitats es poden considerar altres entorns especialitzats com per exemple: per a compartir amb clients o altres col·laboradors externs, entorns isolats per a proves de càrrega, *demos*, etc.

2.4 Principals fluxos i interaccions en un sistema d'IC

Tots els components citats en el punt anterior interactuen entre ells per implementar el flux d'Integració Continua. El següent gràfic il·lustra d'una forma més visual aquestes interaccions:

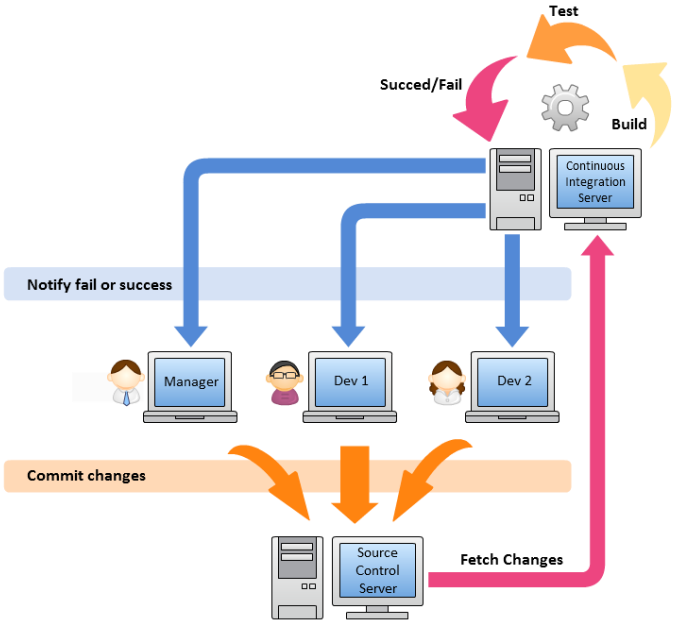


Figura 3: Components i interaccions bàsiques a un sistema de IC (font)

En aquest gràfic es poden identificar les 3 següents accions:

1. Principalment els programadors i altres càrrecs dels equips de treball un cop tenen una nova versió de codi que volen integrar publiquen aquest nou canvi al repositori de codi font central de l'organització. Aquest pot estar en algun servidor local de la xarxa interna, al *Cloud* corporatiu o a plataformes externes com GITHUB p.ex. Prèviament han:
    a. Compilat correctament en el seu equip local fent servir una eina específica o IDE de desenvolupament.
    b. Assignat versionatge.
    c. Fet *commit* i *push* a la branca que volen integrar (generalment la *MASTER*).
2. El servidor de codi font enregistra el nou canvi i aplica tots el processos necessaris per a combinar (*merge*) aquest codi nou amb la resta de canvis previs d'altres programadors.
3. El servidor de IC monitoritza tota l'activitat que es dona al repositori de codi font i executa un procés de:
    a. Compilació + *build* a fi de tornar a validar l'aplicació en un entorn el més semblant als real d'integracio/pre-produccio i producció.
    b. Execució de tots els diferents *tests* que pugui portar el codi. En aquest punt es pot fer servir una amplia gama d'eines de *tests* que dependran del tipus d'aplicació i de les eines emprades.
    c. Enregistrar i notificar a tots els implicats els resultats.

2.5 Característiques i pràctiques d'un sistema d'IC

Coneguts ja els principals mòduls i interaccions d'alt nivell que es donen en un sistema d'IC cal definir les principals característiques que ha de seguir un sistema d'IC tant a nivell tècnic com a nivell de procediments. La Integració Continua no és solament el conjunt d'eines, aplicacions i tecnologies que ajuden a aplicar-ho sinó que és també un conjunt de "regles i pràctiques" que s'acaben plasmant en procediments explicatius de com ha de treballar i operar l'organització durant tot el cicle. És tan important aquesta part operativa com les tecnologies emprades en la IC, ja que sense la correcta actitud

de tots els actors que participen en el procés de desenvolupament, l'aplicació de la IC no es donarà de forma correcta ni eficient.

Hi ha molta literatura, opinions i corrents sobre les principals característiques que defineixen la IC. En els següents punts es resumiran de forma general quines són aquestes característiques i les pràctiques principals de la IC.

2.5.1 Manteniment d'un repositori de codi font

S'ha de portar a terme una correcta gestió del codi font i de tots els recursos que componen un projecte. Això s'aconsegueix fent servir una aplicació específica encarregada només de gestionar el repositori de codi font (com poden ser GIT, Subversion, etc).

Tots els programador han de:

- Saber perfectament on es troba el repositori a fer servir per a cada producte.
- Aquest ha de ser l'**únic** en ús.
- Tenir clar on és troben dins l'estructura del repositori tots els components i versions que composen l'aplicació.
- Ha d'afegir al repositori **tots els fitxers i recursos necessaris** per a poder compilar i executar l'aplicació (codi font, fitxers configuració, esquemes de BBDD, *scripts*, llibreries de tercers, etc..). No es poden fer servir cap tipus de referència locals que impossibilitin a altres persones compilar i executar el projecte.
- No guardar **mai** al repositori els fitxers binaris resultat de la compilació.
- Fer un ús racional i no abusiu de les branques.

Tot programador ha de poder fer un *check-out* des de zero del projecte per obtenir tots els components que el formen. Aquest codi ha de compilar per si mateix sense la necessitat d'afegir referències o recursos de tercers externs. El codi ha de ser funcional, ha d'estar sempre preparat, per a qualsevol persona que el vulgui compilar i fer servir.

2.5.2 Automatització del procés de muntatge

Requisit indispensable per a poder córrer una Integració Continua és l'automatització del procés de compilació i muntatge de l'aplicació. Aquesta s'ha de poder muntar amb l'execució d'un únic script o procés automàtic, que faci tota la feina per a obtenir el resultat final de l'aplicació un cop muntada. Això també inclou les instruccions necessàries per a desplegar l'aplicació en un entorn similar al de producció o integració necessari.

Aquest script ha de contenir totes les instruccions necessàries per a:

- Compilar.
- Moure tots els recursos necessaris (fitxers, llibreries, etc) a les seves destinacions finals.
- Aplicar les configuracions i esquemes necessaris a Bases de Dades i altres recursos que necessitin ser inicialitzats.

A més a més, el procés de muntatge ha de ser:

- **Ràpid**. El temps necessari ha de ser el mínim possible amb l'objectiu de garantir el màxim nombre d'integracions que es puguin donar i obtenir el *feedback* dels

resultats el més ràpid possible. D'aquesta forma s'intenta maximitzar l'eficiència de l'equip de desenvolupament. Temps de resposta de molts minuts o hores no són acceptables per a la pràctica eficient de la IC.
En aquest aspecte molts sistemes treballen en l'execució en paral·lel dels diferents agents que es poden executar a la vegada en diferents servidors.

- **Eficient**. Ha de ser prou intel·ligent per a només recompilar o reconstruir la part afectada del projecte. Per exemple no sempre cal compilar i muntar tota l'aplicació sencera per un canvi específic en alguna línia de codi.

La majoria d'aquests aspectes ja són gestionats automàticament per molts IDEs de desenvolupament que fan servir els programadors.

2.5.3 Fer el codi auto verificable

Es necessari que el codi i l'aplicació final un cop han sigut compilats i empaquetats tinguin el comportament esperat. Hi ha molts errors i problemes que no es poden detectar en les fases de compilació i que només es produiran un cop s'executi el codi o es donin certes interacciones entre els diferents mòduls (serveis externs, BBDD remotes, configuracions de xarxes, etc) que el composen. Això s'aconsegueix en gran mida amb l'execució automàtica d'un conjunt de proves "*tests*" sobre el codi i el flux de l'aplicació.

En la pràctica de la IC és molt important invertir temps i esforços en la programació i especificació de *tests* que verifiquin:

- Cada nova funcionalitat introduïda en el codi font. Aquí poden ser útils els tests unitaris com a pràctica per testejar seccions i funcions de codis. No cobreixen per exemple *tests* de flux.
- Que la resta de funcionalitats no s'han vist afectades per aquest canvi.
- Que la funcionalitat global de l'aplicació és la correcta. Aquí son útils tècniques de validació funcional per a validar a més alt nivell operacions de flux de l'aplicació. Eines com HTTPUnit o Selenium es poden fer servir per a validar interfícies d'usuari per exemple.

L'esforç dedicat a programar aquestes validacions com a part habitual i necessària del procés de desenvolupament implicarà una reducció en el futur, concretament en les fases posteriors de la integració, del nombre d'errors i del temps necessari per a trobar-los i corregir-los. No és una solució per a garantir al 100% el correcte comportament de l'ampliació però ajudarà a reduir dràsticament els errors i en conseqüència la qualitat del software lliurat. Sense aquesta activitat, seria inviable la IC. A [8] es pot trobar informació més detallada.

2.5.4 Pràctiques de publicacions a la branca *master*

Un dels principals objectius de la IC és minimitzar el temps i esforç dedicat a cadascuna de les integracions que es fan de manera que es pugui disposar d'una versió preparada per anar a producció en qualsevol moment.

Així doncs, tots els desenvolupadors que estan treballant en un projecte, i en especial en els equips que treballen en metodologies àgils, han de publicar els seus canvis seguint una sèrie de pràctiques recomanades (tot i que es podrien interpretar com a obligacions). A continuació les més significatives:

2.5.4.1 Freqüència de publicació

Una bona pràctica és publicar a la branca *master* del repositori el més aviat possible, com a mínim 1 cop al dia.

Com més freqüentment es publica al repositori, més ràpidament es comuniquen aquests canvis a la resta de desenvolupadors, s'integra el nou codi i es detecten els possibles conflictes introduïts per altres programadors amb l'execució del les proves de validació.

2.5.4.2 Procediment de publicació

El programador ha d'adquirir la pràctica de seguir aquest cicle cada cop que es vol publicar un nou canvi:

- Actualitzar el repositori local amb la versió actual de la branca *master.*
- Resoldre qualsevol conflicte entre el seu nou codi i l'actual obtingut en el punt anterior.
- Compilar i empaquetar correctament en la seva versió local (això pot incloure també execució de tests).
- Si tot és correcte, llavor publicar els canvis en la branca *master* del repositori de codi font remot.

Arribats a aquest punt, es torna a engegar tot els procés automàtic d'integració per a tornar a validar.

D'aquesta forma ens hem assegurat que el nou codi és compatible amb l'última versió de la *master* que conté la resta de canvis dels altres programadors.

2.5.4.3 Integració immediata en un entorn unificat

Un altre aspecte important a considerar és que totes les publicacions fetes a la branca *master* s'han d'empaquetar i validar **immediatament** en un entorn d'integració unificat per a tots els programadors. Que significa això?

- S'ha de verificar que les diferències entre els entorns de desenvolupament dels programadors (sistemes operatius, diferents versions de IDE o de llibreries, compiladors, recursos de xarxa, etc) no puguin estar causant falsos resultats. És a dir, funciona en la màquina local del programador però no en la màquina d'integració i viceversa.
- Que no s'han agafat canvis anteriors previs al moment de fer la publicació i que aquests canvis introduïts causen conflicte amb els últims de la *master*.

Per això és important que tot aquest procés no depengui de processos manuals sinó d'un procés automàtic i totalment integrat en el flux de treball, gracies a un sistema de IC. El servidor de IC com que monitoritza contínuament el repositori, agafarà una còpia de la versió de la branca *master* i la integrarà dins el servidor d'integració a fi de verificar automàticament que tot es correcte.

En la següent figura es pot veure visualment com es relacionen els principals components d'un sistema IC.

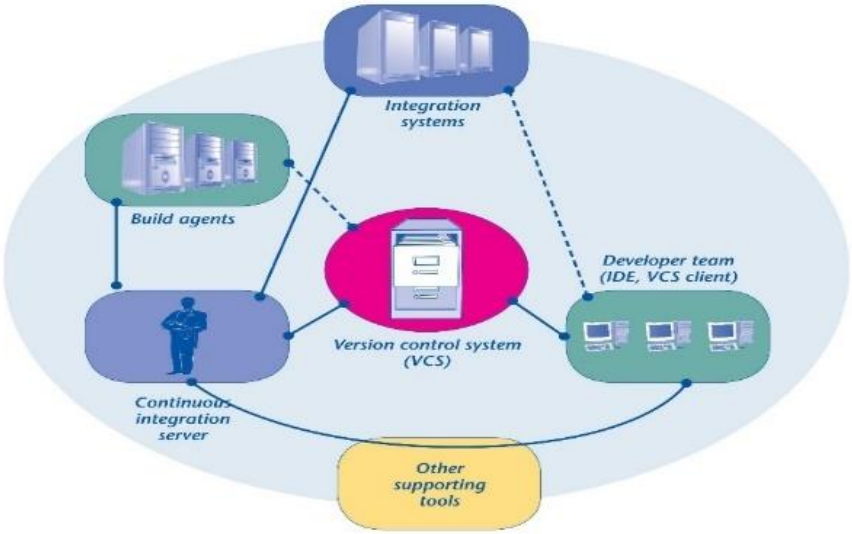


Figura 4: Relacions i dependències dins el flux de IC (font)

Es pot veure que tant els entorns de desenvolupament de forma indirecta com el servidor de IC directament fan servir servidors d'integració dedicats i unificats. El servidor de codi font (VCS en la figura) és qui centralitza les diferents versions i tots els servidors i equips de desenvolupament han d'obtenir i publicar el codi través d'ell.

2.5.4.4 Correcció immediata dels errors

Tot equip de desenvolupament per procediment, ha d'evitar en tot moment que la branca *master* del projecte no es pugui validar (es trenqui). Si en el moment d'introduir un nou canvi i llençar la integració cap a la *master* es dona un error, el primer objectiu abans que qualsevol altre és corregir aquesta situació. Si la cadena de la Integració Continua es trenca no es permetrà als altres desenvolupadors continuar amb una versió estable i funcional de l'aplicació.

En aquesta situació s'ha de tornar a l'última versió estable de la *master* i a continuació en altres branques diferents investigar i solucionar el problema.

Aquesta pràctica és una actitud **essencial** que ha de fer seva l'equip de desenvolupament. El cost de corregir un error un cop ha anat a producció es major que si s'hagués detectat i solucionat immediatament.

2.5.5 Validar el software en un clon de l'entorn de producció

Amb l'objectiu de garantir que el software integrat sigui totalment funcional un cop es posi en producció, cal revisar la infraestructura de l'organització. S'ha de garantir que es disposa d'un entorn de proves que estigui en les mateixes condicions que l'entorn de producció. Es dona per descomptat que la naturalesa de l'organització i del producte en si determinaran el grau de viabilitat. No obstant cal intentar almenys garantir aspectes com:

- Fer servir la mateixa versió / gestor de BD.
- Fer servir els mateix SO, servidor Web i/o d'aplicacions, etc.
- Les mateixes llibreries i recursos/serveis de tercers que es tindran a producció.

- En la mida del possible, la mateixa configuració de xarxa que a producció o el més similar possible.
- El mateix tipus de hardware.

Tècniques de virtualització de servidors amb copies dels entorns de producció poden ser útils, així com diferents serveis al *Cloud* (Microsoft Azure o Amazon AWS) que ofereixen un gran ventall de possibilitats per a entorns basat en *Cloud*.

2.5.6 Facilitar l'accés a l'última informació

Tot sistema de IC ha de proporcionar als seus usuaris informació sobre què està succeint en temps real en el processos d'integració. Tot entorn d'IC ha de proporcionar eines com panells, serveis de notificacions, *mailings*, etc que mostrin i informin de forma automàtica i immediata informació sobre:

- Resultats de totes les integracions realitzades.
  - S'ha de poder accedir de forma fàcil al últim resultat.
  - Estat actual de tots els projectes involucrats.
  - Monitor d'activitat. Mostrar qui està fent alguna operació i què està fent.
  - Disposar del registre històric complet de tota l'activitat feta anteriorment.
  - Estadístiques.
- Alertar en cas d'error en l'execució d'una compilació, empaquetatge, prova de validació, etc. Com fer-ho? Dependrà de cada producte de IC, però ha de ser una comunicació senzilla i efectiva.

La següent figura il·lustra un possible escenari on es mostren exemples de fluxos de comunicació en el que un sistema amb IC notifica missatges d'errors en diferents parts del flux d'integració.

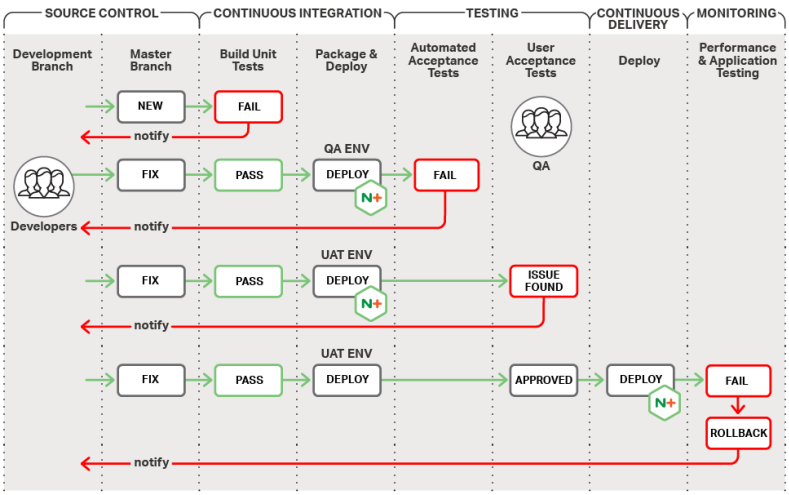


Figura 5: Exemple de flux de treball i notificacions

En tots els casos els desenvolupadors sempre són avisats davant de qualsevol contingència. Llavors s'han de portar a terme les accions necessàries per a solucionar el problema.

Part de la informació i funcionalitat que també es requereix d'un servidor de IC és la d'obtenir de forma senzilla l'última versió correcta de l'aplicació que es vol. En qualsevol moment, tothom ha de poder "descarregar" l'última versió executable per a testejar-la,

ensenyar-la a altres membres de l'equip, a clients, etc. L'important és garantir que és la versió correcta i que funciona tal i com s'espera. Això és força habitual en entorns de desenvolupament Àgils on la interacció producte-client és constant. Un bon gestor de IC serà clau en aquest aspecte.

2.5.7 Automatització de la publicació

En molts sistemes de IC, l'últim pas que es dona en la cadena d'integració és la publicació de l'aplicació. Aquest pas en molts casos és una funcionalitat afegida en el servidor d'IC.

Aquesta publicació pot ser tan en els entorns de tests com els de producció. Com més ràpida i automàtica sigui, més fàcil i segura serà la publicació. Aquesta automatització també fa necessari el mecanisme invers per a desfer la publicació (*rollback*), tornant a l'estat anterior de forma automàtica i segura.

Amb els mecanismes de publicació i de *rollback* automatitzats l'equip de desenvolupadors serà capaç de publicar els nous canvis més ràpidament i més sovint. Qualsevol contingència o problema, si cal, pot ser solucionada amb un clic al botó de *rollback*.

Normalment aquesta automatització es realitza amb la configuració de scripts que poden ser executats pel servidor de IC un cop totes les validacions han sigut correctes. Altres eines mostren panells on poder programar de forma més visual tots aquests processos. Es defineixen en *plans* per a cada aplicació i entorn i es poden configurar els diferents esdeveniments que fan executar aquest processos.

El mode en com es gestiona aquesta part específica del procés de publicació, defineix altres metodologies i pràctiques que enriqueixen les funcionalitats base d'un sistema de IC.

2.6 Altres pràctiques relacionades amb la IC

En el model d'Integració Continua ens hem centrat bàsicament en la fase que va des del desenvolupament del software per part dels programadors, fins que es valida i es crea l'empaquetatge corresponent. Normalment aquesta versió ha superat un conjunt de *tests* automàtics que la valida i es té un software preparat per a anar a producció. Com es pot portar aquesta aplicació a producció i de quina forma?

En el següent gràfic es mostren passos addicionals posteriors a la fase d'integració que es donen si s'adopten altres pràctiques que estenen un pas més enllà el concepte de la Integració Continua.

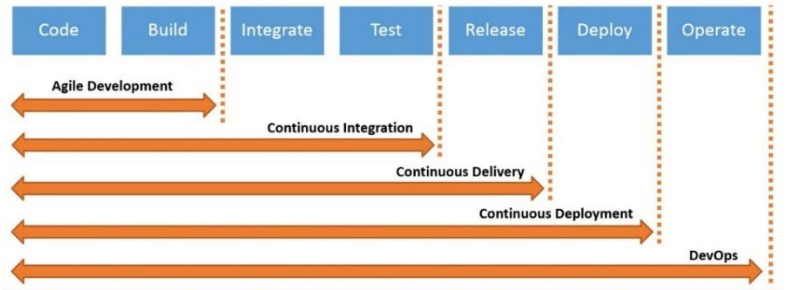


Figura 6: Altres pràctiques relacionades (font)

Amb la Integració Continua com a tal s'arriba fins al pas de tenir un codi totalment integrat i que ha superat les proves de validació. Es té doncs una versió apta per a ser empaquetada i emprada.

Amb l'automatització del passos de *Release* i *Deploy* es pot controlar com es lliuren les aplicacions als clients finals.

2.6.1 Lliurament Continu

Una organització de desenvolupament de software, que fa servir IC fa ús de Lliurament Continu quan el sistema de IC és capaç de produir de forma automàtica una versió empaquetada del codi que està totalment validada i preparada per a ser publicada a producció prèvia autorització **manual**.

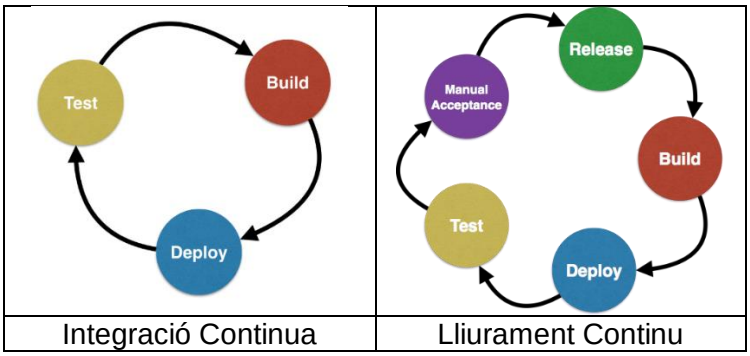


Figura 7: Integracio Continua Vs Lliurament Continu

Mentre que en la IC només s'arriba a validar el software amb el cicle "Build => Deploy (en els entorns de validació) => *Test*, amb el lliurament continu es configura el sistema per a que amb un sol clic en el panell d'administració de l'eina el nou software sigui automàticament publicat en producció (Release).

Amb aquesta pràctica s'afegeix un punt més de control abans de lliurar al client final. Cada publicació a *master* per part dels desenvolupadors està potencialment de forma automàtica a només 1 clic de ser lliurada al client. Normalment aquesta pràctica està molt relacionada a com l'organització, concretament l'àrea de negoci, vol administrar el procés de lliurament, sent ells els últims a prendre la decisió. Sense dubte aquesta pràctica pot comportar una baixada de la quantitat i velocitat de lliuraments que pot fer l'organització.

2.6.2 Desplegament Continu

La pràctica del desplegament continu consisteix en automatitzar encara més el procés per a que cada publicació dels programadors al repositori, que superi totes les fases de validacions de la IC, es publiqui **automàticament** a producció. És automatitzar la publicació portant un pas més enllà el Lliurament Continu.

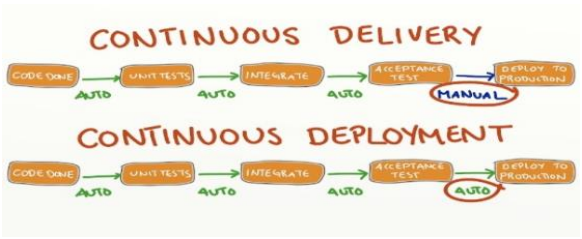


Figura 8: Continuos Delivery Vs Continuous Deployment

Per a què una organització pugui adoptar aquesta pràctica ha de plantejar-se un canvi radical en la seva forma de treballar en la que habitualment el procés de lliurar el software al client era més complex requerint un llarg procés de proves i validacions, personal dedicat a aquest procés, etc. En aquest context com que el nombre de lliuraments era molt petit, el feedback obtingut dels clients era poc freqüent i normalment corresponia a grans parts del producte o de tot ell. És un procés poc "àgil".

Ara, l'organització gracies a un sistema de Integració Continua i una metodologia de treball enfocada en ella, ha de canviar l'estratègia clàssica de fer grans lliuraments per la de lliurar als clients petites funcionalitats de forma gradual i constant. En conseqüència es tindrà una resposta més ràpida amb els resultats per part del clients (i poder adaptar/corregir més ràpidament) i anar lliurant progressivament l'aplicació completa.

L'organització ha de posseir una forta cultura en aquesta metodologia, en programar una gran quantitat de *tests* que verifiquin al màxim el comportament correcte de l'aplicació. Es vol minimitzar el risc d'introduir errors en l'entorn de producció, trencant el correcte funcionament de l'aplicació. Aquest canvi de mentalitat per part de l'equip no és gens fàcil.

3 Anàlisis de productes IC en el mercat

En aquest capítol es durà a terme una cerca de les principals eines del mercat, ja siguin de codi lliure/gratuïtes com a propietàries/de pagament que ofereixen els serveis del model de Integració Continua exposat. En tant que el nombre de productes és força elevat i no es pot fer una anàlisi complet, es posarà focus en les següents 5 eines.

3.1 Jenkins

3.1.1 Breu introducció històrica

Jenkins és un eina de Integració Continua basada en OpenSource, escrita en el llenguatge de programació Java.

Jenkins prové del projecte Original anomenat Hudson [9], desenvolupat originalment pel programador Kohsuke Kawaguchi a l'any 2004 quan treballava a Sun Microsystems. Uns anys més tard l'adquisició per part d'Oracle de Sun, va causar que la comunitat de Hudson fes un bifurcació cap a l'any 2011 del projecte original en un nou projecte: **Jenkins**. Oracle va continuar treballant Hudson com a tal esdevenint dos projectes totalment separats i diferenciats. Des d'aquell moment Jenkins es va distribuir com a *Open Source* a GITHUB sota llicència MIT [10].

3.1.2 Principals Característiques

Jenkins és un servidor que permet automatitzar totes les tasques i processos relacionats amb el desenvolupament, validació, empaquetatge i publicació del software sota un model d'Integració Continua. Fonamentalment és una màquina d'automatització que permet definir múltiples patrons i regles d'execució de tasques que componen el procés complert de la IC. La unitat bàsica de configuració és l'anomenat "Jenkins Pipeline".

3.1.2.1 Jenkins Pipelines

Jenking Pipelines no és més que un conjunt de codi i/o *plugins* que executen una tasca en concret. Es poden combinar aquestes tasques de forma que es construeix un flux específic. Aquest flux es pot combinar a la vegada amb altres fluxos per tal d'obtenir el procés final desitjat.

A continuació es mostra un possible escenari bàsic modelat a partir de la definició i execució en paral·lel de diferents Pipelines. Aquesta acció està iniciada amb el *commit* fet al Repositori i arriba al resum de les proves de validació i publicació de la versió integrada.

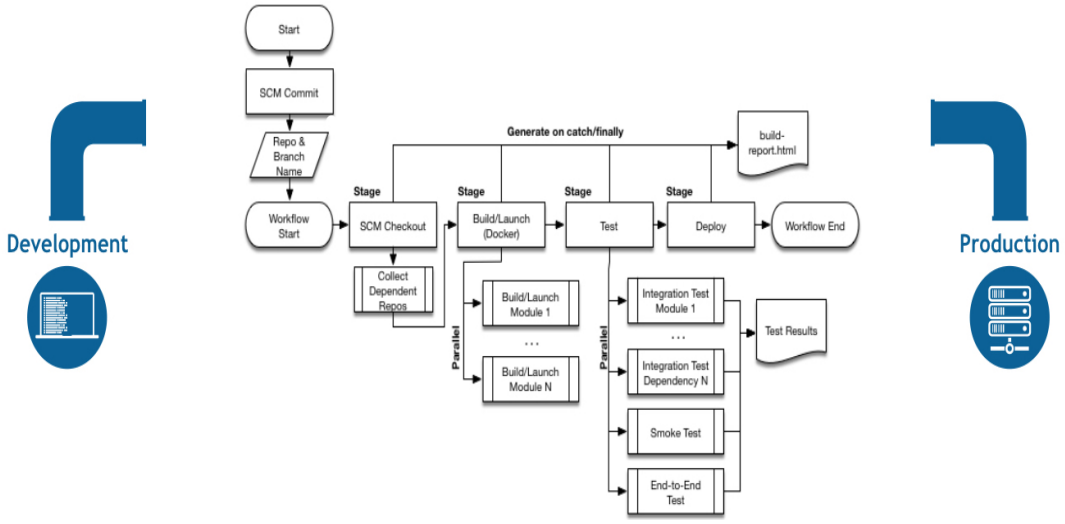


Figura 9: **Exemple de configuració de tasques en Pipelines**

Definint de forma correcta aquestes tasques en "Pipelines" es poden obtenir les funcionalitats desitjades d'una aplicació de IC per tal de tenir un flux totalment automatitzat i segur.

3.1.2.2 Definició i gestió de Pipelines

Jenkins ofereix una llenguatge de programació propi, anomenant "*Pipeline Domain Specific Language (DSL) syntax*" [11] que permet:

- Modelar regles senzilles en *pipelines.*
- Definir fluxos complexos amb la interacció de diferents *pipelines.*
- Escriure el flux d'execució de les *pipelines* fent servir un codi de programació propi.
- Emmagatzemar com a qualsevol codi aquestes configuracions en un repositori de codi font.

Les configuracions de les *Pipelines* es guarden en fitxers de configuració específics en un format determinat anomenat: ***Jenkinsfile***.

Aquests fitxers poden ser editats manualment o bé a través d'una interfície gràfica. Jenkins disposa de diferents eines per a fer-ho, però potser la més habitual i bàsica és l'editor: "*Blue Ocean Pipeline Editor*".

Aquest editor permet, entre altres, les següents funcionalitats:

- Edició de *Pipelines* seguint un model visual basat en el concepte de "***WYSIWYG " - what you see is what you get"*** [12]**.**
- Declarar i modificar definicions de *Pipelines.*
- Creació de "*stages*". Cada *pipeline* es pot executar al llarg del temps passant per diferents etapes i estats.
- Configuració visual de la relació i interacció entre les *pipelines.*
- Configuració visual de l'execució en paral·lel de les tasques.
- Configuració de variables d'entorn.
- Configuració dels diferents agents que executen les *pipelines.*
- Detecció en temps real d'errors en la configuració que es mostren immediatament en l'editor mentre s'edita.

- Guardar la configuració en format de fitxers **Jenkinsfile.**

La següent figura mostra uns quants exemples de les pantalla principals de l'editor (font [13])..

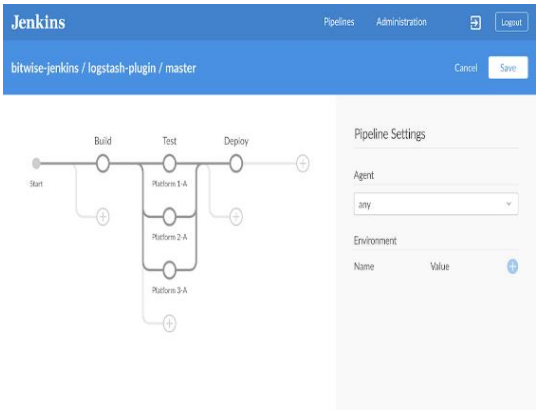


Figura 10: Jenkins - Edició i visualització de *pipelines*

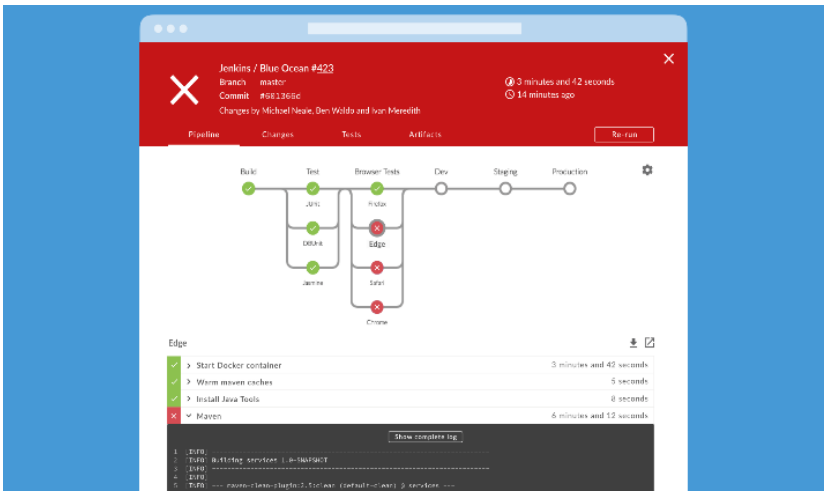


Figura 11: Jenkins - Visualització d'una execució fallida de les *pipelines*

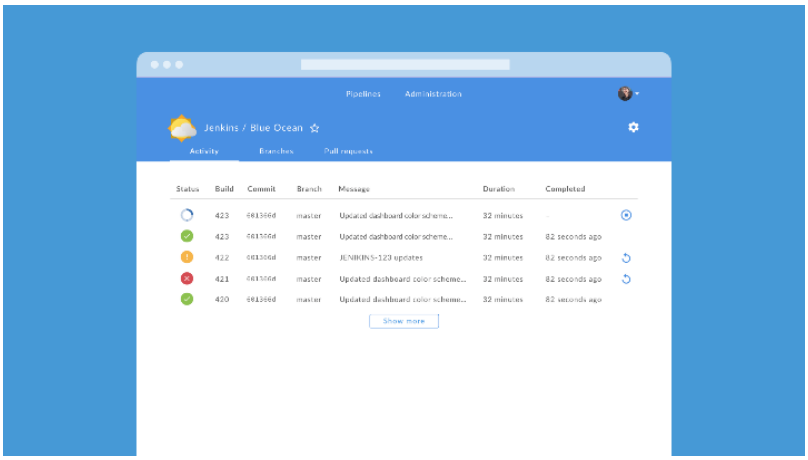


Figura 12: Jenkins - Visualització de l'activitat històrica d'una pipeline

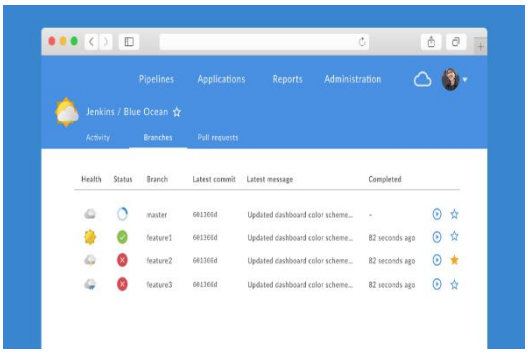


Figura 13: Jenkins - Visualització del estat de les branques

La idea principal és guardar i associar tots els fitxers de definicions als mateixos repositoris de codi font, lligat amb el codi font que s'està integrant. Així queden unides les versions de codi i les *pipelines* que han sigut executats i integrats en un moment concret. Això permet entre d'altres avantatges:

- Disposar automàticament dels mecanismes de *pipelines* per a cada versió i branca específiques que es vol integrar. No cal tornar a crear-los des de zero.
- Auditoria automàtica de les *pipelines* aplicades.
- Relació unívoca i de confiança entre les versions integrades i les *pipelines* involucrades.

3.1.3 Plataformes suportades i requisits del sistema

Jenkins és un software multi plataforma. Al ser programat en Java, pot pràcticament córrer en qualsevol sistema que suporti una màquina virtual Java. Pot ser instal·lat i executat de diferents formes i córrer en diferents sistemes operatius.

3.1.3.1 Mode d'execucions suportats

- Pot córrer i ser instal·lat com una aplicació independent (*stand-alone app*) dins d'un sistema operatiu suportat.
- Pot ser executat dins del servidor web propi incorporat amb Jenkins anomenat "Jetty".
- Pot córrer com un "*Java servlet*" dins d'un contenidor de servlet Java existent, com pot ser per exemple Apache Tomcat [14].
- Pot ser executat com un contenidor basat en tecnologia Docker [15].

3.1.3.2 Sistemes operatoris suportats

Jenkins pot ser instal·lat en els següents sistemes operatius:

- SO basats en Windows.
- SO basats MacOs.
- Linux (distribucions basades en Debian i Ubuntu).
- Entorns UNIX: Solaris ,OmniOS, SmartOS.

3.1.3.3 Arquitectura distribuïda

Un dels punts forts de Jenkins és la seva capacitat d'escalament, de créixer de forma sostenible, i poder cobrir les necessitats d'integració de l'organització. Jenkins està tant indicat per a petites empreses en les que amb 1 únic servidor es poden cobrir les necessitats, com per a grans empreses. En aquest cas, fent servir un model de *Master/Slave* Jenkins pot executar de forma distribuïda totes les diferents operacions de compilació, empaquetatge i *tests* en diferents nodes esclaus.

En aquesta configuració es deixa el node MASTER bàsicament al càrrec de:

- Administració de tot l'entorn del clúster composat pels diferents nodes esclau.
- Programació i assignació de les tasques: delega en els nodes esclau les operacions més costoses en termes de memòria i CPU. Així no es comprometen els recursos del node principal.
- Monitorització dels nodes: els pot posar *online* i *offline* sota demanda.
- Recol·lecció i presentació resultats.

L'assignació i distribució de les tasques en els nodes esclau es pot fer automàticament (el servidor assigna la tasca al primer node lliure que troba, de forma balancejada) o bé certs tipus de tasques es poden pre-assignar a uns nodes específics. En aquests nodes per exemple es poden instal·lar diferents sistemes operatius i entorns de compilació, específics per al tipus de projecte que es vol integrar.

La comunicació i execució dels nodes esclaus es poden fer a partir de diferents protocols i tècniques:

- Llançament de l'esclau a partir d'un agent client SSH. És el més habitual en entorns Linux/Unix.
- En entorns Windows, es poden llençar remotament els esclaus fent servir la tecnologia WMI i DCOM (disponible en les versions Windows Server 2000 i superiors).
- Es pot crear un script personalitzat que llenci les tasques a mida fent servir connectors disponibles en l'organització.
- Execució de l'agent esclau fent servir Java Web Start(JNLP).
- Fent servir també tecnologies de Virtualització com KVM (Kernel-based Virtual Machine) [16] es poden córrer múltiples nodes esclau en un mateix servidor. Dependrà dels recursos d'aquest servidor el correcte rendiment de l'arquitectura.

El següent gràfic il·lustra un senzill escenari d'un servidor *master* Jenkins que monitoritza el repositori GitHub de codi font, llençant les diferents tasques d'integració als nodes esclau multi sistemes operatius. La sortida del sistema és un report del test executat.

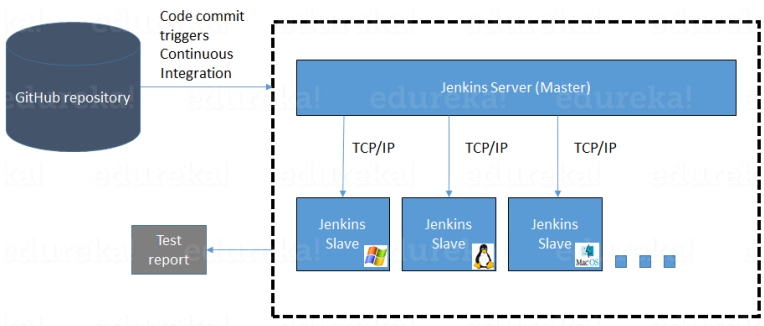


Figura 14: Exemple escenari plataforma Jenkins

3.1.4 Jenkins Plugins

L'èxit de Jenkins es basa també en la gran varietat de *plugins* que poden ser afegits al sistema per enriquir i facilitar encara més el treball. En la seva web es proporciona un cercador amb més de 1000 *plugins* diferents [17] desenvolupats per la seva comunitat de col·laboradors.

Aquets *plugins* són el mitja principal per suportar i donar resposta a les necessitats especifiques de cada organització, sent el mecanisme principal per a créixer en funcionalitats a partir de la instal·lació bàsica.

Es poden descarregar automàticament del repositori oficial i es disposa d'un servei d'actualitzacions i gestió de dependències.

3.1.5 Suport

Jenkins disposa de múltiples canals de suport online a més de la seva comunitat de desenvolupadors que col·laboren en el projecte i en els seus *plugins*. Altres recursos important són la seva WIKI [18], la seva comunitat Jira [19] on es poden veure les diferents tasques, activitats, *bugs*, etc de forma pública i un complet catàleg de documentació online [20]. Jenkins proporciona un Framework complet per a cobrir totes les necessitats d'una organització que pot interactuar amb una gran multitud de *plugins*, software, eines i altres plataformes àmpliament reconegudes pel mercat.

La següent il·lustració mostra un exemple de l'ecosistema que envolta Jenkins gràcies a les connexions amb múltiples plataformes a partir de *plugins* dedicats.

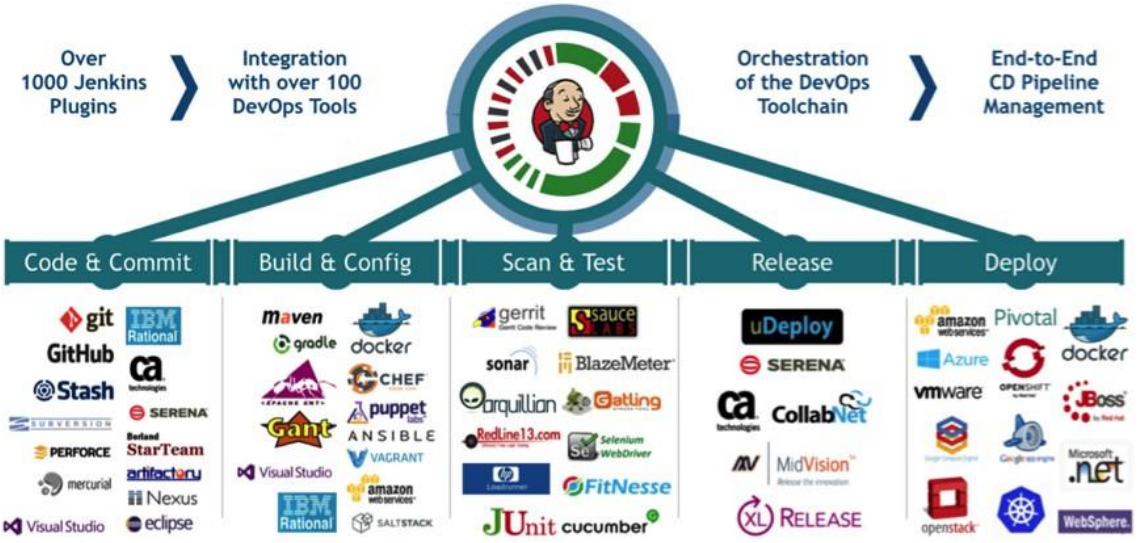


Figura 15: Ecosistema de Jenkins i connexió a altres plataformes (font)

3.2 Bamboo

3.2.1 Breu introducció històrica

Bamboo és un servidor de IC creat per la companyia de desenvolupament de software australiana Atlassian a l'any 2007 en el llenguatge de programació Java. Aquesta

compania, fundada al 2002, està especialitzada en eines de suport al procés de desenvolupament i gestió del software, eines d'administració i suport en general als projectes.

Els seus productes més famosos són l'eina Web de suport a la gestió de projectes i seguiment/enregistrament de *issues* JIRA [21] i l'eina Web de gestió del coneixement i treball col·laboratiu Confluence [22]. Totes dues eines, també desenvolupades en Java estan perfectament integrades entre si i s'acostumen a fer servir conjuntament.

Actualment s'han fet més de 25 lliuraments de diferents versions de Bamboo, des de la versió inicial 1.0 fins a l'última 6.2 publicada al setembre del 2017.

3.2.2 Caracteristiques principals

Bamboo té la capacitat d'automatitzar totes les tasques relacionades amb el procés d'integració i de proporcionar un mecanisme de lliurament continu als seus usuaris.

Com a valor afegit, per a les companyies que trien com a model de IC l'eina Bamboo, les 3 eines (Jira, Conflunce i Bamboo) poden funcionar de forma conjunta oferint total cobertura a la gestió dels projectes, documentació i enregistrament i IC.

Bamboo, al igual que altres productes de IC, monitoritza el repositori de Codi Font i gestiona automàticament totes les validacions del procés d'empaquetament i publicació de l'aplicació.

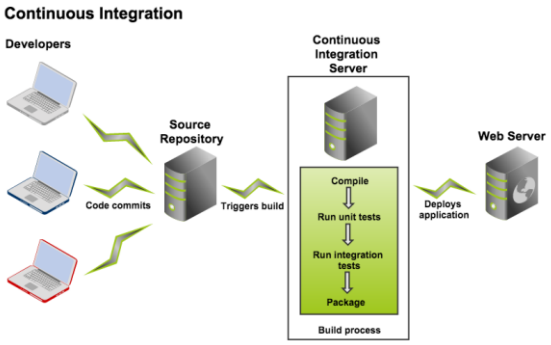


Figura 16: Model arquitectura bàsica Bamboo

3.2.2.1 *Organització del treball*

Bamboo es basa en la configuració de *workflows*. Aquests fluxos de treball modelen l'ordre, seqüència i comportament de totes les tasques. Les següents són les diferents entitats conceptuals amb les que treballa Bamboo:

- **Projecte**. Aquesta és l'entitat de treball de més alt nivell. Per a cada aplicació que es vol integrar, es defineix un projecte. Un projecte és a la vegada una col·lecció de Plans.
- **Pla**. El pla és la primera unitat d'agrupació que defineix un comportament específic dins de la *Pipeline*. Tot pla té com a minin una etapa o fase anomenat *stage*. Es poden definir múltiples *stages* i agrupar en ells l'execució de diferents treballs (*jobs)* d'una forma específica. En un pla es configuren característiques com:
  - Quin **repositori** de codi font es treballa.
  - En quin orde de seqüencia s'executen els *stages.*

- Quin *trigger* del repostori de codi font dispara els diferents esdeveniments.
- La dependència dels *triggers* entres els diferents plans.
- Les notificacions produïdes amb els resultats obtinguts.
- Usuaris amb permisos de lectura i d'edició del pla.
- Definició de les variables de sistema emprades al pla.

Cal ser usuari administrador per a poder editar els projectes i plans.

- **Stage**. Cada pla està composat d'un o més *stages*. L*'stage* representa un estat concret, dins d'una seqüencia de passos, en el que s'executen un conjunt de *jobs* específics. Cada *stage* té un o més *jobs* associats a diferents *steps*. Els *stages* es poden executar automàticament o bé ser disparats manualment. Respecte als *jobs*:
  - Es poden processar en paral·lel en múltiples agents.
  - S'han d'executar i completar tots els *jobs* en l'ordre especificat per a poder passar el següent *stage.*
  - Com a resultats s'obtenen *artifacts* que es poden fer servir com a entrada en *stages* posteriors.

  La figura de sota mostra un petit exemple de com es poden configurar *jobs* en diferents *steps*.

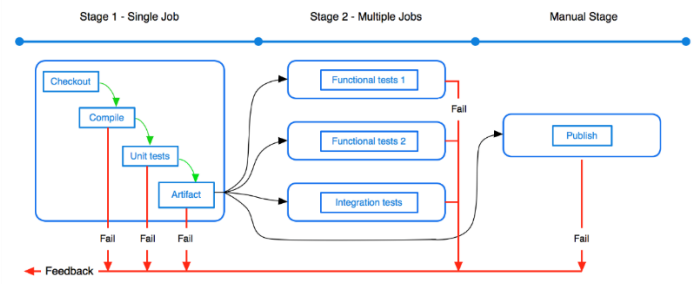


Figura 17: Exemple de configuració de jobs en Bamboo (font)

- **Job**. El *Job* és l'entitat que defineix 1 o N tasques que s'executen seqüencialment dins del mateix agent. Cada agent pot estar especialitzat en un tipus d'execució específic amb *capabilities* concretes. La forma en que estan definits determina l'ordre exacte en què s'executa cada tasca. En els *jobs* també es defineixen els *artifacts* produïts i les diferents etiquetes o *tags* reportades als *logs*.
- **Task.** Finalment la tasca és la unitat mínima de treball. Pots ser una acció, una execució d'un script, una validació d'una condició, etc. Les tasques s'executen dins el *Job* contenidor en un directori de treball únic per a l'agent en execució.

Aquesta figura resumeix segons el propi manual de Bamboo [23] les relacions i jerarquia de tots aquest elements anteriorment descrits:

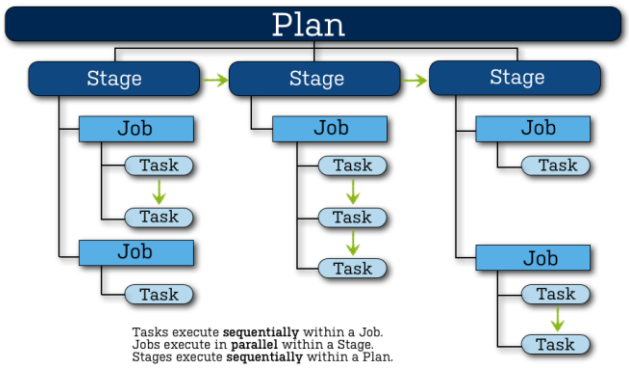


Figura 18: Relació entre *stage*, *task* i job a Bamboo

3.2.2.2 *Eina Dashboard*

Bamboo disposa d'una interfície d'accés web on es pot accedir a una *Dashboard*. Des d'aquí es poden accedir a 3 grans seccions:

- **MyBamboo**: Amb les últimes informacions dels plans executats, els favorits, resultats i estadístiques de l'usuari.
- **Build Plans**: Llista de tots els plans configurats conjuntament amb el resultat de la seva última execució.
- **Build Activity**: Visualitza informació del *build* que s'està executant i de l'estat de la cua de processament de Bamboo on estan en cua tots els *builds* que té Bamboo pendents de processar.

De forma molt visual es pot veure l'estat i informació relativa d'un projecte com: el número de *build*, l'estat, autor, última data d'execució, tests passats, motiu de l'execució, etc. La següent figura és un exemple:

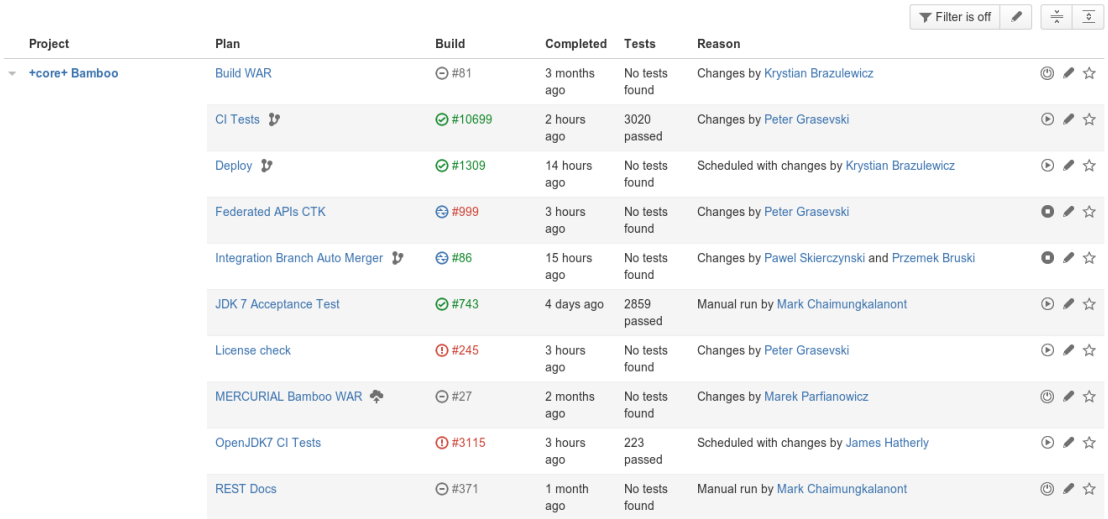


Figura 19: Panell d'estat del projecte a Bamboo (font)

La informació visual de l'activitat i l'estat de les execucions que ofereix Bamboo és un dels seus punts forts, donant en tot moment i de forma molt detallada l'estat dels processos. A més a més, per a cada *build* es guarden els logs complerts (*stack traces* en cas d'errors) que poden ser revisats en detall des del mateix navegador web.

Bamboo també disposa d'un panell informatiu "*wallboard*" que visualitza l'estat dels plans desitjats. Es mostra en color vermell els que han tingut algun error trencant la cadena d'integració i en verd tots els que estan en un estat correcte. Pot ser personalitzat i ampliat amb l'ús d'altres *plugins.*

Figura 20: Bamboo *Wallboard*

3.2.2.3 *Configuration as a code*

Bamboo recentment ha incorporat la possibilitat de crear la configuració de tots els plans com a codi. Fins ara (versió 6 cap endarrere) només es podia crear i administrar els plans a Bamboo mitjançant la seva interfície Web. En les últimes versions és possible crear un codi font, en un llenguatge específic, que permet guardar tota aquesta configuració. Això permet integrar la configuració en altres editors de desenvolupament (IDEs) i automatitzar i estandarditzar encara més la configuració. Al ser codi, es poden aplicar tots els conceptes de programació clàssics com: re usabilitat, parametrització, encapsulació en mètodes, iteracions, versionatge, etc. Si es fa servir un IDE encara s'enriqueix més amb: autocompletar, validació de sintaxis, eines de refactoritzacio, etc.

3.2.2.4 *API REST*

El servidor de Bamboo ofereix pels desenvolupadors i administradors una interfície REST API que permet una interacció externa amb els recursos interns (*Data Entities*) del servidor de Bamboo. Es pot programar un script que faci crides GET al servidor REST que tornaran objectes en format XML o JSON. El API fa servir un protocol totalment estàndard, permetent accedir des de qualsevol client compatible amb REST. Per exemple es poden accedir als següents recursos remotament:

- Llista de projectes i llista de plans que els composen.
- Detall de cada pla i totes les accions disponibles.
- Resultats de cada *build* d'un pla i projecte en especial.
- *Artifacts* obtinguts.
- Diferents tipus de reports.

Per exemple es pot accedir a l'API que manipula la cua dels *builds* i es pot invocar de forma remota un nou *build* que es posarà a la cua per a ser processat.

Amb aquesta API es poden integrar *plugins* o altres eines externes que mostrin en un panell o *dashboard* informacions del servidor.

3.2.2.5 *Integració amb branques features*

En el procés de desenvolupament és normal que es necessitin temporalment branques, de tipus *features,* on es programen i es proven noves funcionalitats que encara no es volen integrar per algun motiu en la *master*. El problema és que moltes vegades es vol

també integrar aquesta *feature* fent servir les pipelines definides al nostre sistema de IC que s'aplica normalment a la branca *master*. En molts servidors de IC si es vol integrar una branca a mida, cal copiar tota la configuració a mida i a ma en la nova, sent poc fiable i flexible.

Amb Bamboo, si es fa servir GIT como a gestor del codi font, es poden mapar directament les branques/*features* existents al repositori GIT als plans de compilació definits a Bamboo. Es pot, per exemple, alhora de fer el *build* d'un projecte, triar de forma manual o automàtica la branca desitjada i aplicar tota la configuració del pla que es tenia configurat per la *master*. Això porta un pas més enllà el concepte d'integració estenent un model d'Integració Continua automàticament a les branques/*features* que els desenvolupador vagin creant sota demanda dins el seu procés de desenvolupament.

3.2.2.6 Model d'execució d'agents

El servidor de Bamboo corre en un servidor central on es coordina tot el procés d'integració dels diferents plans. El agents que s'encarreguen d'executar els processos de *build*, poden ser executats en diferents entorns, depenent del requisits de l'organització, la càrrega del servidor i el model d'escalament que es vulgui assolir. Els agents es poden executar com a:

- ***Local agent***. L'agent s'executa dins el mateix servidor Bamboo, dins la mateixa Màquina Virtual Java (JVM). Es poden tenir tots els agents locals que permetin els recursos físics o virtuals del servidor.
- ***Remote agents***. Es pot escalar el sistema executant agents en altres servidors remots que tenen instal·lat l'entorn necessari.
- ***Elastic agent***. Fent servir un *driver* específic és poden executar agents remots a l'entorn Amazon Elastic Compute Cloud (, EC2).

Com a punt important a considerar, el cost de les llicències varia principalment en funció del nombre d'agents remots desitjats. Els agents locals no tenen cap sobrecost en la llicència de Bamboo.

3.2.3 Plataformes suportades

Bamboo pot ser instal·lat i executat en diverses plataformes i es pot connectar a diferents gestor de Bases de Dades. Bamboo és accessible a traves d'una aplicació web i guarda totes les configuracions en una Base de Dades dedicada.

En la següent llista es detallen breument les plataformes suportades en l'última versió 6.2 de Bamboo.

Plataformes Java : Oracle i Open JDK Java 1.8.

Sistemes Operatius

- Windows.
- Linux.
- Solaris.
- MacOS/OSX.

Gestor de Bases de Dades

Només per a finalitats d'avaluació es pot fer servir la base de dades incorporada a Bamboo : HSQLDB (HyperSQL DataBase). Per a entorns productius cal fer servir algun dels gestors següents:

- MySQL – fent servir el connector JDBC Connector/J 5.1:
  - MySQL 5.6.3.
  - MySQL 5.7.
- Microsoft SQL Server - fent servir el *driver* Microsoft's JDBC.
  - SQL Server 2012.
  - SQL Server 2014.
  - SQL Server 2016.
- Oracle 12c.
- PostgresSQL – fent servir el propi *driver* JDBC de Bamboo.

Navegador i Servidors Web

Bamboo ha d'executar-se dins del navegador Tomcat incorporat en la distribució. Només es dona suport per a aquest servidor. Els següents navegadors WEB (últimes versions) estan suportats per a accedir a la interfície web de Bamboo:

- Mozilla Firefox.
- Google Chrome.
- Safari.
- Microsoft Edge.
- Microsoft Internet Explorer 11.

Repositoris de codi font

Bamboo es connecta i monitoritza diferents plataformes de repositori de codi font. Les següents són les principals suportades:

- GIT, versió 1.8.1.5 o posterior.
- GITHUB.
- Mercurial 1.7.
- Subversion 1.5 i superiors..
- CVS.
- Bitbucket: (servidor i *cloud*)

Comentar que Bitbucket [24] és un producte propi de Atlassian que fa servir GIT com a gestor de codi oferint emmagatzemament web pels projectes de l'organització, similar a GITHUB.

Suport d'agents de *build*

Bamboo permet de sèrie compilar i empaquetar codi amb múltiples eines com:

- Apache Ant.
- Maven.
- Make.
- MSBuild.

Suport per a agents de *test*

Bamboo pot executar de sèrie validacions escrites per a les següents eines de *test*:

- PHP Unit.
- JUnit.
- Selenium.

Tot i així moltes altres es poden afegir fent servir *plugins* específics de tercers.

3.2.4 Cost de les llicències

Bamboo s'ofereix amb un període de prova gratuït de 30 dies amb suport inclòs. Pot ser estès fins a 90 dies. Un cop esgotat, la política de llicència es basa en la capacitat d'execució (en el nombre d'agents simultanis que es volen), en comptes del nombre d'usuaris que el fan servir. Es disposa de llicències per a ús acadèmic i comercial.

Els plans estan separats en dos grups: per a petits equips o per a grans equips en creixement. En ambdós casos es dona suport + actualitzacions + *bugfix* inclòs de 12 mesos.

El preu més bàsic comença per a un petit equip des de 10$ anual (10 *jobs*, agents locals il·limitats, no agents remots) + 10$ per any de suport addicional.

Per a equips més grans, la quota de nova llicència dona dret a fer servir el producte infinitament. El preu mínim és de 880$ amb 1 agent remot i *jobs* i agents locals il·limitats). Si es vol ampliar a 5 agents remots per exemple, el preu s'incrementa fins a 2420$. S'ofereixen llicències fins a un màxim de 250 agents remots. El preu del manteniment i suport és proporcional al nombre d'agents remots contractats. Aquest suposa per any la meitat del preu total dels agents. El servei de manteniment i suport és opcional i no condiciona el funcionament de la llicència adquirida.

3.3 Team City

3.3.1 Breu introducció històrica

Team City és un servidor de Integració Continua, desenvolupat per la companyia JetBrains [25], en el llenguatge de programació Java. Va ser creat a l'any 2006 i es distribueix sota una llicència comercial propietària. JetBrains està especialitzat en el desenvolupament d'eines (IDEs) i extensions que s'integren dins altres IDEs del mercat com per exemple VisualStudio. Aquestes extensions donen suport als programadors en múltiples llenguatges de programació i plataformes.

Actualment l'ultima versió de TeamCity és la versió 10.1 (equivalent a la 2017.1 segons el seu sistema de versiontge partícular).

3.3.2 Característiques principals

3.3.2.1 Conceptes i organització del treball

Al igual que altres servidors de IC, TeamCity treballa amb conceptes molt similars alhora d'organitzar el seu funcionament intern.

- **Servidor**. Entitat central que monitoritza el sistema i totes les connexions amb els agent de *build*. S'encarrega d'assignar i distribuir la feina en funció de la disponibilitat i compatibilitat dels agents i de reportar els resultats. Tota la informació i *artifacts* es guarden en la seva base de dades.

- **Build Agent:** És l'aplicació encarregada exclusivament d'executar *builds*. En TeamCity aquestes poden córrer en múltiples ubicacions i sistemes operatius.
- **Project**. És la unitat d'organització a més alt nivell. Equival a un projecte de software específic o a una versió d'ell. Els projectes a la vegada són una col·lecció de configuracions d'agents de *build*.
- **Build Configuration**. És una combinació de configuracions que defineixen un procediment de *build*. Es configuren, entre d'altres, les branques del repositori a monitoritzar, els *steps* del *build* i els *triggers* que els llençaran.
- **VCS Root.** Representa una col·lecció de paràmetres que defineix com es connecta i què es monitoritza d'un servidor de control de versions (repositori de codi font central). Es consideren: *urls*, *paths*, credencials, mode de *checkout*, etc.
- **Build Step**. Els *steps* representen cadascuna de les tasques, en un ordre específic, que s'executaran de forma seqüencial. Per a cada *step*, s'assigna un *build runner*, que el connecta amb una eina d'integració que s'encarregarà del *build* en qüestió. El resultat d'un *step* pot servir com a entrada per al següent i pot condicionar o no la seva execució en funció del seu resultat.
- **Build Runner.** És l'agent encarregat d'executar la part d'integració amb una eina de build en concret: MSBuild, Ant, Maven, etc. El llistat de *runners* és força extens, abastant un gran conjunt de plataformes distintes [26].
- **Build trigger**. Regla que dispara l'inici del procés de *build*. Pot ser automàtica, manual, sota certes condiciones, etc.
- **Build Artifact.** És el resultat obtingut al final del procés de *build*: llibreries, logs, reports, instal·ladors, etc.

3.3.2.2 Flux de treball

En TeamCity, per a cada procés d'integració es segueix el següent flux de treball:

1. El servidor monitoritza el repositori de codi. Per a cada canvi que detecta, inserta aquest canvi en la seva base de dades.
2. D'acord amb la configuració del *triggers* que disparen els *builds*, es detecta el canvi succeït la BD i s'afegeix el *build* a la cua.
3. El servidor cerca el primer recurs (agent de *build*) disponible que sigui compatible i l'assigna (pot ser local, remot, *cloud*, etc).
4. L'agent executa tots els *steps* configurats i va enviant en temps d'execució al servidor tota la informació sobre el progrés, permetent un monitoratge en temps real.
5. Un cop finalitzat el procés, si tot és correcte, es produeixen els *artifacts* corresponents i són enviats al servidor.

3.3.2.3 Control de qualitat del codi

TeamCity suporta diferents tècniques que ajuden a millorar la qualitat del codi, en temps real o sota demanda. Aquestes comprovacions validen i identifiquen possibles problemes al codi alhora de fer el *build* i suggereixen millores i re factoritzacions.

Tant per a Java amb IntelliJ IDEA i per a .NET amb ReShaper o dotCover es poden realitzar controls de codi duplicat, "*code coverage*" per a mesurar el grau de cobertura dels *tests* incorporats al codi font, anàlisis estàtic del codi i detecció de potencials problemes i degradacions en els procés de *build,* entre d'altres.

3.3.2.4 Pre-tested Commit

Una tècnica molt útil de TeamCity és la capacitat de validar i integrar el codi mentre s'està treballant sense la necessitat de realitzar un *commit* a la branca del respositori de codi font. La majoria de servidors de IC monitoritzen les branques del repositori central i necessiten un *commit* per a engegar el procés de validació. TeamCity implementa el patró de "*Gated commit build*" amb la tècnica de "*pre-tested commit*". A la pràctica consisteix en retardar el *commit* real a la branca del repositori fins que s'ha fet la validació completa amb tot el procés habitual d'integració al servidor de IC. D'aquesta forma es persegueix minimitzar la probabilitat de trencar el codi. En un esquema clàssic si el codi no és correcte durant el procés de validació, malauradament ja està al repositori. Amb TeamCity el procés és com s'il·lustra en la següent figura:

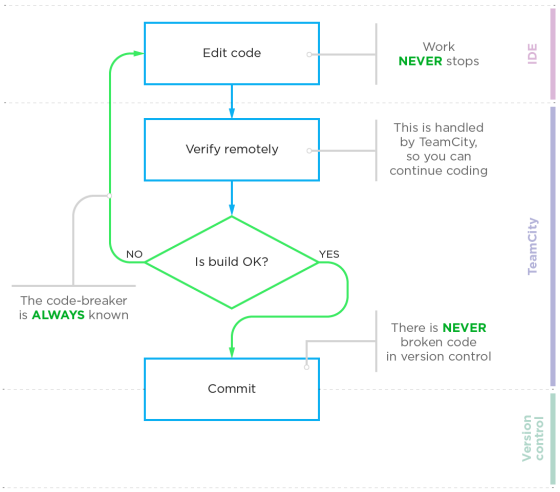


Figura 21: Flux de pre-validació de TeamCity

El codi ha d'estar "OK" per a produir-se el *commit* real. Per a aconseguir aquesta tècnica TeamCity interactua automàticament amb el repositori per a validar el *commit* en una branca temporal, llençant tots els *builds* necessaris com si fos un cas real i finalment fer un *auto-commit* si el procés és correcte.

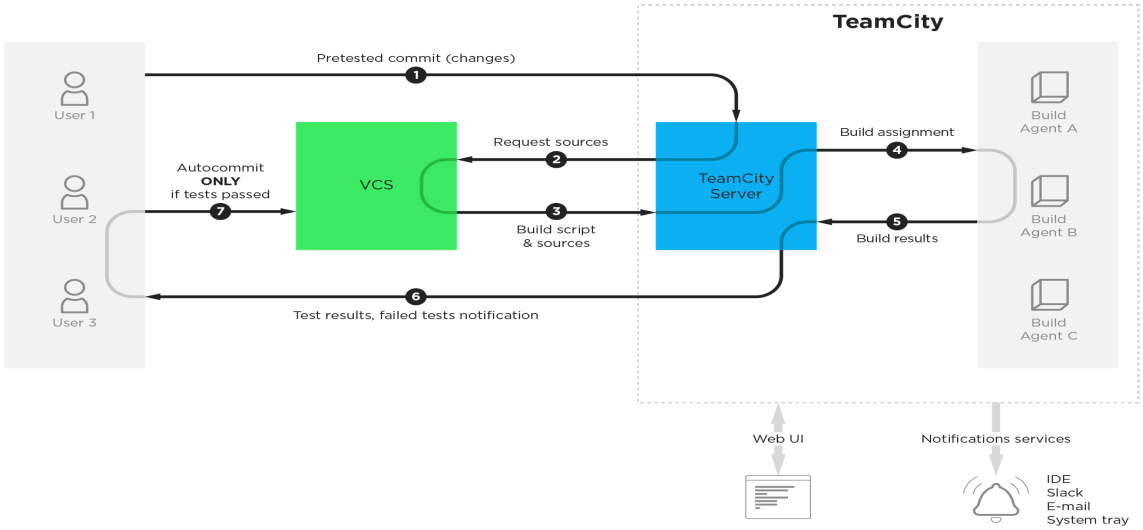


Figura 22: Arquitectura TeamCity per a pre-validar el codi

3.3.2.5 *Integracions al núvol*

Team City permet nativament configurar i integrar diferents plataformes basades en el núvol. Quan les necessitats de computació del servidor de IC central creixen, TeamCity pot administrat de forma automàtica recursos remots basats en les següents plataformes:

- **Amazon EC2**. A partir de la càrrega de la cua interna de *builds*, pot automàticament executar sota demanda nous agents a la plataforma d'Amazon. Enregistrant el compte amb els paràmetres corporatiu, pot aixecar, suspendre i resumir les màquines virtuals en funció de la càrrega necessària del servidor.
- **Microsoft Azure**. Es pot configurar una granja de recursos remots a Azure on s'executaran instàncies d'imatges de màquines virtuals Linux o Windows. Dins d'aquestes màquines, en funció de la demanda i de la configuració dels agents, s'executaran tots els *jobs* necessaris. TeamCity, a partir d'un plugin especialitzat, serà capaç de gestionar quines instàncies s'activaran gestionant en tot moment la distribució de la càrrega de la cua dels *builds*.
- **VMWare vSphere**. Amb aquesta tecnologia es poden executar agents de *build* externs que corren en màquines virtuals en un *cloud* local o remot. Com si fossin recursos locals disponibles pel servidor, TeamCity analitzarà la càrrega i aixecarà les màquines virtuals WMWare necessàries.

3.3.3 Suport multiplataforma

Entorns d'execucio. TeamCity suporta multitud de plataformes, tant de desenvolupament com d'eines d'integració i sistemes operatius. Les funcionalitats del nucli de TeamCity són independents de la plataforma on s'executin, funcionant en qualsevol sistema operatiu que pugui córrer les següents plataformes Java (JRE):

- Oracle Java 8 i superior, 32 or 64 bit (64 bit recomanat per a producció).
- OpenJDK 8. 32 or 64 bit

TeamCity server s'executa com una aplicació Java J2EE que es pot instal·lar a qualsevol servidor d'aplicacions compatible. Les versions instal·lables ja incorporen el servidor recomanat Tomcat 8.5.

Sistemes operatius. La majoria de versions recents dels sistemes operatius estan suportats, com per exemple:

- Linux (Ubuntu, Debian, RedHat, SUSE, i altres).
- macOS.
- Windows 8, Windows 10 amb Tomcat 7.
- Windows XP , Windows 7/7x64.
- Windows Server 2008, 2012, 2016.
- Server Core installation of Windows Server 2016.

Execució *Build* Agents. Els agents són en aquest cas aplicacions independents "*stand-alone*" que s'executen instal·lades en els mateixos sistemes operatius suportats amb alguna de les versions Java:

- Oracle Java 6-8, (Java 8 és recomanat).
- OpenJDK 6-8 és suportat, però Oracle JDK és el recomanat.

Suport Navegadors Web

- Google Chrome.

- Mozilla Firefox .
- Safari en entorns Mac.
- Microsoft Edge.
- Microsoft Internet Explorer 9+ .
- Opera 15+.

Suport Frameworks de *Build*

El llistat de *Frameworks* de *build* suportat és força ampli cobrint els principals per a les plataformes Java i .Net. A continuació es llisten els més significatius:

- Java
  - Ant 1.6 – 1.9.
  - Maven 2.0.x, 2.x, 3.x.
  - IntelliJ IDEA Project, entre d'altres.
- .Net
  - MSBuild
  - Microsoft Visual Studio(2013-2017)
  - Nant
  - Nuget, entre d'altres.

Suport Repositoris de Codi Font

- Git amb Git Client instal·lat al servidor.
- Subversion (1.4-1.9 i superiors).
- Team Foundation Server 2005, 2008, 2010, 2012, 2013, 2015, 2017 .
- Mercurial .
- CVS.
- SourceGear Vault 6 and 7 .
- Borland StarTeam 6 i superior.
- Microsoft Visual SourceSafe 6 i 2005, entre d'altres.

Support integració a altres IDEs

Amb la utilització d'alguns *plugins* addicionals, es pot connectar amb:

- Eclipse version 3.8 i 4.2-4-6 amb Java 1.5+.
- ItelliJ Platform, compatible amb al plataforma IDEA 13.0.x - 2017.2.x de JetBrains.
- Microsoft Visual Studio versions 2010, 2012, 2013, 2015, 2017 .

A [27] es pot trobar més informació del suport de plataformes.

3.3.4 Cost de les llicències

TeamCity proporciona diversos tipus de llicències que s'adapten a les necessitats de cada organització en funció dels plans de *build* i agents necessaris.

La versió de servidor Professional, es pot descarregar gratuïtament i suporta fins a 20 plans de configuració, accés a totes les funcionalitats del producte, suport estàndard al fòrum de la comunitat i 3 agents de *build*.

Es poden adquirir també llicències de *builds* addicionals sota demanda, a partir de 299€.

La versió de servidor Enterprise, dona accés a un nombre il·limitat de plans de configuració, 1 any gratuït d'actualitzacions, suport prioritari per correu i un nombre finit d'agent de *build*. El preu final és en funció del *builds* adquirits: mínim 3 des de 1999€, fins 100 agents de *build* per 21999€. El cost de les renovacions de les subscripcions anuals, també va en funció dels agents contractats i és aproximadament la meitat del preu de nova llicència: renovació 1 agent 149€, renovació Enterprise 3 agents 999€, Enterprise 100 agents 10999€.

Altres llicències són:

- Startup licenses. Per a noves organitzacions *startups* es pot aconseguir un descompte del 50%.
- *Enterprise Evaluation license:* es pot fer servir TeamCity amb totes les característiques de la versió Enterprise amb una llicència de proves de 60 dies.
- Gratuïta per a projectes *OpenSource*. Es pot gaudir si es compleixen certs requisits de llicència gratuïta per un període concret i limitat, si la finalitat d'us és per a un projecte basat en *OpenSource*.

3.4 CircleCI

3.4.1 Breu introducció històrica

La companyia CircleCI amb seu a San Francisco, va ser fundada l'any 2012. Des dels seus començaments el seu primer objectiu ha sigut crear eines per a donar suport als desenvolupadors en el procés de la Integració Continua. Estan especialment enfocats en la velocitat i la ràpida capacitat de publicació. El seu principal i únic producte és l'aplicació d'Integració Continua que porta el mateix nom, CircleCI. Treballen principalment en aquest producte i tenen un equip de treball de més de 100 persones. Actualment el producte ha evolucionat de la versió 1.0 i es troba en la v 2.0.

3.4.2 Característiques principals

El producte CircleCI està enfocat en dos grans models: instal·lació + execució al *cloud* de CircleCI o bé instal·lació *on-premise* als servidor o *clouds* propis de l'organització del client.

3.4.2.1 Arquitectura

CircleCI està format per dos components principals: Serveis i *Builders.* El servei és l'aplicació principal que s'executa en una única instància. S'encarrega de les funcionalitats principals de gestió de l'aplicació i de les tasques, de l'emmagatzemament i de les comunicacions per xarxa. A més a més CircleCI pot gestionar un nombre definit i limitat de *builders,* que són les instàncies que fent servir diferents contenidors interns executen els *jobs* corresponents. Aquests *builders* és comuniquen amb protocols específics amb el servei principal per a informar dels resultats obtinguts. Ambdós components necessiten comunicar-se amb el gestor de versions per a poder realitzar la seva feina.

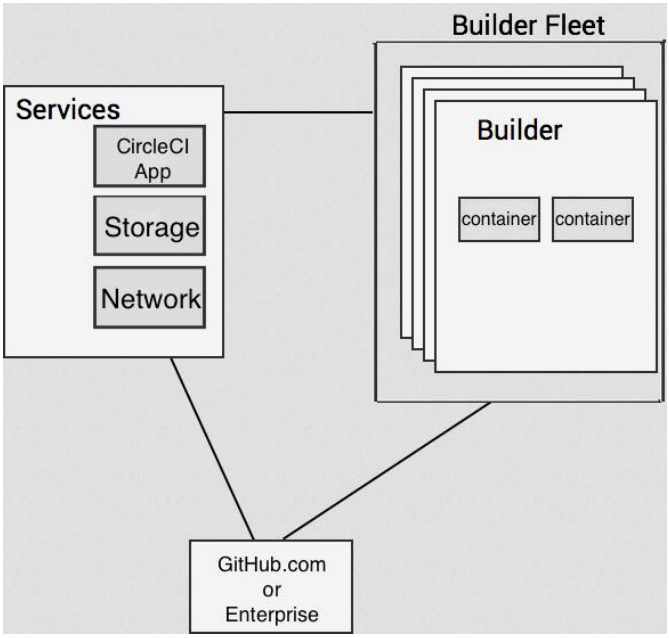


Figura 23: Arquitectura bàsica CircleCI

El servei s'ha d'executar en una màquina persistent que no es reiniciï cada cop i que necessita suport de backup tot i que s'executi en un entorn de màquina vsirtual. En canvi les instàncies dels *Builders* no necessiten guardar el seu estat ja que només viuen durant l'execució del procés de *build* i s'apaguen en la seva finalització. Cal configurar i permetre en l'entorn de xarxa on s'executen i al *firewall* els diferents fluxos de comunicacions SSH, HTTP/S i d'altres que es faran servir en la comunicació entre els 3 components.

3.4.2.2 Workflows, jobs i steps

Un *workflow* és un conjunt de regles que defineixen una col·lecció de jobs. Aquests s'executen en un ordre específic o en paral·lel per a aconseguir els resultats el més ràpid possible. Els jobs són a la vegada una col·lecció d'*steps* on cada *step* és un conjunt de comandes executables. Tots els *steps* del *job* s'executen dins de la mateixa instància de contenidor de CircleCI. La configuració es guarda en fitxers de format específic (YAML) anomenats `config.yml`.

L'execució dels workflows en CircleCi està optimitzada per tal de rebre en cas d'error en algun job en concret una notificació en temps real. D'aquesta forma si cal es pot tornar a executar aquest *job* exclusivament sense la necessitat de tornar a executar el *build* complet i tots els seus jobs i *tests* de nou. Aquesta particularitat fa CircleCI força més optimitzat que altres productes.

Els workflows es poden classificar i executar de diverses formes. Es disposa d'una interfície web força senzilla que permet visualitzar l'estructura del workflow, controlar l'execució i consultar els resultats.

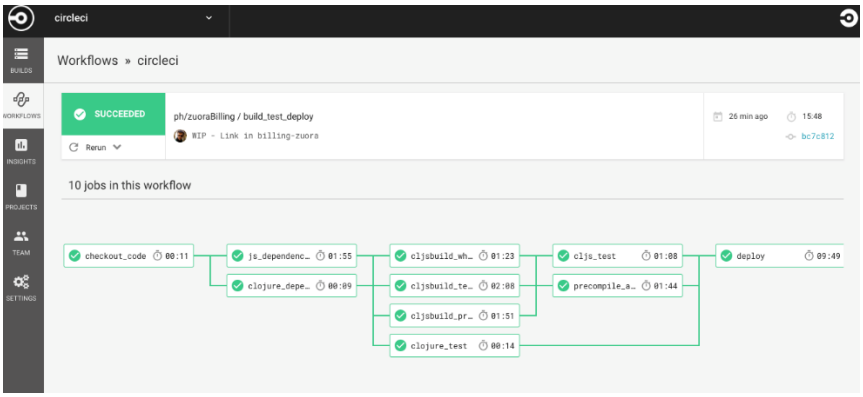


Figura 24: CircleCI visualització dels detalls d'un *Workflow*

En funció de les necessitats, els *workflows* es poden organitzar i executar de forma:

- **Seqüencial**. S'espera el resultat del *job* anterior per a continuar amb el següent.
- **En paral·lel – *Fan-Out/Fan-in***. Per exemple després d'un procés de *build* es poden llençar N tests d'acceptació en paral·lel i al final una nova tasca per a validar el conjunt de resultats i fer la publicació corresponent.
- **Sota confirmació manual**. Es poden configurar per a que en un moment donat un *job* es quedi pendent d'autorització (en *Hold*) per a poder continuar. En la interfície d'usuari el *job* quedarà a l'espera demanat el permís de l'usuari per a continuar.
- **Programada**. Es pot programar per a que es dispari un *trigger* que executi en un moment donat el *workflow*, per exemple en moments de poca càrrega del sistema, en horari nocturn, etc.

Els *workflows* es poden personalitzar a nivell de cada branca i es poden configurar filtres per tal d'executar o descartar *jobs* a partir de *tags* publicats en un commit del repositori de codi font. Una altra característica és que es pot definir un *workspace* local com a mitjà de compartició d'informació entre els diferents *jobs*, que només existirà per a una execució del workflow específica.

3.4.2.3 Entorns i mode d'execució

CircleCi està enfocat per a executar-se en entorns **virtuals i al *cloud*.** La majoria de les seves funcionalitats prenen avantatge d'aquest mode d'execució, aconseguint un producte molt eficient i de fàcil escalabilitat.

En entorns basats en el *cloud* d'Amazon (AWS EC2), es pot instal·lar l'aplicació de 3 formes diferents:

- **Single-box.** Per a petits equips o *demos*, on s'executa la plataforma i els *builders* en 1 única màquina virtual d'Amazon.
- **Clustered installation**. Enfocada per a la majoria d'entorns productius on s'executen els *builders* en un clúster de N màquines.
- **Highly-available configuration.** Amb la llicència corresponent es pot configurar CircleCI en mode d'alta disponibilitat. Consisteix principalment en externalitzar la BD local que corre en la mateixa màquina de servei i instal·lar-la en instàncies

dedicades de Base de Dades de AWS. Es fan servir 3 servidors nous en rèplica per a MongoDB i una altra instància autoescalable de PostgreSQL al *cloud* AWS. Requereix coneixements molt específics d'aquests tipus de components a Amazon.

Amb CircleCI es poden triar 3 tipus d'entorns d'execució:

Docker. Es pot fer servir un contenidor docker per a executar els jobs. Cada contenidor serà una instància de la imatge Docker que farà justament la feina que ha de fer. Les imatges Docker es poden optimitzar creant-les a mida carregant en elles només el que es necessita per a fer el *build*. També es poden tenir pre-instalades les eines externes necessàries en la imatge i així minimitzar en el seu ús el temps d'instal·lació d'aquests components addicionals. CircelCI manté un repositori d'imatges Docker [28] pels llenguatges i SGBDs més populars (PHP, Java, Node.js, PostgreSQL, etc) que es poden fer servir en qualsevol moment. Es poden utilitzar també imatges privades a mida fetes per tercers o per la pròpia organització.

Machine. En aquest entorn els *jobs* s'executen en màquines virtuals Linux volàtils (d'un sol ús). Aquest mode és l'indicat si es vol permetre accés directe als recursos típics del sistema operatiu o altres utilitats Linux instal·lades. Es podrà executar qualsevol aplicació que funcioni en un entorn Linux.

MacOS. Amb aquest *executor* es poden executar els *jobs* en sistemes operatius MacOS amb Xcode instal·lat.

3.4.2.4 Debug de Jobs per SSH

En certes circumstàncies es necessari poder accedir en viu a l'entorn on s'executa el *build* amb la finalitat de debugar, de consultar el progrés o de saber que ha passat. Si es fan servir contenidors Docker, CircleCI pot enllaçar un accés per terminal SSH amb la instància de la imatge Docker en execució. Això pot permetre accedir al sistema de fitxers, *logs*, processos, etc per a consultar. Es poden configurar els *builds* per tal d'escoltar connexions SSH remotes fins a 30 minuts després de finalitzar el procés de *build*. Un cop expirat el temps, la màquina virtual que executa el contenir s'apagarà.

3.4.2.5 Execució de Jobs amb l'API Rest

Com altres productes, CircleCI exposa una petita API Rest que permet de remotament des d'un client compatible gestionar l'execució de *jobs*. Es podran iniciar, reiniciar i cancel·lar *builds* i consultar informació bàsica referents als *builds* disponibles. No es poden, de moment fer, operacions a més alt nivell com per exemple l'execució d'un workflow complet.

3.4.2.6 Optimització dels recursos d'execució

Es poden configurar diversos paràmetres relacionats amb els recursos associats a l'execució dels *jobs* i els contenidors. Per exemple amb CircleCI es pot configurar el grau

de **concurrència** que gestiona quantes instàncies de contenidors es poden fer servir a la vegada per a executar múltiples *builds* concurrents. Aquest paràmetre està evidentment limitat pel nombre disponible de contenidors que permet la llicència. En entorns Linux a més es pot configurar el grau de paral·lelisme amb el qual es poden executar i distribuir diferents *build* de *tests* repartits en múltiples contenidors a la vegada. Això permet accelerar considerablement la velocitat amb que s'executen grans lots de *tests* en paral·lel. Per descomptat, cal considerar la limitació de contenidors que la llicència suporti. CircleCI també permet, depenent de la llicència, configurar els recursos de CPU i de RAM dedicats a *jobs* específics permetent una configuració a mida per a processos de *build* delicats o que requereixen recursos molt particulars.

3.4.3 Suport de plataformes

CircleCi es pot connectar amb menys gestors de versions que altres productes similars. La llista de proveïdors suportats també varia en funció del tipus d'instal·lació, però en general es pot connectar a GitHub, GitHub Enterprise i Bitbucket.

A nivell de llenguatges i entorns de *build*, en principi es suporten mitjançant imatges Docker predefinides per CircleCI, llenguatges de programació com Java, Php, Node.js, Python i Ruby entre d'altres. Addicionalment el fabricant també proporciona un mecanisme de compatibilitat independent del llenguatge de programació emprat, sempre que aquest sigui capaç de ser compilat en Linux o MacOs. Això amplia el suport a altres plataformes com C++, Javascript i .NET. Tot i així no és una opció recomanada per a llenguatges de la plataforma .NET. al no ser un suport directament de forma nativa per CircleCI.

3.4.4 Model i cost de llicències

CircleCI és un producte amb llicència propietària. Es disposa d'una petita versió gratuïta de l'aplicació amb capacitats limitades per a entorns de *build* en Linux. El model de llicència depèn de l'entorn on es volen executar els *builds* (Linux o MacOs) o si es vol una solució particular per a ser instal·lada darrera el tallafocs corporatiu. En funció de l'entorn triat, el cost de llicència està basat en magnituds relacionades amb la capacitat com per exemple el nombre de contenidors, nivell de concurrència, nombre de *builds* o minuts d'ús, usuaris, etc. A continuació un petit resum:

- Entorns Linux
  - El primer contenidor és gratuït amb 1500 minuts de *build*/mes com a límit.
  - Contenidor addicional a 50$/mes, sense límit de minuts de *build*.
  - Es pot configurar el nivell de paral·lelismes i concurrència.
- Entorns MacOs
  - Aquí només es disposa d'un pla de proves de 2 setmanes gratuït.
  - El cost varia en funció de les 4 següents propostes:

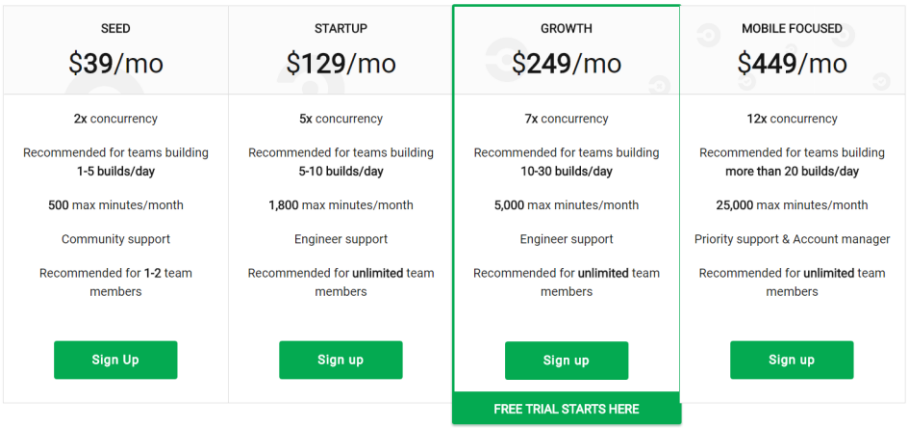


Figura 25: CircleCI MacOs licensing

- Mode darrera *firewall*
  - El cost varia en funció dels usuaris: 35$/User/mes amb contracte anual.
  - Per a mes de 100 usuaris, cal consultar condicions.

3.5 GoCD

3.5.1 Breu introducció històrica

GoCD és un servidor complet d'Integració Continua creat per la companyia de software ThoughtWorks [29]. Inicialment el producte de IC de la companyia s'anomenava CruiseControl i va ser publicat l'any 2007. Més endavant es va fer una completa evolució del producte al 2010 i es va canviar el nom per GoCD. En l'any 2014 es va prendre la decisió d'oferir el producte completament com a *open source* a la plataforma GITHUB.

GoCD està especialitat en com es controlen i configuren les *pipelines* de tal forma que resulta molt fàcil i senzill treballar amb escenaris i fluxos complexos. GoCD també implementa una bona gestió per a eliminar colls d'ampolla amb l'execució de tasques en paral·lel. L'aplicació està escrita, com molts altres servidors de IC, en el llenguatge de programació Java.

3.5.2 Característiques principals

3.5.2.1 Arquitectura

L'aplicació GoCD consisteix en la instal·lació i execució de dos tipus de components principals, el servidor i l'agent. El servidor pot controlar un gran nombre d'agents i es necessita com a mínim un per a poder realitzar qualsevol feina útil.

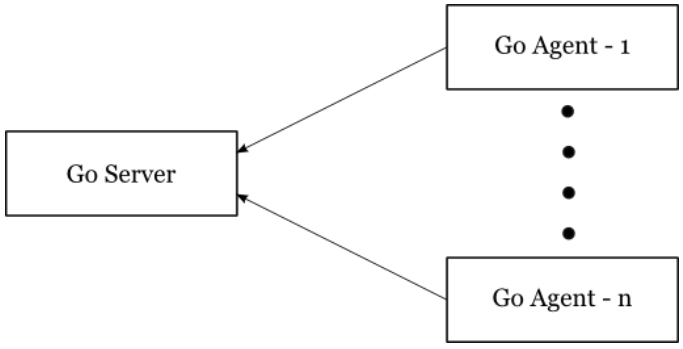


Figura 26: Estructura bàsica de GoCD

Un cop el servidor està funcionant, aquest comença a escoltar peticions HTPP al port TCP 8153 i peticions HTTPS al port TCP 8154. L'agent es pot instal·lar en la mateixa màquina on corre el servidor o bé en una màquina externa. L'agent només necessitarà poder connectar als ports anteriorment citats per a comunicar-se amb el servidor.

En GoCD el servidor no monitoritza ni envia activament (*push*) les tasques de *build* als agents, sinó que són els propis agents qui cada cert temps es van connectant al servidor per a consultar i demanar si hi ha *jobs* disponibles per processar (*pull*). Això evita que els agents estiguin contínuament escoltant en un port connexions entrants. El servidor avaluarà el resultat del *build* executat a l'agent, i en funció del resultat i el *workflow* configurat decidirà l'estat de *l'stage* i coordinarà els següents passos a realitzar.

3.5.2.2 Organització del treball

En GoCD la *pipeline* està formada per un conjunt *d'steps* que s'executen en ordre seqüencial. En la definició de la *pipeline* estan descrits tots els passos, accions i tasques a realitzar mitjançant la configuració dels corresponents *steps*, *jobs* i *tasks*. En la següent figura es pot visualitzar un exemple de com està organitzada internament una *pipeline*.

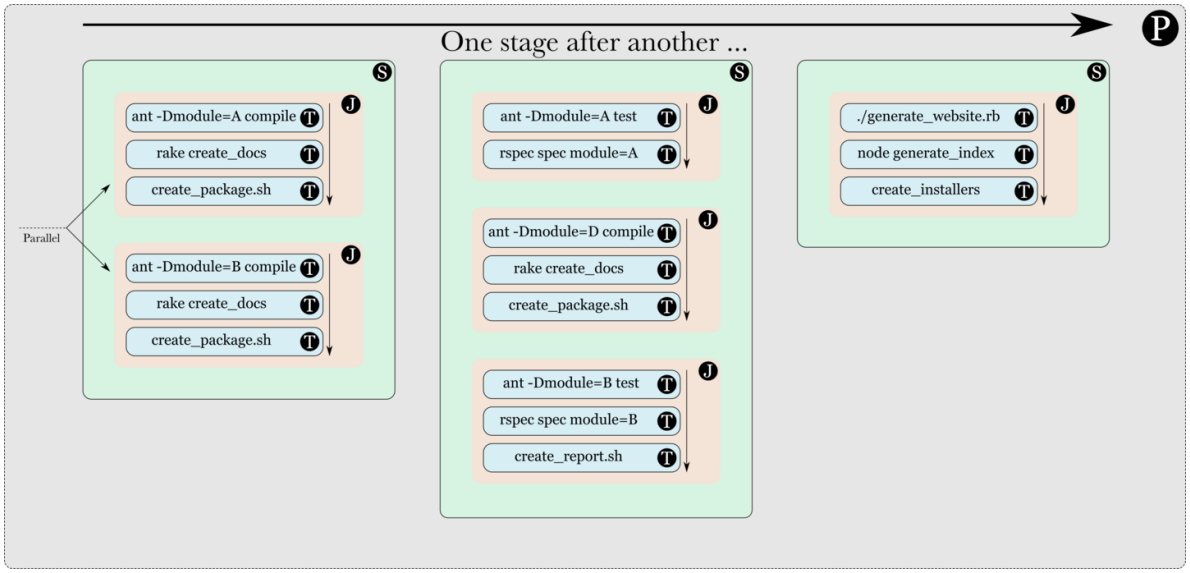


Figura 27: Exemple composició d'una pipeline en GoCD

En la figura els element amb la lletra "S" corresponen a *stages*, els de la lletra "J" a *jobs* i finalment els de la lletra "T" a tasques. L'element "P" és la *pipeline*, que engloba tots aquests subelements.

Dins de cada *step* es defineixen els jobs corresponents. En GoCD els *jobs* són independents els uns dels altres i poden ser executats en paral·lel. Si un *job* falla, *l'stage* sencer es considera com a erroni. La resta de *jobs* de *l'stage* continuaran fins a finalitzar la seva execució.

Els *jobs* són un conjunt de tasques que s'executen en l'ordre configurat. Si es dona algun error en alguna tasca, les següents no s'executaran i es considerarà el *job* com a erroni. Cada tasca del *job* s'executa en un programa independent de tal forma que no es poden compartir variables d'entorn entre elles. En canvi, qualsevol modificació en el sistema de fitxers si que serà visible per a les posteriors tasques.

Materials.

GoCD introdueix el concepte de material per a fer referència a elements externs que causen que s'executi una *pipeline*. Per exemple, els repositoris de codi font externs com GIT o SVN són considerats com a materials. També es pot aplicar aquesta definició a esdeveniments programats que engeguen la *pipeline*. El servidor anirà contínuament consultant els materials que té registrats, i quan detecti un canvi en algun d'ells, disparará el *trigger* que executarà la *pipeline*.

Les *pipelines* es poden combinar i poden interactuar de moltes formes. Per exemple un *stage* d'una *pipeline* pot ser emprat com a material d'una altra *pipeline*. D'aquesta forma es poden configurar noves relacions i dependències entre les *pipelines*. GoCD també implementa els conceptes de *fan-out* i *fan-in* de *pipelines*. Amb *fan-out* es permet engegar múltiples *pipelines* com a resultat de la finalització de *l'stage* d'una *pipeline*. Amb *fan-in* es necessiten els resultats de diverses *pipelines* per a causar l'inici de la següent *pipeline*. Tots aquests lligams i dependències es resumeixen en una vista anomenada *Value Stream Map*. La següent figura mostra un exemple.

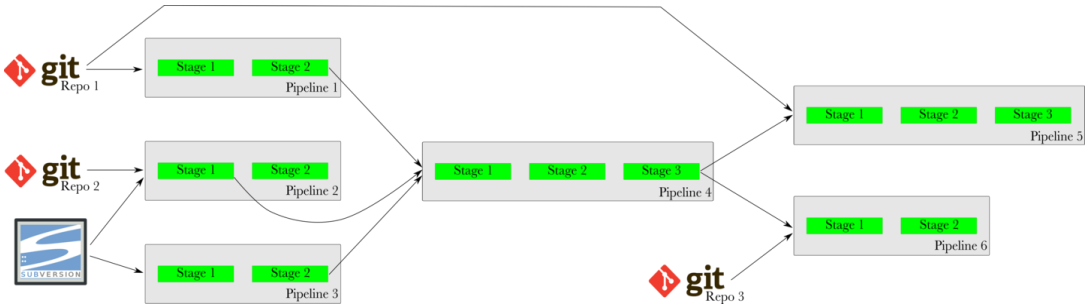


Figura 28: Value Stream Map en GoCD

Generalment, GoCD mostra en la seva interfície web d'administració una vista (*Dashboard*) completa sobre l'activitat de les diferent *pipelines* configurades al servidor.

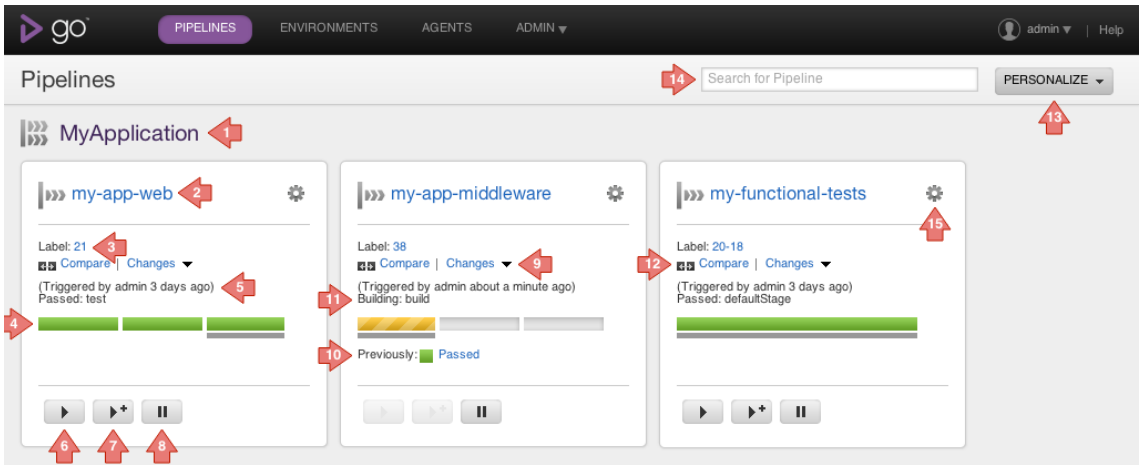


Figura 29: GoCD Dashboard

3.5.2.3 Comparació de builds

GoCD proporciona una interesant funcionalitat que permet fer comparacions entre dos *builds* donats. Es podrà veure que ha passat entre aquestes dues instàncies concretes de l'aplicació en termes de canvis al codi font i canvis en les definicions de les *pipelines* emprades. Es visualitzarà també un vincle cap a l'eina d'enregistrament de canvis

associats a cada *buid*. També es podrà accedir al navegador de l'històric dels canvis en el repositori i en la *pipeline* facilitant qualsevol cerca d'informació històrica.

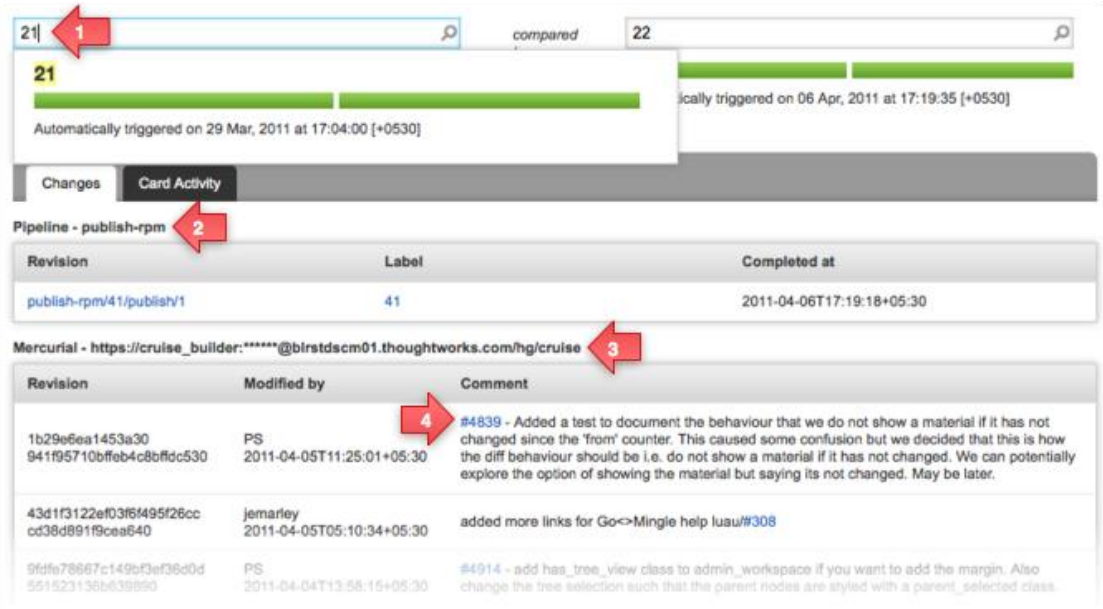


Figura 30: GoCD utilitat comparació de builds

3.5.3 Plugins

GoCD disposa d'un gran catàleg de complements amb múltiples funcionalitats que es poden afegir a la versió estàndard del programa. En la seva web [30] estan classificats per la seva funcionalitat, com per exemple gestors d'autenticació, gestió de repositoris, de notificacions, de tasques, etc. Molts estan disponibles gratuïtament però d'altres tenen un cost addicional.

3.5.4 Suport multiplataforma

GoCD està composat per dos mòduls que s'han d'instal·lar de forma independent, el servidor i l'agent. Es proporcionen instal·ladors optimitzats per als diferents sistemes operatius i també un instal·lador genèric en format .ZIP. Els sistemes operatius suportats pel servidor i per l'agent són:

- Windows - Windows Server 2003, Windows Server 2008 i Windows 7.
- Mac OSX - 10.7 (Lion) i superiors.
- Linux - Debian 6.0 (squeeze) i superiors.
- Linux - CentOS/RedHat - CentOS/RedHat version 5.0 i superiors.

EL servidor necessita com a mínim 1 GB de RAM (2 GB recomanats), una CPU de 2 cores a 2GHz i 1GB d'espai buit de disc. Els agents necessiten uns 256MB de RAM i 1 CPU a 2GHz mínim. Caldrà dimensionar correctament les capacitats dels servidors on corren els agents ja que són els que consumiran la majoria de recursos de la plataforma GoCD. Ambdós components han de fer servir la mateixa versió de Java JRE v8.

GoCD no suporta de forma nativa ni incorpora *builders* específics, com passa en altres productes, per a llenguatges de programació. Com que en última instància les tasques

executen bàsicament comandes, és responsabilitat de l'usuari proporcionar les comandes necessàries per a treballar amb les plataformes, eines i llenguatges de programació que vulgui integrar. Caldrà que estiguin degudament instal·lades en la plataforma. GoCD proporciona tota la infraestructura i s'encarrega de tota la lògica de gestió d'alt nivell per a poder realitzar un procés d'Integració Continua complet.

3.5.5 Política de llicències

GoCD és un producte *OpenSource* i el seu codi font està disponible per a la comunitat en el seu compte de GITHUB. Tant els servidors com els agents es poden descarregar i instal·lar sense cap limitació gratuïtament.

D'altra banda, la companyia propietària de GoCD Thoughtworks, si que proporciona un servei de suport comercial amb un cost anual. En comptes de fer pagar pel producte, es fa pagar per serveis i funcionalitats addicionals com:

- Serveis de suport sense límits mensuals, accés a l'equip de desenvolupadors, notificacions de *bugs* i actualitzacions de seguretat.
- Complements amb funcionalitats avançades com integració del gestor de BBDD PostGreSQL o *Business Continuity* per a garantir el servei amb redundància de servidors i altres elements de la plataforma.
- *Plugins* corporatius com connectors a la plataforma d'Amazon EC2 o gestors d'autenticació corporatius com LDAP.

El cost de tots aquests serveis va un funció del nombre de *pipelines* configurades al servidor. La quota anual es resumeix en la següent taula:

NUMBER OF PIPELINES	INCLUDES	PRICING, PER YEAR, USD
1-50	Support only	$5,000
51-250	Support + Add-ons & Plugins	$15,000
251-500	Support + Add-ons & Plugins	$25,000
500+	Support + Add-ons & Plugins	Let's Talk

Figura 31: Taula resum cost servei de suport de GoCD

3.6 Resum i comparativa de les eines

Amb l'anàlisi de les eines d'Integració Continua exposades, es pot veure que hi ha característiques i conceptes compartits en elles.

El concepte de *pipeline* es àmpliament emprat per aquestes eines. És la base per a configurar tots els passos i accions que ha de fer el servidor de IC des de que detecta un canvi en el repositori de codi font fins que es publica una aplicació en algun dels entorns de l'organització. La majoria d'eines disposen d'editors visuals que mostren l'estructura de les *pipelines* i quasi tots ells organitzen la feina a fer fent servir els mateixos conceptes de: stage, job i task. Després cada producte tracta el mode d'executar-los i les dependències entre ells de forma particular.

A nivell d'arquitectura tots es basen en un model servidor – agent. L'aplicació servidor controla la lògica i porta el control de l'execució dels *workflows* monitoritzant en tot moment la resta de components externs com els repositoris i distribuint la feina entre els diferents agents. També són funcions importants del servidor enregistrar i notificar de forma activa o passiva tot el que passa en la plataforma de IC.

En la part dels agents és on es produeixen més diferencies entre les eines. La majoria poden executar l'agent en la mateixa màquina on corre l'aplicació però tan bon punt és necessària una mica més de capacitat, els agents s'han d'executar en màquines remotes independents. La diferencia entre els productes de IC radica en com gestionen els agents aspectes com:

- Mode d'execució dels agents remots i en quin grau de concurrència i paral·lelisme ho poden fer (i a quin preu).
- Tipus de tecnologies o plataformes suportades: instàncies d'imatges de contenidors Docker, imatges de màquines virtuals, instal·lacions normals en servidors físics o virtuals, etc.
- Si estan fets i dissenyats principalment per a executar-se al núvol d'Amazon i/o Microsoft Azure o bé simplement ho suporten com a funcionalitat addicional.

Un altre aspecte a destacar és que tots ells, excepte CircleCI que no ho indica, estan desenvolupats en Java i necessiten una màquina virtual Java per a ser executats. La majoria ho fan dins un servidor d'aplicacions Java tipus Tomcat.

A nivell de costos tenim productes OpenSource completament operatius sense cap cost, com Jenkins i GoCD (excepte el suport) i d'altres de propietàries que, tot i oferir versions de proves o limitades en capacitat, cobren un cost per la llicència d'ús. El preu de les llicències de Bamboo, TeamCity i CircleCI van en funció de magnituds de capacitat de procés com el nombre d'agents (locals o remots), nombre de *builds* permesos, el nombre d'usuaris o el nombre de contenidors que s'executen. Cada producte ha definit una estratègia comercial pròpia i diferenciada de la resta dels competidors. Addicionalment també inclouen serveis de suport amb les llicències, ja sigui inclòs o per separat en el cost de la llicència.

En la següent taula s'ha intentat resumir i sintetitzar les principals característiques dels 5 productes com tipus de plataformes, modes d'execució, *builders* i repositoris suportats i el tipus de cost/llicència de cadascun.

Producte / Característica	Jenkins	Bamboo	TeamCity	CircleCI	GoCD
Plataforma	Multiplataforma	On-premises	On-premises	Hosted	On-premises
Mode execució Servidor	Dins un contenidor Java. Com a servei. Dins contenidor Docker.	Dins servidor contenidor Tomcat.	Dins servidor contenidor Tomcat o compatibles	En màquines virtuals al cloud	Aplicació servei instal·lada al servidor amb servidor web propi.
Mode Execució Agents	En un procés o servei al servidor.	Aplicació servei instal·lada en els servidors. Amazon EC2	Aplicació servei instal·lada en els servidors Cloud: Amazon EC2, Azure i VMWare vSphere.	En màquines virtuals al cloud	Aplicació servei instal·lada en els servidors
OS Suportats	Windows MacOS Linux Ubuntu/Debian	Windows MacOS Linux Solaris	Windows MacOs Windows Linux (Ubuntu, Debian, RedHat, SUSE)	Amazon WebServices AWS.	Windows MacOsX Linux – Debian, RedHat, CentOS
Builders	MSBuild, Maven, NAnt, Ant, Shell and batch script, Python entre d'altres	Apache Ant, Maven, make, MSBuild, bash i powerShell	Ant, Maven, IntelliJ IDEA, MSBuild, VS, Nant, etc	A partir d'imatges Docker Java, Php, Node.js, Python i Ruby.	Lina de comandes
CVS suportats	CVS, Git, Mercurial Subversion, TFS, GitHub, i molts més amb plugins addicionals.	Git, GitHub, Mercurial, subversion, CVS i Bitbucket	Git, Subversion, TFS, Mercurial,CVS, SourceGear i més.	GitHub i Bitbucket.	Git i GitHub
Tipus llicència	OpenSource MIT	Propietària	Propietària	Propietària	Open Source
Cost	Gratuït	Basic: 10$ (limitat 10 jobs, angets locals) + 10$ suport any. Professional: 880$ 1 sol cop per a 1 agent remot - 2420$ 1 sol cop per a 2-5 agents remots. Màxim 250 agents remots. Inclòs 1 any manteniment. Renovació manteniment: la meitat del cost de nova llicència. Suport inclòs.	Professional: gratuïta limitada 20 plans, 3 agents, ~~299€~~ build addicional any. Enterprise: plans il·limitats, 3 agents 1999€ - 100 agents ~~21999€~~. Renovacions. Meitat preu cost nova llicència. 50% descompte per a startups. Gratuïta per a projectes OpenSource amb límits.	Versió Gratuïta amb limitacions sols per Linux. En funció del OS: Linux: 1 contenidor gratis, 1500 minuts/build/mes. Contenidor addicional 50$/mes MacOs: en funció concurrència, builds i minuts 39$, 129$, 249$, 449$ mes Darrera Firewall: 35$ usuari, +100 usuari preu a mida.	Gratuït servidor i agent. Suport en funció pipelines: 1-50 5000$ any sols suport. Suport + complements: 51-250 15000$ any 251-500 25000$ any.
Versió de proves		Si, 30 dies	Si, Versió Enterprise 60 dies	Si, per MacOs 2 setmanes.	

3.7 Justificació del producte triat

Es complicat definir un criteri correcte i vàlid per a poder decidir quina és la millor eina de les 5 analitzades. Caldrà considerar diverses característiques a la vegada per a identificar quina solució s'adapta millor a les nostres necessitats. És obvi que no ens podem fixar només en el cost del producte sinó que hi ha més factors a tenir en compte alhora de triar una solució. Per suposat el cost que es pugui permetre cada organització serà diferent. Aquest cost limitarà i farà descartar alguna de les opcions. Tot i així cal fer també altres tipus de preguntes relatives a la tecnologia involucrada com les següents:

- Quin tipus de tecnologia fa servir el producte?
- És una solució basada en instal·lacions a servidors *on-premise* o fa servir serveis al *cloud*?
- Quin tipus d'arquitectura i tecnologia fa servir actualment l'organització? Encaixa?
- L'organització disposa d'aquests recursos i de prou coneixements per administrar-los?

Altres qüestions relatives a la infraestructura existent i a com desenvolupa l'organització les aplicacions haurien també de ser considerades. Per exemple:

- En quins llenguatges de programació desenvolupa l'organització?. L'eina té *builders* per a treballar amb ells? Són de fàcil ús?
- És compatible el producte amb els repositoris de codi font que es fan servir actualment?
- Es necessita i es valorarà un suport de qualitat de l'eina?
- Els productes es poden integrar amb altres aplicacions? Es disposa actualment de software del mateix fabricant o que es pugui integrar? Aquesta integració suposarà un valor afegit per a l'organització?

Finalment, un cop es tenen valorades totes aquestes qüestions, segurament es descartarà algun producte per raons tècniques. A continuació, caldrà afegir el cost del producte i valorar quina és la millor opció.

En el nostre cas per a organitzacions que disposen d'una infraestructura clàssica de servidors físics o virtualitzats en un DataCenter totes les opcions, excepte CircleCI, poden ser perfectament viables. Tots 4 poden ser executats en Windows i Linux. Aquest fet no ens limitarà alhora de triar l'opció ja que la majoria d'organitzacions fan servir aquests dos sistemes operatius. Si l'organització busca una solució que estigui totalment hostatjada fora de la infraestructura de l'organització, o no es disposen de recursos locals per a la plataforma de IC, CircleCI és una bona opció.

Si ens fixem en quin tipus de recursos fan servir els agents per a executar-se, hi ha també dos escenaris ben diferenciats:

- Productes que poden escalar i executar agents de forma habitual en entorns remots basat en el cloud. En aquest grup estarien Bamboo, Team City i CircleCI.
- Productes que sols poden fer-ho en màquines locals o virtuals: Jenkins i GoCD.

En el primer grup, Bamboo està més orientat a fer-se servir en instal·lacions en servidors *on-premise* més que al núvol. Tot i així és força interessant per a un futur creixement el suport d'executar agents remots en Amazon. Es una solució força equilibrada en aquest aspecte. TeamCity està més orientat i suporta 3 tipus de plataformes remotes, sent una bona opció per a empreses que ja disposen d'aquests serveis en la seva infraestructura i dels coneixements necessaris per a la seva correcta administració. CircleCI es pot descartar si es vol tenir qualsevol part de la plataforma en servidors *on-premise*. Tot i que el fabricant diu que es poden fer solucions a mida darrera el Firewall corporatiu, continuen basant-se en solucions al *cloud* 100%.

En el segon grup, Jenkins i GoCD, executen els agents de forma similar. Tots dos són gratuïts. Així que en aquest cas altres aspectes com *builders* i repositoris de codi font suportats i el servei de suport són els que haurien de determinar l'elecció. Jenkins suporta un ampli ventall de *builders* i llenguatges de programació nativament amb *plugins*, mentre que GoCD no incorpora de forma nativa *builders* per a suportar llenguatges de programació. GoCD solament executa comandes i delega en l'usuari la instal·lació de les eines i comandes necessàries per a per fer els *builds*. En aquest aspecte Jenkins seria més fàcil de fer servir i també suporta mes repositoris de codi font. GoCD cobra pel servei de suport, mentre que amb Jenkins es disposa del suport general de la comunitat.

Si ens centrem una mica més en l'escenari concret que ens trobarem en aquest PFC, amb una organització amb les següents característiques:

- Infraestructura servidors *on-premise.*
- No disposa actualment de serveis en *cloud*, ni dels coneixement per a administrar-los.
- Els desenvolupadors programen fent servir la plataforma .NET de Microsoft.
- Fa servir Git, VisualStudio, Jira i Confluence, entre d'altres, com a principals eines de suport al desenvolupament.

Es descarta CircleCi perquè és un producte basat en tecnologies al *cloud* i GoCD per no portar suport natiu per a *builders* de Microsoft tipus MSBuild i VisualStudio.

Entre Jenkins, Bamboo i TeamCity, es proposa triar Bamboo per que encaixa perfectament en la infraestructura *on-premise* de l'organització, suporta sense problemes *builders* per .NET i Microsoft, pot treballar nativament amb repositoris GIT i suporta agents remots al cloud d'Amazon. A més a més, es traurà profit de l'ús actual de Jira per a integrar-ho en el *workflow* de Bamboo i així disposar de vincles directes entre els *builds* i les tasques Jira relaciones. També en el futur es podria fer servir Jira per a controlar i llençar releases automàticament a Bamboo cada cop que es fa el release d'un projecte en Jira. Per un altre costat es valora un suport professional per part d'una empresa com Atlassian que ens pugui ajudar en el procés d'instal·lació i d'operacions del producte, en detriment d'opcions opensource com Jenkins. El cost de la llicència d'1 agent remot (880$/any), com es veurà en propers apartats, serà suficient per a cobrir les necessitats de l'organització i ens donarà accés al servei de suport.

4 Implementació d'un Sistema d'Integració Continua

En aquest capítol es descriurà un exemple del procés d'implantació d'un sistema de Integració Continua en una empresa tipus. Partint d'un escenari definit es procedirà a analitzar a fons tots els aspectes relacionats directa i indirectament amb la IC. S'identificaran els problemes i s'avaluaran els canvis necessaris per a implantar un sistema de IC.

S'aplicarà un model de gestió de la IC que permetrà, fent servir com a base l'eina de Integració Continua Bamboo, un correcte flux de treball que:

- Permeti una correcta organització de tots els projectes dins d'un repositori de codi font de forma estructurada.
- Permeti gestionar fàcilment la integració del codi de forma automàtica executant totes les validacions necessàries.
- Permeti estandarditzar el procés de publicació de les aplicacions.
- Permeti fàcilment fer *tests* i integracions en els diferents entorns de validació i proves.
- Permeti enregistrar i accedir fàcilment al històric de totes les operacions realitzades en la plataforma.

4.1 Descripció i anàlisis de l'entorn inicial

L'empresa JASMobile SL és una empresa dedicada al desenvolupament d'aplicacions Web per a dispositius mòbils que opera en el mercat des de l'any 2005. El seus principals clients són altres empreses del sector que volen que JASMobile els desenvolupi portals WAP/WEB per a dispositius mòbils. De forma habitual es consumiran APIs externes d'altres proveïdors de continguts.

L'empresa està formada per uns 45 empleats, ubicats majoritàriament en una seu central HQ a Barcelona i dues oficines remotes situades a Lisboa i Toulouse. En les oficines remotes treballa tant personal comercial com desenvolupadors locals. La seu central disposa d'un petit *DataCenter* on es troben els servidor centrals de l'organització (*Backup*, Comptabilitat, BD internes, servidor Correu, ActiveDirectory, Git repo, etc) i de l'equip de desenvolupament. Els servidors productius on es publiquen les aplicacions pels clients finals es troben hostatjades externament en un proveïdor local que ofereix servei de *DataCenter* d'alta disponibilitat connectat a altres *Backbones* de proveïdors d'internet (ISPs)

L'accés per part de les oficines remotes a la xarxa de la seu central i al *DataCenter* es fa mitjançant una connexió VPN permanent entre els diferents *Firewalls* de les seus. A la vegada tots els empleats poden accedir des dels seus PCs portàtils o dispositius mòbils a la xarxa interna de les seus a partir del client Software VPN facilitat pel fabricant del Firewall (Fortinet).

4.1.1 Metodologia de treball i recursos dels desenvolupadors

L'equip de desenvolupadors està composat per 20 tècnics aproximadament. Els programadors principalment fan servir la plataforma .NET per al desenvolupament de la majoria d'aplicacions. Depenent de la seu i dels equips, es fan servir les versions del IDE de Visual Studio 2013 i la 2015. Aquesta última versió s'ha començat a adquirir recentment pels nous equips.

Una petita part dels projectes han de fer servir llibreries de tercers desenvolupades en Java. Aquestes llibreries acostumen a estar empaquetades normalment en fitxers **.jar** o bé en contenidors d'aplicacions J2EE **.war**. Per aquest motiu alguns programadors també han de fer petits desenvolupaments en Java fent servir el IDE Eclipse.

Es disposa d'un servidor Linux Centos en el DataCenter de la seu central amb un repositori GIT on es guarden les diferents solucions dels projectes de desenvolupament. Cada programador, fent servir el client Git per a Windows publica els seus canvis al repositori GIT. Cada equip de treball té un *release manager* que és l'encarregat de portar el control de les diferents versions del codi i és qui realitza les publicacions. Aquest procés es fa de forma manual: s'obté del repositori l'última versió correcta, es fa el *build* local i es copia per SCP (màquines Linux) , RPD o FTP als servidors Windows de producció i de test.

4.2 Anàlisis d'entorns

En aquest context s'entén com a **entorn** l'agrupació dels diferents recursos de xarxa, servidors, configuracions i aplicacions que formen un context diferenciat, isolat i únic on s'executen les aplicacions. Normalment cada entorn està especialitzat i dedicat a una funció específica i acostumen a estar isolats els uns dels altres. Aquest últim aspecte és molt important perquè els canvis produïts o les dades emprades puntualment per una aplicació que per exemple està en fase de desenvolupament no ha d'interferir sota cap circumstància en els serveis i les dades que s'executen en producció.

Dins de la metodologia *Agile*, els diferents entorns facilitaran el treball dels equips amb la creació dels "*Development sandboxes*" corresponents. Bàsicament un *sandbox* correspon a l'entorn i als recursos necessaris que cada desenvolupador i equip de treball necessiten per a poder fer la seva feina. En la següent figura es pot veure una proposta de diferents entorns *sandbox* que pot encaixar amb la proposta d'entorns que es plantejarà en els propers apartats per a l'organització.

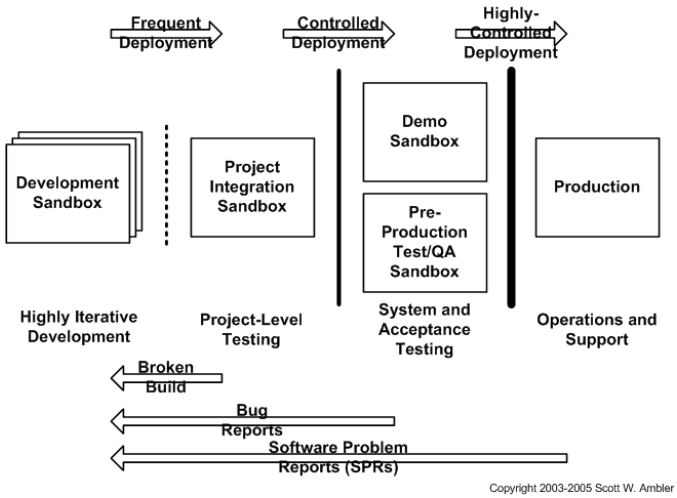


Figura 32: Conjunt de *sandboxes* – entorns en un entorn de treball Àgil

Des de la perspectiva de la IC, els diferents entorns han de formar part del procés d'integració. Així els desenvolupadors podran fer-los servir de forma transparent i automatitzada a través de les funcionalitats del servidor de IC.

Amb la finalitat d'encaixar millor amb la pràctica i adopció de la IC es suggereix la creació i revisió dels entorns que es descriuen tot seguit.

4.2.1 Entorn de desenvolupament local

És l'entorn en el que cada desenvolupador treballa i valida les funcionalitats del seu codi. Normalment és la pròpia màquina del programador. Amb l'objectiu de minimitzar els riscos d'incompatibilitat o diferència d'entorns de desenvolupament, per a cada estació de treball es revisarà:

- L´ús d'una única versió del IDE Visual Studio 2015. Caldrà actualitzar de forma progressiva les llicències dels equips més antics.
- Estandardització de sistemes operatius, actualitzant les llicències a la versió Windows 10 Pro en totes les estacions de desenvolupament.
- Versió .NET Framework. Comprovar que tots els equips tenen instal·lades fins a la versió .NET 4.6 (incorporada amb VS 2015) [31].
- Client GIT i *plugin* GitFlow tinguin la mateixa versió en tots els equips.
- Client d'accés al servidor de Bases de Dades SQLServer: l'última versió és Management Studio v17.
- Servidor de Bases de Dades local SQLServer (Versió Developer o Express Edition, sense cost addicional de llicències)

Amb aquestes actuacions s'intentarà que tots els desenvolupadors treballin amb les mateixes eines i entorns amb l'objectiu que el codi es pugui integrar en un entorn el més homogeni possible.

4.2.2 Entorn d'integració o test

Aquest entorn estarà dins els servidors de la seu central de Barcelona ja que la majoria de l'equip de programadors està a la seu central. En aquest entorn es publicaran aplicacions que requereixen per a ser validades accés des d'altres equips o servidors externs o bé que no es poden validar executant l'aplicació en local.

En aquest entorn es proposa disposar de la següent infraestructura de servidors per a les proves d'integració:

- 1 servidor Windows Server versió 2012 R2 amb servidor d'aplicacions Internet Information Server (IIS) versió 8 instal·lat per a publicar les aplicacions FrontEnd de tipus: Web Applications, serveis web, MVCs, WebApis, etc.
- 1 servidor Windows Server versió 2012 R2 per a executar aplicacions de tipus servei o stand-alone (aplicacions de BackEnd) que no necessitin exposar cap tipus de funcionalitat a través del servidor web d'aplicacions IIS.
- 1 servidor Windows Server 2012 R2 amb una instància de SQL Server 2012 que es farà servir com a Base de Dades d'integració. Es crearan les Bases de Dades locals o instàncies (lliures de llicències extres) necessàries per a córrer les proves d'integracions.
- 1 servidor amb una distribució Linux Centos 7 amb un servidor Apache Tomcat 8 per a executar aplicacions i llibreries de tercers en contenidors Java Servlets.

Aquests servidors estaran dins de la xarxa interna local de l'organització i seran visibles des de la seu central de l'organització i des de les seus remotes a través de l'accés per VPN corporatiu. Es farà servir adreçament per DNS intern exclusivament.

Les característiques d'aquests servidors seran el més semblants en termes de sistemes operatius, versions de servidors d'aplicacions i *frameworks* instal·lats als servidors de producció. D'aquesta forma, tot i que no tinguin les mateixes capacitats de computació i de connexió que els servidors del DataCenter remot, s'intentarà treballar en un entorn

més el més similar possible. L'objectiu és garantir que no es produiran problemes d'incompatibilitat entre aquest entorn i el de producció en aspectes de dependències de software , llibreries i recursos del sistema operatiu.

Eventualment des del Firewall de la seu central, quan sigui necessari l'accés extern a algun recurs o aplicació que s'executi en aquest servidors, es podran exposar alguns serveis de forma pública habilitant les regles de Firewall i NAT corresponents. També es pot fer ús d'un servidor "*Reverse Proxy*" per a oferir accés indirecte a aquests recursos interns sense ser exposats directament a internet.

En el servidor de IC Bamboo, estaran configurats en els scripts de *deployment* les regles necessàries per a instal·lar i gestionar les aplicacions en aquests servidors. Com es veurà posteriorment, cada projecte que s'integra amb Bamboo haurà de tenir un pla de compilació específic per a l'entorn de test/integració.

4.2.3 Entorn de preproducció (Pre-Release)

La idea d'aquest entorn és simular un entorn en condicions productives però sense tenir el producte final desplegat en producció ni fer ús dels seus recursos. L'aplicació podrà ser validada pels clients finals o altres departaments de l'organització de la mateixa forma com si fos accedida des de l'exterior en condicions de producció. És en aquest entorn on es realitzaran totes les *demos* i els *User Acceptance Test* (UAT) de les aplicacions que ho requereixin.

L'entorn de pre-producció es trobarà en el mateix servei de hosting de DataCenter extern que producció i serà accessible fent servir adreçament per IPs i DNS públics o a través de connexions VPNs dedicades amb els clients que ho requereixin.

Els servidors de pre-producció seran rèpliques exactes dels servidors de producció. L'única diferencia és que estaran fora del tràfic dels dominis de producció i de les Bases de Dades productives.

De forma similar que a l'entorn d'integració, es disposarà d'un servidor de cada tipus:

- 1 servidor Windows Server versió 2012 R2 amb servidor d'aplicacions IIS 8.
- 1 servidor Windows Server versió 2012 R2 per a les aplicacions de BackEnd i serveis.
- 1 servidor Windows Server 2012 R2 amb una instància de SQL Server 2012 que es farà servir com a Base de Dades de pre-release.
- 1 servidor Linux Centos 7 amb Apache Tomcat 8 per a aplicacions Java/J2EE.

El servidor de IC tindrà accés a aquests servidors per a dur a terme el *deployment* corresponent.

4.2.4 Entorn de producció

L'entorn de producció és on s'instal·laran i executaran totes les aplicacions i serveis finals que es lliuren als clients. Tota la infraestructura de la plataforma de producció estarà hostatjada únicament en el proveïdor de DataCenter extern. Aquest entorn estarà internament comunicat amb les seus de l'organització a través de les VPNs.

No és objectiu d'aquest PFC aprofundir en quina és la millor arquitectura per a estructurar la plataforma dels servidors de producció. Dependrà dels requisits de les aplicacions dissenyades, dels volums de transit per part dels usuaris i clients, del pics de tràfic que es vulguin absorbir, etc. Es pot triar per una infraestructura de servidors

físics del proveïdor del *hosting*, optar per una estratègia de virtualització de màquines en aquests servidors, etc. Tot dependrà de les necessitats de l'organització.

Tot i així, la infraestructura de servidors de producció ha de complir uns requisits mínims:

- Els servidors han de córrer el mateixos sistemes operatius que es fan servir als altres entorns de pre-producció i d'integració.
- Han de tenir instal·lats el mateix *framework*, llibreries i altres components de software necessaris per a funcionar de la mateixa forma que servidors dels altres entorns.
- A nivell de connectivitat han de ser accessibles des de l'exterior (internet públic) només els serveis estrictament necessaris (serveis web, APIs, Front-Ends, etc) que siguin part del producte o servei. Han d'estar protegits degudament pel Firewall corporatiu del DataCenter.
- Tot els accessos remots als servidors estaran limitats als administradors de sistemes exclusivament. Cap desenvolupador tindrà accés ni podrà manipular cap recurs sense control dels administradors.
- Els servidors de producció seran configurats en una xarxa privada interna per a poder comunicar-se entre ells. A la vegada s'ha de permetre l'accés intern des de:
    - El servidor d'Integració Continua. Aquest haurà de poder accedir als servidors de producció per a fer els *deploys* necessaris.
    - El servidor de monitorització, per a controlar l'estat del servidor i dels seus recursos.
    - Els servidors de Base de Dades, DNS, correu, Active Directory de l'organització, Servidors de Backup, de logs, etc.
    - Els serveis interns de l'organització (WebServices, sistemes de fitxers en xarxa NAS, etc).

4.2.5 Access als entorns

Un dels objectius de la Integració Continua és la capacitat de validar i posar noves versions de les aplicacions funcionals en pre-producció i producció de forma ràpida i també segura. Per a garantir encara més que el software desenvolupat i testejat funciona com s'espera s'ha de fer un control estricte de l'accés als servidors. No s'ha de poder modificar en calent cap fitxer de configuració, ni el sistema de fitxers, configuracions dels servidors ni cap altre recurs. Qualsevol modificació no controlada pot provocar incoherències entre la versió que es troba al repositori de codi font i la que s'està validant. Encara serà més greu si la versió modificada és la que s'està executant en producció. L'aplicació modificada, per exemple, podria estar fent servir fitxers erronis de configuracions, o una Base de Dades incorrecta. És molt important que el codi que es té (i tots els seus recursos i configuracions) sigui el correcte.

L'accés dels desenvolupadors a aquests servidors, mitjançant inici de sessió remota amb un client de protocol RDP en els servidors Windows i per SSH al servidors Linux, estarà permès només a l'equip administradors de sistemes i DBAs. Només es donarà accés remot al sistema de fitxers per a fer consultes en mode de lectura a les carpetes de *logs* de les aplicacions. Sense aquests tipus de controls bàsics, no es pot garantir una Integració Continua de forma correcta.

4.3 Anàlisis de l'estructura del codi font

L'estructura del codi font és fonamental per a permetre la correcta automatització de molts dels nous processos que s'incorporaran a l'organització amb la implantació de la IC. El control del codi font i la seva gestió és el pilar fonamental en la nova plataforma que s´introduirà. Cal fer una revisió general de com es gestionen a nivell global els projectes dins dels repositoris de codi font i com estan estructurats internament cadascun d'ells. Dins de l'organització es llistaran tots els projectes informàtics en ús i es procedirà a revisar diversos punts que es descriuen en els següents apartats.

4.3.1 Estandardització de la nomenclatura dels projectes

Cal definir i fer servir una nomenclatura comuna per a tots els desenvolupadors, que indiqui clarament el tipus de projecte, el nom, el client, la funcionalitat, el país o àrea als que fan referència el projecte. Tenint en compte els tipus de projectes i clients amb què treballa l'organització, es proposa la següent nomenclatura:

[Àrea_O_País].[TipusProjecte].[SubTipusProjecte].[NOM client/api/llibreria]

Per exemple uns possibles valors per a cadascun dels camps definits entre claudàtors podrien ser els següents:

- **[Area_o_Pais]**: Poden ser països o regions geogràfiques com Spain, Portugal, EMEA, LATAM, etc o bé interns com "Corporate".
- **[TipusProjecte]**: Possibles valors poden ser per exemple:
  - Mobile.
  - Web.
  - EntryPoint.
  - Tool.
  - Library.
  - Platform.
  - Service.
  - Connectivity.
  - ...
- [SubTipusProjecte]: Com per exemple:
  - FrontEnd.
  - OperatorEntryPoint.
  - GUI.
  - PartnerEntryPoint.
  - Gateway.
  - BusinessLogic.
  - DataModel.
  - ...

Per exemple un *Gateway* és un subtipus de projecte de tipus "*Connectivity*".

El camp nom ha de ser curt i representatiu de què és i per a qui és el projecte. Uns possibles exemples de projectes ficticis podrien ser:

- Corporate.Library.BusinessLogic.Client
- Spain.Mobile.FrontEnd.Badoo
- Corporate.Connectivity.Gateway.PayPal
- Italy.Platform.DataModel

Amb cada projecte correctament categoritzat i amb una funcionalitat fàcilment deduïble a partir del seu nom, es podrà fer més endavant una configuració i organització dels

plans d'integració en l'eina de IC més entenedora. Com més estructurat, millor facilitat d'ús.

4.3.2 Normalització dels projectes

A fi d'identificar i posar el nom correcte a cada projecte és necessari que cada solució només contingui la pròpia implementació de la funcionalitat indicada. Per exemple un projecte de subtipus *Gateway* només ha d'implementar el protocol de connexió amb el servei a la que es connecti i res més. En altres paraules, caldrà separar i dividir aquelles solucions que aglutinen diferents funcionalitats en d'altres de més petites. L'objectiu és integrar projectes el més senzills i atòmics possibles per a facilitar la gestió de tots els processos d'Integració al servidor de IC.

Prenent, per exemple, com a referència un projecte .NET desenvolupat amb l'IDE VisualStudio, caldrà normalitzar l'estructura de la solució de la següent forma:

- La solució ha d'incloure 1 projecte amb el mateix nom que la solució, que serà el que es configurarà en el servidor de IC. Aquest projecte conté la implementació de la funcionalitat del projecte: una llibreria DLL, un entryPoint SOAP, un WebAPI, una servei Windows, una aplicació *stand-alone*, etc. Això permetrà una millor configuració del pla d'integració al servidor de IC.
- La solució no ha de contenir altres projectes de tipus llibreria DLL que implementin una catàleg de funcionalitats que es facin servir també en altres projectes o que estiguin duplicats en altres projectes. Cal separar i refactoritzar el codi d'aquest projecte en una nova solució independent, amb un nou nom que indiqui la seva funcionalitat. Dins d'aquesta nova solució quedarà encapsulada la funcionalitat en forma de llibreria DLL i podrà ser integrada per separat. El projecte original ara farà ús de la nova llibreria referenciant el nou paquet Nuget (explicat en propers apartats).
- No ha de contenir referències locals (per mitjà d'un *path* del sistema de fitxers) a altres projectes d'altres solucions. Només pot tenir referències a projectes de la mateixa solució. Si per exemple un altre programador modifiqués el projecte extern, el nostre projecte quedaria afectat indirectament sense tenir control ni constància.
- Totes les referències a llibreries externes de la solució han de ser:
  - Referències als propis *assemblies* disponibles al *framework* .NET.
  - Referències a llibreries de repositori de paquets NuGet.
  - **No** han de ser referències a cap DLL local continguda en les carpetes del sistema de fitxers de la solució. Cal evitar aquesta mala pràctica ja que no hi ha cap control automàtic ni enregistrat de la versió de la DLL ni del seu codi font. A més no es disposa de cap mecanisme d'actualització que garanteixi la compatibilitat amb el codi del projecte.

  Totes les DLLs de l'organització que es facin servir, s'han de separar en solucions independents, i com es veurà en propers apartats, ser emmagatzemades en un repositori intern de paquets en format NuGet.

- La solució ha d'incorporar els projectes de *test* necessaris per a validar les funcionalitats del codi. Aquests test seran els que s'executaran automàticament en el procés de *build* en el servidor de IC.

En la següent figura es pot veure un exemple bàsic de l'estructura d'un projecte en VisualStudio amb l'estructura i nomenclatura correctes. Al nivell superior hi ha la solució. La solució és un contenidor de N projectes. En l'exemple es tenen 3 projectes diferents: la llibreria principal, un de test des de línia de comandes i un altre que defineix els

UnitTest de la llibreria principal. En la figura també es poden veure per al primer projecte les referències a altres llibreries. En aquest cas la majoria són cap a components proveïts pel propi *framework* .NET. En al cas de la llibreria NLog és pot veure que es tracta d'una referencia instal·lada per NuGet. Un cop descarregada de repositori es guarda a la carpeta *packages* de la solució (veure *Path* en la part inferior de la imatge).

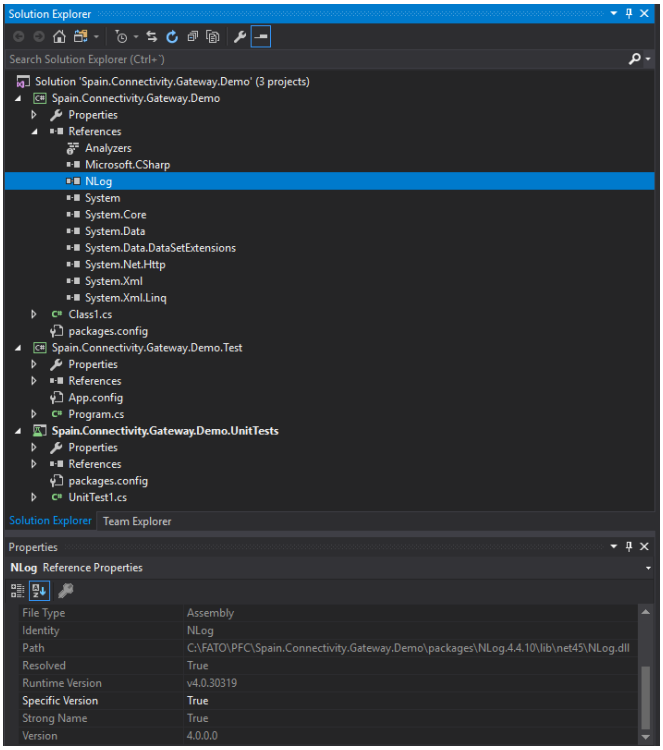


Figura 33: Exemple bàsic estructura projecte en VisualStudio

4.3.3 Normalització de plans de configuració dels projectes

En VisualStudio, cada solució té per defecte assignats dos plans de configuració. Un de Debug i un de Release. Cada pla defineix configuracions relacionades amb les propietats d'arquitectura i CPU i d'altres paràmetres de compilació i un fitxer de configuració associat a aquell pla. En aquest fitxer de configuració es poden definir propietats i variables que es podran fer servir des de les aplicacions. Aquestes variables poden ser personalitzades dinàmicament en funció de l'entorn en que es compila i s'empaqueta l'aplicació. Concretament, els valors dels paràmetres de configuració del directori, fitxers de logs, paràmetres de connexions de BD, credencials, *paths* o *urls* a recursos externs, entre d'altres, han de ser diferents per a cada entorn. Per exemple, si l'aplicació s'executa a l'entorn de proves, la connexió de BBDD serà diferent que si s'executa en producció. El mateix pot passar per a les credencials o paths de fitxers de logs.

Es proposa, fent referència als apartats anteriors de normalització dels projectes i anàlisis dels entorns, definir en cada projecte els següents plans de configuració:

- **Debug**. Correspon a l'entorn emprat quan es depura l'aplicació en l'equip local del desenvolupador.
- **Test**. Correspon a l'entorn de test-integració.
- **PreRelease**. Correspon a l'entorn de PreProducció.

- **Release**. Correspon a l'entorn de Producció.

D'aquesta forma, com es veurà en propers apartats, serà possible indicar en el pla d'integració de l'eina de IC quin pla de configuració del projecte es farà servir. En conseqüència també quedarà definida amb quina configuració es crearà l'*artifact* de l'aplicació a publicar.

4.3.4 Configuració de solucions tipus llibreria per a paquet NuGet

Un pas important en la reordenació de les solucions és la refactorització del projectes de tipus llibreria DLL. Aquests seran integrats en el servidor de IC i dipositats al repositori de paquets NuGet per a que estiguin fàcilment disponibles per a la resta de desenvolupadors.

Per a permetre que el projecte es pugui integrar cal afegir en el projecte un fitxer descriptor de la llibreria anomenat: ***nomdelprojecte.nuspec***. Es tracta d'un fitxer XML amb informació relativa al nom, copyright, Url del repositori, tags, etc. Aquesta informació la farà servir el servidor de IC per a fer el *build,* aplicar versionatge, empaquetar els *artifacts* obtinguts i fer la publicació dins el repositori NuGet. A més a més caldrà tenir correctament configurats els *namespaces* i les propietats d'*Assembly Information* a les propietats del projecte per a que el procés funcioni correctament. A continuació es pot veure un exemple de fitxer *nuspec* i configuració de les propietats de *l'assembly* del projecte.

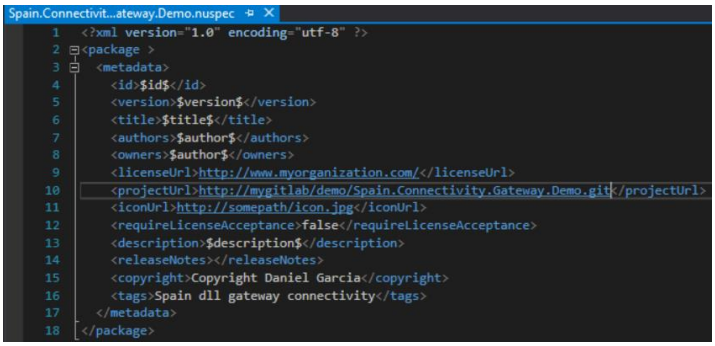


Figura 34: Exemple de descriptor Nuget nuspec

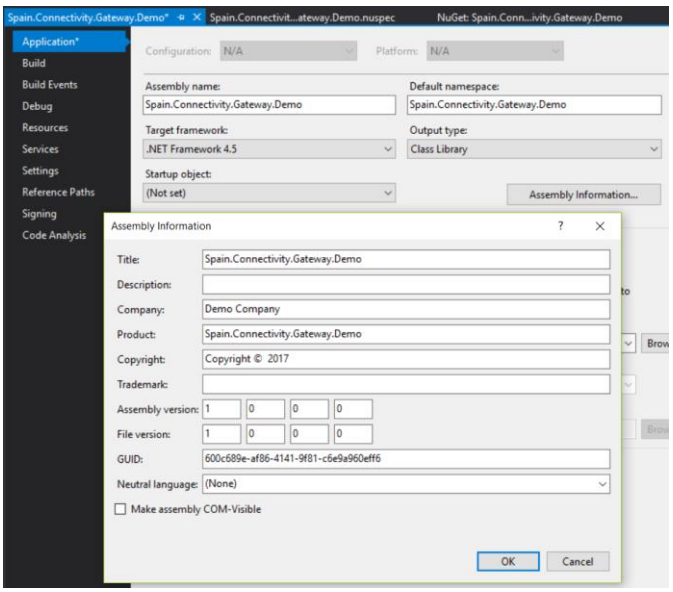


Figura 35: Configuració de l'Assembly Information del projecte en VisualStudio

4.3.5 Repositori de paquets .NET NuGet – ProGet

Dins de la programació d'aplicacions en .NET existeix un mecanisme estàndard i àmpliament reconegut per la comunitat de desenvolupadors per a compartir codi i llibreries de funcionalitats en format de paquets (*packages*) NuGet [32]. Aquest paquets contenen encapsulades les diferents llibreries compilades en format DLL i els fitxers o recursos necessaris del projecte per a ser consumits. Més informació que es pot incorporar al paquet són els símbols de debug i de codi font, que permetrà posteriorment al programador depurar la llibreria descarregada des de NuGet. Aquests paquets són fitxers en format zip i extensió ".**nupkg".**

La forma correcta de gestionar els paquets NuGet és a través d'un repositori dedicat i compatible amb NuGet.

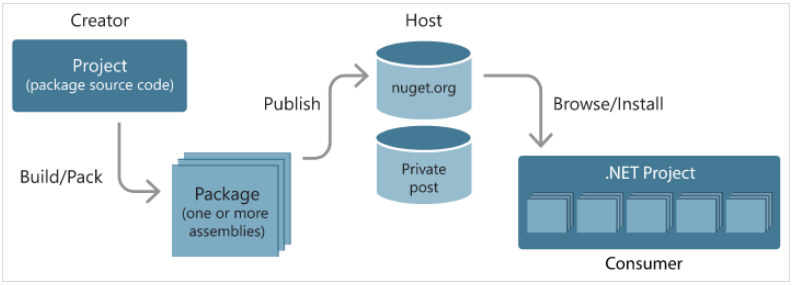


Figura 36: Flux de treball amb paquets NuGet

Aquest repositori, de la mateixa forma que ho fa un repositori de codi font, allotja i enregistra totes les diferents versions del paquets i de les seves dependències amb altres paquets. Aquest últim punt és **clau** per a poder implementar correctament un procés d'IC. És necessari que cada paquet només es pugui fer servir si és compatible amb l'entorn (versió .NET Framework) i amb les versions de les altres llibreries amb els quals té dependències i que formen part del projecte.

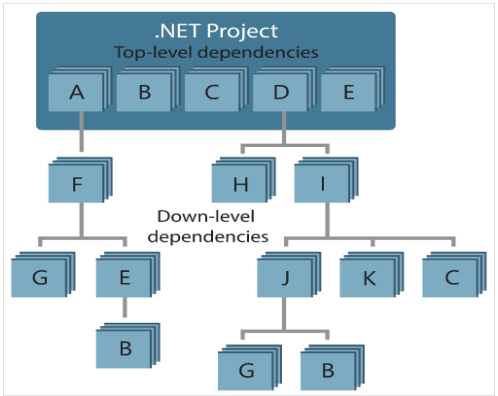


Figura 37: Arbre de dependències de llibreries d'un projecte .NET

Tal com es pot veure en la figura anterior de dependències, NuGet gestionarà totes les sub-dependències dels 5 paquets que composen el projecte original i instal·larà les versions correctes que satisfacin les compatibilitats de tots ells. En el cas del paquet "B", s'instal·larà la versió que sigui compatible amb el projecte i les llibreries E i J. Tot aquest control és gestionat automàticament. En cas de no poder satisfer les dependències, el paquet no es podrà instal·lar.

Es proposa fer servir el software ProGet [33] com a gestor de paquets NuGet de l'organització, on s'emmagatzemaran totes les llibreries produïdes i des d'on estaran disponibles per a fer-se servir en qualsevol moment. Aquest gestor de paquets universal és compatible amb l'estàndard NuGET i proporciona diferents opcions de llicència i de funcionament. Es pot descarregar gratuïtament i ser instal·lat en un servidor local de l'organització. També es poden contractar llicències de pagament per a serveis de *hosting* online amb suport d'escalabilitat i balanceig de trànsit, basats en AmazonS3 i Microsoft Azure Blob Storage. Aquestes funcionalitats estan enfocades per a grans empreses amb necessitats d'alt rendiment i disponibilitat. Per a les necessitats de la nostra organització i el volum de programadors que hi treballen es considera suficient la versió gratuïta per a fer-la servir com a repositoris NuGet local.

El servidor es pot instal·lar tant en entorns Linux com a Windows. Per raons de rendiment i eficiència i cost de llicències, es proposa crear un nou servidor Linux dedicat amb una distribució Centos 7 dins el DataCenter de la seu central. S'instal·larà la versió 4.4 de ProGet que està oficialment suportada per a entorns Linux.

Un cop instal·lat, ProGet pot organitzar els paquets en diferents *feeds*. Cada *feed* pot correspondre, per exemple, a una àrea determinada de l'organització. ProGet és un servidor que pot gestionar de forma centralitzada diferents tipus de paquets en altres formats, els quals poden estar a altres repositoris interns o externs. Un cop instal·lat cal configurar connectors cap a altres repositoris, com per exemple al repositori oficial de Nuget (https://www.nuget.org/api/v2/) i cap a altres *feeds* interns si s'escau.

El servidor ProGet té una interfície web des d'on es podran configurar els diferents paràmetres del servidor així com navegar i cercar dins el catàleg de paquets hostatjats. Donat un paquet en concret es podrà veure el registre de les diferents versions, dates de publicació, dependències, accedir als fitxers físics i informació de com instal·lar per línia de comandes cada versió en particular del paquet.

4.3.6 Client Nuget – Visual Studio NuGet Package Manager

Els programadors de l'organització gracies a ProGet disposaran d'un mecanisme per a dipositar, versionar, controlar dependències i accedir als diferents paquets en format NuGet generats com a resultat d'un procés d'integració. Per a fer servir els paquets interns de l'organització o paquets oficials del repositori de NuGet, cal tenir instal·lat al IDE Visual Studio el gestor "*NuGet Package Manager*". Aquest gestor està ja integrat amb el IDE que fan servir programadors de l'empresa i cobreix perfectament les necessitats. Per a configurar-ho cal simplement donar d'alta al *Package sources Manager* el repositori ProGet local de l'organització i també l'oficial de NuGet. La següent figura mostra la pantalla del gestor de repositori NuGet de VisualStudio.

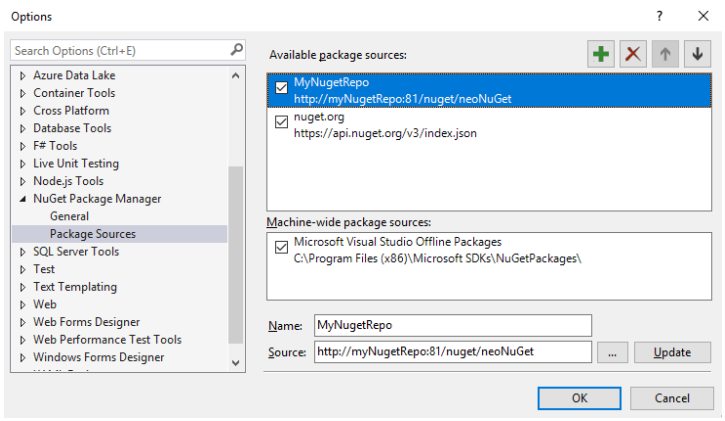


Figura 38: Gestor de repositoris NuGet de VisualStudio

Gràcies a la introducció dels paquets NuGet i l'ús del gestor de paquets de Visual Studio el programador pot accedir de forma visual a una informació important i necessària per fer la seva feina, com:

- Quins paquets, llibreries i quines versions exactes estan instal·lades en cadascun dels projectes de la solució.
- Si hi ha actualitzacions disponibles i quines dependències tenen.
- Informació del paquet (comentaris, autor, url, tags, etc).

A més serà possible dur a terme les següents accions:

- Cerca de nous paquets en tots els repositoris o en el seleccionat al *Package source*.
- Actualitzar a la versió desitjada, ja sigui més nova o més antiga.
- Desinstal·lar completament un paquet i totes les seves dependències que no es facin servir des d'altres paquets.

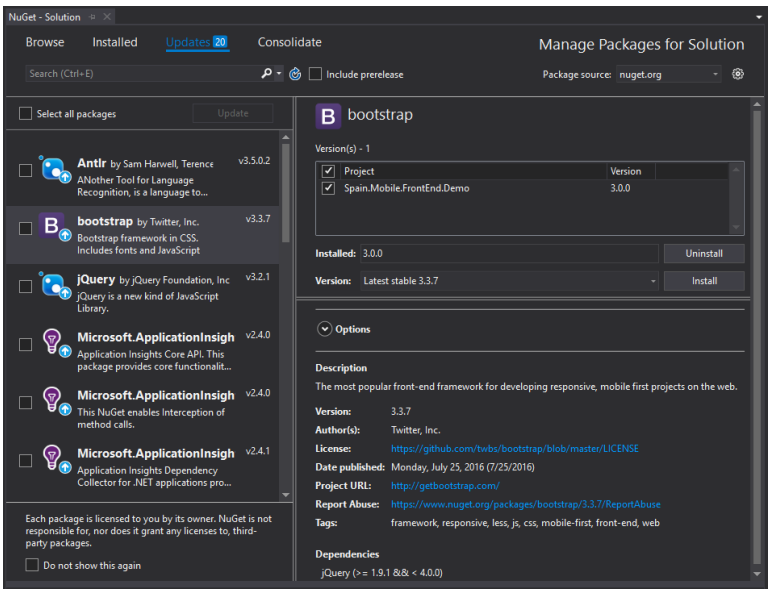


Figura 39: NuGet Package solution manager del VisualStudio

En la figura anterior es pot veure, per exemple, les actualitzacions de paquets instal·lats disponibles. En la finestra de la dreta és pot visualitzar el detall de la llibreria *bootstrap* i s'informa que hi ha una nova versió 3.3.7 més recent que l'actual instal·lada (v 3.0).

La gestió de les llibreries desenvolupades per altres programadors, que han sigut integrades i dipositades a traves d'un procés de IC al repositoris de NuGet és un gran avenç en la forma de treballar de l'equip. Ara tot el procés serà molt més automàtic, es tindrà total control de les llibreries que es faran servir i es sabrà en tot moment qui, què i quan ho ha fet. En aquest model, es pot confiar en el paquet NuGet perquè és resultat d'un procés previ d'Integració Continua que valida el seu correcte funcionament.

La introducció del repositori NuGet en l'organització serà tot un salt qualitatiu i un gran avantatge pels programadors perquè:

- Les llibreries de tots els programadors seran integrades més ràpidament.
- El fet d'haver superat un procés d'Integració garanteix que la llibreria del repositori és funcionalment segura.

- Seran correctament versionades i controlades: es sabrà qui i quan ha publicat un canvi i sempre estaran disponibles al repositori.
- Es permetrà gestionar de forma molt més automàtica i centralitzada l'actualització de qualsevol llibreria. Només cal publicar-la al repositori i la resta de programadors podran rebre notificacions d'una nova versió disponible. Fent servir el gestor integrat d'actualitzacions de VisualStudio, l'actualització serà automàtica i fàcil. Si es compara amb qualsevol mecanisme anterior manual, en què es compartien directament les DLLs, copiant-les en alguna carpeta en xarxa, per FTP, a penDrives, etc sense cap tipus de control automàtic de les versions, el canvi és dràsticament positiu.

4.4 Anàlisis de recursos necessaris

Com es descriurà en els propers apartats, la implantació de la plataforma de Integració Continua en l'organització comportarà força canvis en l'organització. A banda dels canvis purament de procediments i de metodologies, caldran canvis en els recursos tecnològics de l'organització. Aquests canvis estaran centrats en els servidors dedicats a la plataforma de Integració Continua i els necessaris per a poder disposar d'un entorn de Test i de PreProducció.

En la següent taula es resumeixen els servidors dedicats que es necessitaran per a la proposta de plataforma d'Integració Continua.

Cluster	Nom Servidor	Descripció	Sistema Operatiu	Serveis	CPU	RAM	Disc Dur
Plataforma CI	CI_GITLAB_01	Servidor GitLab Corporatiu	Linux Centos 7	GitLab PostFix Nginx PostGreSQL	CPU 2x2.67GHz 64-bit	4 GB	500GB
	CI_PROGET	Servidor ProGet	Linux Centos 7	ProGet	CPU 2GHz 64-bit	4 GB	500GB
	CI_BAMBOO	Servidor CI Bamboo	Windows Server 2012 R2	Bamboo Tomcat	CPU 8x2.67GHz 64-bit	8 GB	1 TB
	CI_BAMBOO_RM01	Servidor Agent Remot Bamboo 1	Windows Server 2012 R2	Bamboo Remote Agent	CPU 4x2.67GHz 64-bit	8 GB	100 GB
	CI_BAMBOO_DB	Base Dades Bamboo	Linux Centos 7	PostGreSQL	CPU 4x2GHz 64bit	4 GB	250 GB
Entorn Test	CI_TEST_FE_01	Servidor FrontEnd Integració Test	Windows Server 2012 R2	IIS	CPU 2x2.67GHz 64-bit	8 GB	50 GB
	CI_TEST_BE_01	Servidor BackEnd Integració Test	Windows Server 2012 R2	Backend	CPU 2x2.67GHz 64-bit	4 GB	50 GB
	CI_TEST_DB_01	Servidor BD Integració Test	Windows Server 2012 R2	SQL Server	CPU 2x2.67GHz 64-bit	16 GB	100 GB
	CI_TEST_LINUX_01	Servidor Linux Integració Test	Linux Centos 7	Tomcat	CPU 2GHz 64-bit	4 BG	40 GB
Entorn Pre	CI_PRE_FE_01	Servidor FrontEnd Integració Pre	Windows Server 2012 R2	IIS	CPU 2x2.67GHz 64-bit	8 GB	50 GB
	CI_PRE_BE_01	Servidor BackEnd Integració Pre	Windows Server 2012 R2	Backend	CPU 2x2.67GHz 64-bit	4 GB	50 GB
	CI_PRE_DB_01	Servidor BD Integració Pre	Windows Server 2012 R2	SQLServer	CPU 2x2.67GHz 64-bit	16 GB	100 GB
	CI_PRE_LINUX_01	Servidor Linux Integració Pre	Linux Centos 7	Tomcat	CPU 2GHz 64-bit	4 BG	40 GB

Figura 40: Taula resum servidor plataforma IC

4.5 Instal·lació d'un repositori de Codi Font

Tota organització de desenvolupament de software té la necessitat de guardar i compartir el codi font d'una forma estructurada i segura. En un entorn on es volen aplicar mecanismes d'Integració Continua la primera peça clau d'aquesta plataforma és el repositori de codi font. S'ha de triar un repositori plenament confiable i suportat per l'eina de IC triada. A més, ha de complir altres criteris importants com: suport, una comunitat al darrera que el recolzi, rendiment adequat, ser una eina estàndard en el mercat i encaixar en el context actual des del que parteix l'empresa en aquesta transició.

4.5.1 GitLab

Es proposa l'ús com a servidor repositori de codi font i control de versions l'eina GitLab [34]. GitLab és una eina de gestió de versions de projectes que permet la instal·lació en

un servidor local de l'organització d'un complet gestor de repositori GIT privats. El projecte és OpenSource i es pot revisar i contribuir en el seu codi font si es vol. Un gran avantatge de GitLab és que permet crear infinits repositoris privats locals, ideals per a projectes d'empreses privades, sense cap cost. GitLab no és solament un gestor de GIT, sinó que també ofereix eines de col·laboració en comunitat, com ho fa el seu gran competidor homòleg del mercat GITHUB.

El procés d'instal·lació és molt senzill i es proveeixen paquets oficials d'instal·lació per a diferents distribucions Linux, entre elles Centos [35].

GitLab encaixarà perfectament en organitzacions amb equips de desenvolupament que ja han treballat amb anterioritat amb repositoris i projectes GIT. Amb l'ús conjunt de VisualStudio i el del seu *plugin* client de GIT, es permet treballar perfectament amb els repositoris gestionats amb GitLab.

Caldrà doncs realitzar una migració, després de normalitzar tots els projectes com s'ha exposat anteriorment, del repositori central GIT actual al nou repositori de GitLab.

La millor opció per a realitzar aquesta migració és fer una instal·lació des de zero neta de cada projecte creant el projecte buit a GitLab i després amb el gestor de Git local afegir les solucions i tots els recursos al nou repositori.

Cal comentar que si es vol conservar el projecte original i la base de dades amb l'històric, es pot migrar fàcilment fent una clonació i posterior publicació (push) al nou repositori en GitLab. Això dependrà de les necessitats de l'organització i dels recursos que disposi per a realitzar aquesta migració.

4.5.2 Requisits servidor GitLab

D'acord amb la capacitació de l'organització i seguint la documentació oficial de GitLab [36] es proposa configurar un nou servidor amb les següents característiques:

- Sistema operatiu Linux Centos 7.
- CPU amb 2 cores com a mínim.
- 4GB RAM com a memòria mínima recomanada per a suportar fins 100 usuaris.
- Capacitat d'emmagatzematge: Anirà en funció de la mida i de la quantitat de projectes de l'organització. Es recomana per a futurs increments muntar els discs amb LVM (Logical Volum Manager) per a poder afegir disc addicional, o bé fer servir alguna solució d'emmagatzematge en xarxa (NFS, SAN, NAS, etc).
- Base de Dades. GitLab farà servir una BD PostgreSQL amb una capacitat mínima de 10GBs. Aquesta mida dependrà de les mides dels projectes i el creixement futur de l'organització.

4.5.3 Gestió dels projectes en GitLab

GitLab, entre moltes altres capacitats, incorpora un control força complet d'usuaris i gestió de permisos que es tenen sobre els diferents projectes. Per a accedir i contribuir a un projecte, cada usuari ha de disposar d'un compte. A cada usuari se li assigna un perfil que pot ser de tipus: *Guest, Reporter, Developer, Master, Owner*.

Els projectes es poden classificar en grups. En el cas de la nostra organització, es crearà un grup per a cadascuna de les àrees o països considerats en el procés d'estandardització de la nomenclatura del projecte. Es tindrà un grup per "Spain", un altre per "Corporate", etc.

A nivell de permisos d'accés als grups els usuaris s'autoritzen en mode d'edició (pot fer *push, merge*, etc) o bé sols en mode lectura (check out). Es poden limitar perfectament les accions de cada usuari en cadascun dels grups i projectes.

Altres funcionalitats importants de la interfície web de GitLab són:

- Funció integrada de cerca de paraules clau i funció de navegació dins l'estructura de fitxers de la solució. Es pot fàcilment cercar per tags o per paraules dins els diferents projectes del catàleg. Un cop seleccionat un projecte, es poden buscar paraules dins els fitxers de codi font o d'altres que formen la solució.
- Visualització del l'històric dels canvis realitzats i en general de tota l'activitat. Es pot fer a nivell de projecte o de fitxer, per a cadascuna de les branques de la solució. Es pot veure en un sol clic, els canvis i les diferències respecte a la versió anterior.
- Es pot descarregar qualsevol versió del projecte o fitxers durant el temps. Per a cada *commit* es pot obtenir el codi exacte que hi havia en aquell moment. Aquesta funció és molt útil per a poder recuperar codi antic o depurar codi del passat.
- Consultar i afegir *tags* a un *commit* en específic.
- Accions habituals d'un repositori GIT: gestió de branques, *forks*, *pull requests*, editar i fer *commit* de canvis, etc.

El ràpid accés de forma visual a tota aquesta informació facilita molt la feina del programador quan necessita consultar canvis anteriors, històrics de fitxers, reproduir codi antic, fer comparacions, etc. A sota es pot veure un exemple de la pantalla principal de visualització dels fitxers d'un projecte amb GitLab.

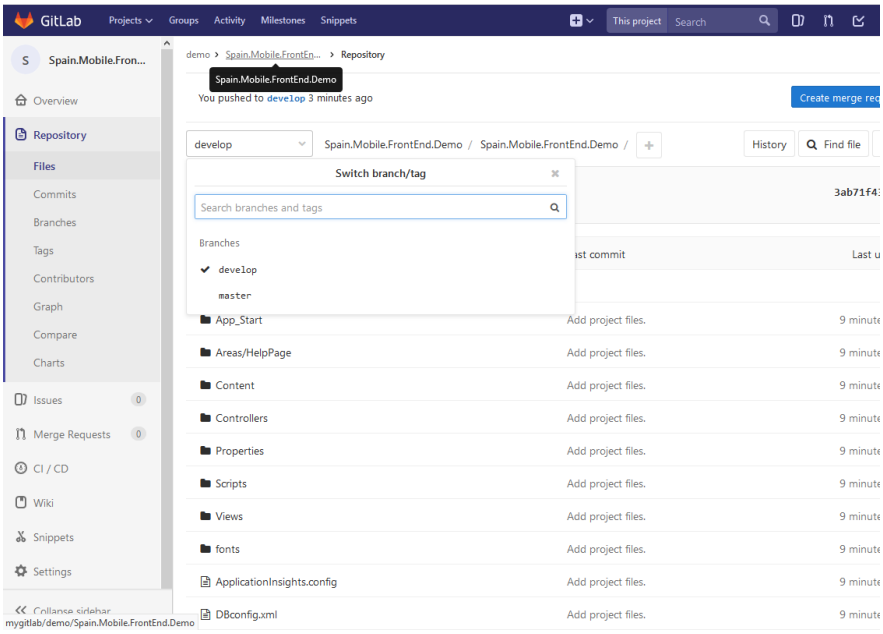


Figura 41: GitLab menú de navegació fitxers d'un projecte.

La connexió al repositori GIT de GitLab és pot fer mitjançant el client Git que es pot descarregar des de la mateixa plana oficial de Git [37]. Amb això es pot treballar amb la consola de comandes Git. En l'IDE Visual Studio es disposa des de les versions 2013 o superiors d'un client GIT que serveix perfectament per a realitzar totes les operacions bàsiques necessàries amb GitLab (o GitHub). Les necessitats de l'organització per a treballar amb el repositori de codi font queden perfectament cobertes amb aquestes aplicacions.

4.6 Instal·lació de l'eina d'Integració Continua

El software Bamboo serà el component principal de tot el conjunt d'eines proposades que formaran part de la nova plataforma d'Integració Continua. Com s'ha indicant anteriorment en la part d'anàlisi, Bamboo es pot instal·lar en qualsevol entorn que pugui executar una màquina virtual Java. Com que la majoria d'aplicacions i projectes de l'organització estan desenvolupats en el *framework*.NET per a plataformes basades en Microsoft Windows, i els servidors són també servidors Windows, es triarà instal·lar Bamboo en una nova màquina Windows dedicada. D'aquesta forma s'intentarà executar la integració en un entorn el més semblant possible al de desenvolupament i producció.

Es proposa triar el pla de llicència que permet executar 1 agent remot i que haurà de suportar un ús mitjà del servidor, amb una configuració de plans locals aproximada d'entre 50 i 60. Es farà un ús limitat màxim de 5 agents locals per no comprometre el rendiment del servidor. Aquests agents locals poden alentir una mica el rendiment del servidor o bé afectar al nivell de concurrència de *builds* en paral·lel. 1 agent només pot processar una tasca, així que si es produeix un pic de càrrega en algun moment i no es disposa d'agents lliures es posarà a la cua i s'haurà d'esperar. De totes formes, donades les dimensions de l'equip de desenvolupament (20 tècnics) i els nombre de projectes (50-60) es considera que no es produiran masses pics de càrrega i que aquesta configuració serà suficient. Caldrà monitoritzar la situació en el futur per si cal incrementar les capacitats del servidor o afegir llicències d'agents remots addicionals.

Caldrà també considerar recursos addicionals de servidors per a poder instal·lar i executar correctament Bamboo. A continuació es detalla l'escenari proposat.

4.6.1 Servidor Bamboo

Es crearà una nova màquina Windows. D'acord amb les especificacions de Atlassian i els nostres requisits, el servidor tindrà les següents característiques:

- Sistema operatiu Windows Server 2012 R2
- 8 GB Ram mínim,
- 8 nuclis
- Disc Dur de 1 TB. L'espai va directament relacionat amb la configuració de Bamboo. El servidor requereix aproximadament 140 MB, però aspectes com quants *test* s'executen en cada pla, quants *artifacts* es generen i quantes version (i durant quan temps) es guarden en l'històric del servidor condicionaran l'espai de disc emprat.
- Servidor d'aplicacions Apache Tomcat 8 amb Java JDK 1.8. Tomcat és l'única opció suportada per Atlassian i no suposa cap cost addicional de llicència.
- Connector a BD PostgreSQL incorporat amb Bamboo. Aquesta BD s'instal·larà en altre servidor dedicat.

4.6.2 Servidor Base de Dades

Bamboo suporta diferents gestors de Bases de Dades relacionals. Per a entorns productius no es pot fer servir el gestor HSQLDB incorporat ni es recomana la instal·lació d'un altre gestor en el mateix servidor. Es proposa triar el Gestor de BBDD PostgreSQL versió 9.6 (l'última versió 10 sembla no estar encara suportada) [38], per la seva facilitat

d'ús, manteniment, amplíssim ús i per ser principalment OpenSource i gratuït. A més, és la primera opció recomanada pel fabricant.

S'instal·larà en un servidor amb sistema operatiu Linux Centos 7. Aquesta distribució està totalment suportada per PostgreSQL, dona un bon rendiment i no suposa cap cost addicional de llicències. S'haurà de configurar el *firewall* local i el gestor de BBDD per a permetre l'accés des de la IP del servidor de Bamboo. Caldrà també configurar correctament en el menú d'administració de Bamboo l'accés a la BD externa fent servir el *driver* JDBC natiu. Addicionalment es tindrà cura de les propietats de ***connection límit*** i ***pool size*** si es donen problemes d'accés, lentitud de la interfície d'usuari o altres errors de connexió.

4.6.3 Servidor Agent Remot Bamboo

La llicència proposada permet fer servir 1 agent remot local. Tot i que nomes hi ha un agent, al tenir aquest un servidor i una màquina virtual Java dedicades, podrà realitzar la feina molt millor i més ràpidament que els agents que s'executen en local en la mateixa màquina del servidor.

Es configurarà Bamboo per a que assigni més càrrega a aquest servidor dedicat i no comprometre el servidor central. Aquest servidor addicional també serà Windows i tindrà instal·lada la mateixa versió Java JDK 1.8 per garantir la homogeneïtat entres els agents locals i remots. Les característiques d'aquest servidor dependran de la capacitat i càrrega que es vulgui assumir, factors que afecten al temps d'espera mitjà per a fer els *builds* corresponents. En aspectes de disc dur, no es requereix tanta quantitat com al servidor central, en tant que no han de guardar-se els *artifacts* i altres arxius temporals.

Caldrà instal·lar el software agent remot, el qual es pot obtenir des del menú d'administració del servidor Bamboo. Es copiarà el jar descarregat al directori de treball del servidor remot i s'executarà des d'una línia de comandes l'agent amb la següent comanda:

```
java -jar atlassian-bamboo-agent-installer-X.X-SNAPSHOT.jar
http://mybambooserver:8085/bamboo/agentServer/
```

on X.X serà la versió de Bamboo final instal·lada al servidor. Per línia de comandes es poden configurar altres aspectes com directoris de treball, fitxer de logs, o instal·lar-ho com a servei de Windows (opció `installntservice`) , entre d'altres. Es recomana optar pel mode de servei en la instal·lació.

Caldrà també configurar al servidor les *capabilities* d'aquest agent, indicant quins tipus d'eines i de *builds* disposa.

4.6.4 Configuració de Bamboo

En aquest apartat es descriuran les principals característiques i configuracions de l'eina Bamboo que s'han de considerar per a fer-lo servir com a servidor de IC de l'organització. No es tractarà en cap cas d'un manual complet sobre totes les opcions i parametritzacions que es poden fer servir en l'eina. Aquestes caldran ser completament estudiades i gestionades pels administradors del sistema amb la documentació oficial de Atlassian. S'identificaran els principals paràmetres i accions a tenir en compte per a "connectar" el servidor a l'entorn i a la resta d'eines de la plataforma (BBDD, Gitlab, servidores productius, etc) i definir una proposta de configuració de plans en projectes tipus.

4.6.4.1 Instal·lació i configuració dels principals paràmetres

En la instal·lació per a entorns Windows, es pot fer servir un Wizard que preguntarà els directoris d'instal·lació i de treball a configurar. Un cop instal·lat caldrà executar-lo per primer cop via línia de comandes "`bin\start-bamboo.bat`". Amb el servidor iniciat es podrà accedir en local des d'un navegador a la interfície web http://localhost:8085/. També es disposarà d'una carpeta al menú d'aplicacions d'es don es podrà accedir a la utilitat de configurar Bamboo com a un servei de Windows. Aquesta opció és recomanable per evitar problemes en cas de reinici inesperat del servidors o fallida.

Un cop s'executa per primer cop, es llençarà l'assistent de configuració on es podran indicar els següents paràmetres:

- Configuració de la llicència.
- Mètode de configuració (*custom*, express).
- Configuració servidor (adreça principal, domini, directoris local de treball, Broker urls per agents remots).
- Base de dades, en el nostre cas externa tipus PostgreSQL. Caldrà introduir totes les dades de connexió i usuaris, *driver*, nom de la BD, etc.
- Creació d'un nou entorn de dades des de zero o importar-ne un d'existent.
- Gestió usuari administrador.

Agents

Des del menú d'administrador es pot accedir a múltiples seccions de configuració. El primer que cal configurar són els agents que estaran disponibles pels plans. Aquests seran locals o remots. En la següent figura es pot veure un exemple del menú d'administració de Bamboo desplegat amb 5 agents locals. En el nostre cas també es configurarà un de remot fent servir el botó de *Install Remote Agent*.

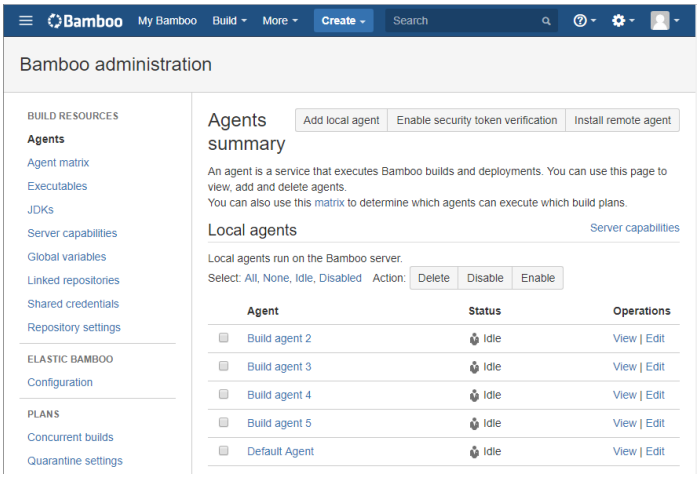


Figura 42: Bamboo menú configuració d'agents

Java JDKs i altres executables

Cal registrar dins de Bamboo les versions de Java JDK disponibles en el servidor per a que puguin ser utilitzades pels agents i mòduls de Bamboo. També cal instal·lar i declarar els diferents executables que es faran servir des dels agents, com poden ser comandes de NuGet, d'execució de tests, *builds*, o altres a mida. Qualsevol executable extern ha d'estar sempre configurat per a poder fer-se servir.

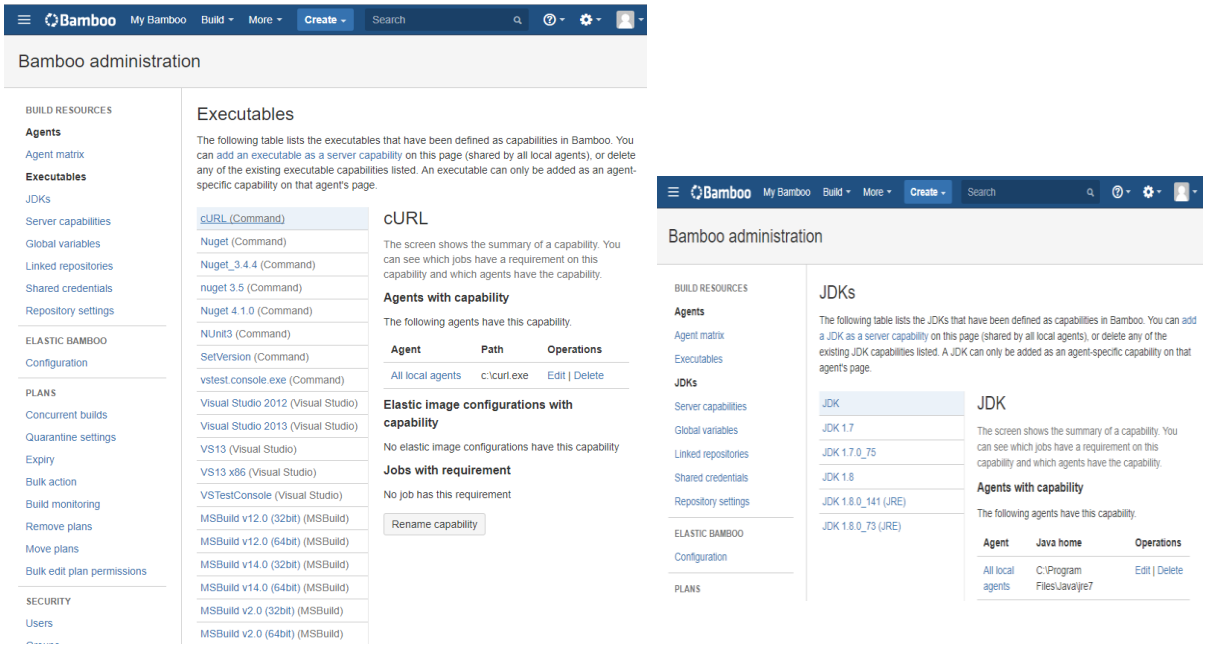
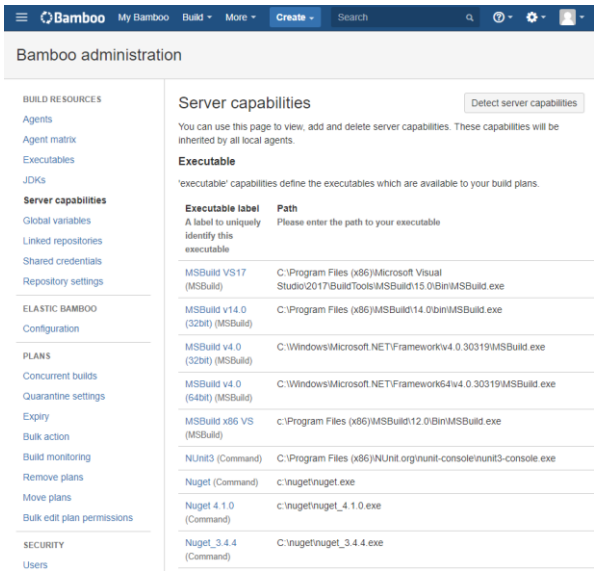


Figura 43: Bamboo configuració d'executables i version de Java

Capabilities

Un cop es tenen els diferents recursos d'executables com MSbuilds, Java o Git, es poden definir les diferents capacitats que es podran assignar als agents. Això determinarà que pot fer i que no pot fer cadascun dels agents. En cada pla d'integració es podrà fer ús de la *capability* necessària assignant en la configuració l'etiqueta corresponent a la *capability*. En l'exemple de sota es poden veure capacitats per a fer *builds* de diferents versions de projectes de visual Studio amb la comanda MSBuild.



Security: Users and Groups

En el menú d'administració també es podran configurar els usuaris i els grups als quals pertanyen. Bamboo pot configurar per a cada usuari el seu nivell d'accés als projectes i als grups. Serà important fer un correcte control i una bona configuració dels permisos per assignar només els estrictament necessaris. Per exemple hi haurà projectes crítics

per a l'organització que només podran ser integrats i publicats en producció per un usuari en concret. Les opcions de control d'usuaris es poden fàcilment adaptar a les necessitats de seguretat de l'organització.

Altres qüestions importants a configurar serien els logs, les polítiques de neteja i expiració, variables de sistema, etc. Es objectiu de l'administrador el coneixement i gestió de totes les opcions de configuració de Bamboo.

4.6.4.2 Configuració dels plans

Totes les operacions de configuració dels plans es duran a terme per un nou perfil responsable dins els equips de desenvolupament o bé dins l'equip d'administradors de sistemes. Es recomana separar i assignar aquest rol a persones específiques per a garantir que tota la configuració i administració de Bamboo és homogènia i coherent. No és una bona pràctica permetre als desenvolupadors directament configurar i administrar els plans dels projectes en els quals treballen. Caldrà establir un procediment de petició de tasques de configuració dels plans, per exemple fent servir l'eina Jira interna, a l'equip administrador. La idea és que un pla, un cop està configurat i funcionant, sigui un recurs disponible per al desenvolupador, però tancat en tant que només podrà accedir als seus resultats i *logs* sens fer cap modificació en ell.

Cada projecte de l'organització en GitLab estarà mapejat 1 a 1 a un Pla en Bamboo. Cada pla de Bamboo estarà inclòs dins un projecte. El projecte serà el contenidor de plans. Al igual que en GitLab, es proposa crear tants grups (projectes dins Bamboo) com a països o àrees s'han definit durant el procés de normalització i revisió de les nomenclatures dels projectes. D'aquesta forma l'organització per grups serà la mateixa en les dues eines i serà més fàcil pels seus usuaris trobar i gestionar els projectes.

En Bamboo es poden configurar dos tipus de plans principalment: el plans de *build* i els plans de Deployment. Els primers seran els que portaran a terme tots els passos relatius a la integració i validació del projecte. El segons, seran els que en funció del resultat del *build* publicaran els *artifacts* obtinguts en una ubicació o repositori indicats. L'aplicació pot ser publicada, per exemple, un en servidor d'aplicacions IIS, una aplicació tipus servei que s'instal·la en un servidor, una aplicació web o una llibreria NuGet. Per a cada tipus de projecte es definirà un pla de *build* i de *deployment* específics.

4.6.4.2.1 Pla de build

A molt alt nivell per a cada projecte caldrà fer el següent:

- Des del menú d'administrador crear un nou pla. Es podrà triar si s'inclou dins d'un projecte ja existent (en el nostre cas grup de país, etc) o es crea un de nou.
- Configurar com a nom del pla el nom estandarditzat del projecte.
- Project key and plan key. Són claus que referencien a cada pla i projecte unívocament. Cal triar amb cura el valor d'aquestes claus ja que no es podran repetir i es faran servir en altres configuracions com a identificador o vincles del pla. Es proposa definir i seguir un mateix estàndard per a tots els plans.
- Definir la *url* del repositori GitLab del projecte. En aquest moment Bamboo lliga el pla amb un repositori en concret i branca.
- Mecanisme de connexió al repositori. Per exemple en el cas de GitLab es pot connectar per HTTP/S o per SSH. Per motius de seguretat es recomana fer servir autenticació per clau privada SSH. Caldrà prèviament generar un parell de claus privada/pública en DSA per exemple i configurar-les correctament als servidors de GitLab i de Bamboo.

Un cop s'ha creat el pla, es crea per defecte un *stage* amb un *Job*. Dins del *Job* es crearan les corresponents tasques, on cada tasca és una operació que s'executa a Bambo a partir d'un executable prèviament configurat al sistema.

Com a procediment general, per a cada projecte caldrà només configurar dins l'*stage* per defecte 1 *job* per a cadascun dels entorns que es volen configurar a Bamboo. Per exemple es poden configurar 3 *jobs*, un per a cadascun dels següents entorns: Test, preproducció i Producció. Això permetrà personalitzar al màxim, en funció de l'entorn, tots els passos i comprovacions del pla d'integració.

Generalment per a cada j*ob* ,mitjançant el *Wizard* de configuració de tasques, es podran afegir tasques a partir d'un conjunt de tipus de tasca ja instal·lats i disponibles a Bamboo especialitzades en operacions comunes del *jobs*. Aquestes tasques predefinides simplificaran molt la creació dels jobs i evitaran l'ús d'*scripts* locals o externs més difícils de mantenir.

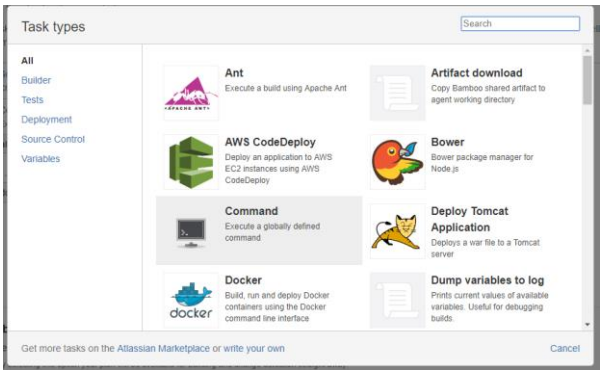


Figura 44: Bamboo task Wizard

Com a resultat final si es pren com a referència un projecte de tipus llibreria DLL, es poden crear les següents tasques que executaran tots els passos necessaris per a fer el *build*. A continuació es detallen.

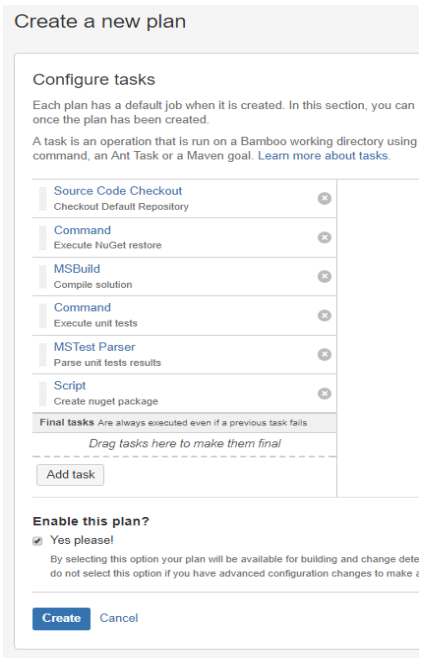


Figura 45: Llistat de tasques d'un Job genèric de Release en Bamboo

- Source Code Checkout. Comanda per defecte que obté el codi font del repositori GitLab prèviament configurat.
- Command – Execute NugetRestore. Tasca de tipus comanda que invoca un executable. En aquest cas és la comanda nuget, que es farà servir per a descarregar tots els paquets NuGets que faci servir la solució de la mateixa forma que es fa dins VisualStudio quan es compila la solució.
- MSBuild. Tasca que executa la comanda MSBuild per a compilar la solució de la mateixa forma que ho faria VisualStudio. D'aquesta forma es compila completament de nou el projecte en la màquina de Bamboo.
- Command – UnitTest. Comanda que invoca l'executable vstest.consolo.exe. Aquest executarà els UnitTests associats a la solució i escriurà els resultats en un fitxer de sortida en format XML. Bamboo incorpora molts *plugins* de tasques per a tractar amb diferents tipus de test i analitzar els seus resultats.
- MSTtestParser. Comanda que analitzarà i interpretarà els resultats del UnitTest anteriorment executat a partir del fitxer XML generat.

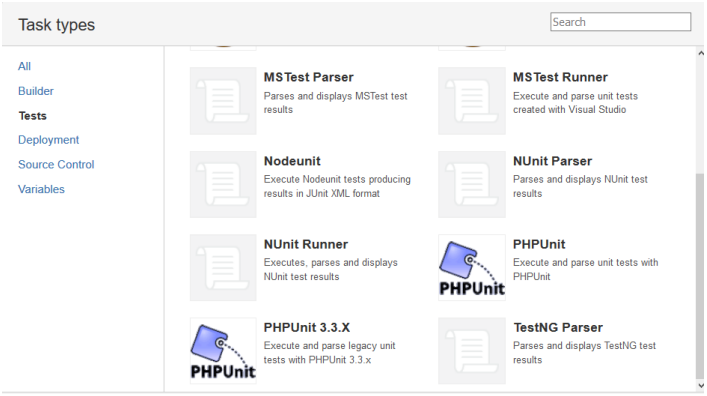


Figura 46: Bamboo tasques pre configurades per a validar tests

- Script – Create NugetPackage. Comanda per a executar un script local. En aquest cas l'*script* simplement executarà la comanda nuguet.exe amb una sèrie de paràmetres per a crear el paquet NuGet. El resultat es farà servir més tard com a *artifact*.

Per acabat caldrà crear una definició dels *artifacts* generats (noms i formats), en el menú d'artifacts del *Job*. Amb això es té configurat un *build* genèric per a un projecte de tipus llibreria DLL. Ara caldrà lligar un *Deployment Plan* amb el pla de *build* fet i així poder publicar els *artifacts* finals obtinguts. En el cas del projecte d'exemple (llibreria DLL) consistirà en publicar el paquet NuGet generat al repositori de NuGet hostatjat al servidor de ProGet intern de l'organització.

En el cas de projectes tipus servei o aplicacions, una possible proposta és afegir dins de la solució de VisualStudio un gestor de paquets d'instal·lació de tipus "Windows Installer". Es pot fer servir la utilitat WixToolset, que disposa d'un *plugin* que s'instal·la a VisualStudio [39]. Un cop configurat té la capacitat de generar com a procés del *build* del projecte un paquet *Windows Installer* compatible. És el que es coneix mes comunament un instal·lador d'aplicacions Windows. En aquest cas, els *artifacts* generats seran els fitxers .exe i .msi generats.

Els plans de *build*, es podran executar sota demanda o bé ser disparats pels esdeveniments que es donen en el repositori de codi font GitLab. Altres tipus de *triggers* es poden configurar, tal com es pot veure al menú de *triggers* del pla. En el nostre cas, es pot configurar el repositori de GitLab per a que dispari l'esdeveniment cap al servidor de Bamboo i recíprocament configurar Bamboo per a que accepti els *triggers* remots

sobre el projecte generats des de GitLab. En GitLab s'ha d'anar al menú de Settings/Integrations del projecte, seleccionant l'*even*t de Push i activant el connector "Atlassian Bamboo CI". Caldrà configurar l'adreça de Bamboo http://mybamboo:8085/ i fer servir la CLAU del projecte indicada durant la creació del pla a Bamboo.

4.6.4.2.2 Pla de publicació

Un cop es té configurat el pla del *build*, caldrà configurar per a cada projecte un "Deployment Plan". Fent clic al botó *Create Related Deployment project* es podrà crear un nou pla. Dins del pla, es podran crear les tasques necessàries per a obtenir els *artifacts* i pujar-los al repositori de Nuget.

En la pantalla que es mostra a continuació es pot veure un resum de les configuracions del pla de *deploy* per a un *environament* específic. Es pot veure el *build plan* associat, l'*artifact* que es publicarà, accedir al conjunt de tasques que executen el *deploy* i configurar els *triggers* que el disparen.

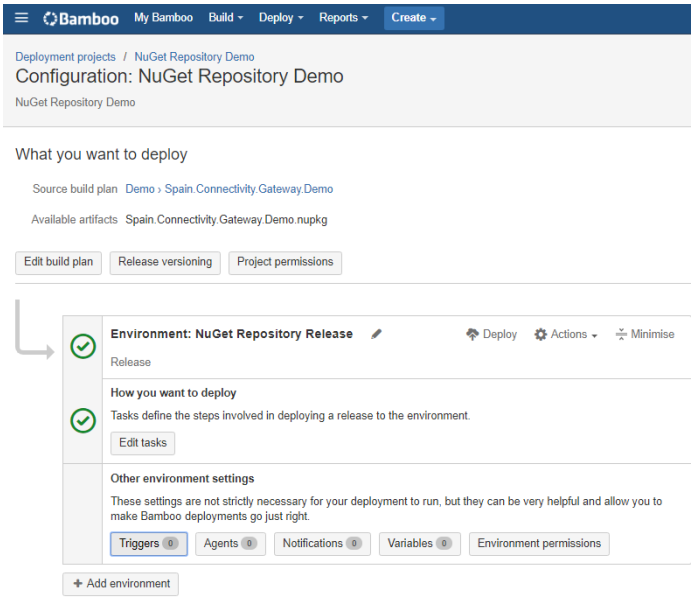


Figura 47: Exemple configuració NuGet Repositery Demo en Bamboo

Depenent de com es configuri el *trigger* de publicació, Bamboo realitzarà el Deployment de forma automàtica (Continuous Deployment), o sinó es configura cap *trigger,* es farà de forma manual (Continuous Delivery). Bamboo proporciona els següents *triggers* automàtics:

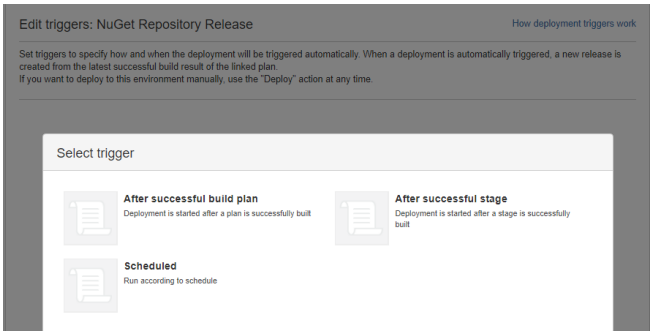


Figura 48: Bamboo triggers automàtics de deployment

En el cas d'aplicacions que es publiquen, per exemple en un servidor IIS, caldrà fer servir una tasca de tipus WebDeployment. Aquesta eina es pot consultar i descarregar de la plana de Microsoft [40]. Bàsicament a partir del script *cmd* generat com a part dels *artifacts* obtinguts amb la comanda MSBuild (recordem, executada dins del pla de *build* previ) es capaç, fent servir la utilitat msdeploy.exe, de publicar de forma automàtica l'aplicació dins el servidor IIS del servidor remot indicat. D'aquest forma no caldrà mai fer cap accés directe al servidor per a poder publicar l'aplicació i tot es farà de forma automàtica.

Per a que un servidor accepti la instal·lació remota amb WebDeployment, cal tenir instal·lat i activat el servei Web Deploy Remote Service. Aquest servei no acostuma a estar instal·lat per defecte i caldrà, amb el suport dels administradors de sistema, revisar la correcte instal·lació. Un cop instal·lat el servei i el *Web Deployment Tool,* cada servidor remot estarà escoltant peticions HTTP en la url i port configurats. Caldrà revisar les regles del Firewall local per a permetre aquestes connexions des del servidor de Bamboo únicament.

En el cas d'aplicacions de tipus servei o executables, Bamboo també pot executar la instal·lació remota. Fent servir un gestor de paquets de Windows compatible es poden instal·lar local i remotament paquets instal·ladors de Windows, amb control de versions, d'actualització, desinstal·lació, etc. Es pot proposar l'ús per exemple de l'eina gestora de paquets Windows Chocolatey [41]. Caldrà tenir instal·lat en els servidors Windows remots el software. Es pot instal·lar fàcilment des d'una PowerShell administrativa executant la comanda: `Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))`

Finalment en el pla de *deploy* de Bamboo per a aplicacions de tipus servei o executables, caldrà configurar una tasca addicional que executi per script la comanda choco.exe que serà l'encarregada de la instal·lació remota del paquet Windows.

4.6.4.3 Operacions en bamboo

Amb l'eina Bamboo es podran dur a terme principalment dos fluxos de treball. Un per a perfils d'administració i un altre per a les operacions diàries dels desenvolupadors. Els administradors de Bambooo seran els encarregats, amb el suport i la col·laboració dels desenvolupadors, de les següents tasques:

- Gestionar totes les configuracions dels plans de *build* i de publicació: alta, baixa i manteniment i redacció dels estàndards a seguir.
- Manteniment del servidor i instal·lació de tots els *plugins* i executables necessaris.
- Creació dels scripts addicionals de suport per a realitzar operacions més complexes. Bamboo de forma general pot invocar l'execució de qualsevols script o comanda local dins una tasca. Això donarà molt marge i flexibilitat per a realitzar operacions complexes, però a la vegada caldrà fer un ús racional i no abusiu a fi de no tenir excessiva lògica i operacions fora de la configuració estàndard de Bamboo.
- En general, amb el suport dels administradors de sistemes, tenir cura de la correcta integració i funcionament amb al resta de components de la plataforma de Integració Continua de l'organització: GitLab, Bases de Dades de suport, Servidor remot de IC, ProGet, etc.

Per part de l'equip de desenvolupadors es realitzaran les següents activitats:

- Cerca i consulta, per projectes, dels plans disponibles.
- Donat un llistat de plans resultat d'una cerca anterior, saber el número, estat de l'últim *build*, la data i l'autor.
- Per a cada pla, accedir a l'històric dels *builds*, als *logs* de cadascun i *commits* associats a cada *build* per a cadascuna de les branques configurades. Des de la vista de detall de cada *build* es podran veure les tasques (*issues*) associades des de Jira a aquell *build*. D'aquesta forma amb un sol clic es podrà accedir a l'aplicació Jira i veure els detalls de la tasca. Aquest és un dels avantatges de fer servir Bamboo en una organització que ja feia ús previ d'altres eines del mateix fabricant com Jira.
- Per a cada pla, fer un *build* sota demanda i veure els resultats per a cadascun dels *steps* i *jobs* que el composen.
- Per a cada pla, accedir als projectes de publicació associats i:
    - Veure per a cada entorn la versió publicada, branca, data i *logs* del *deploy*.
    - Fer sota demanda un nou *deploy* a partir de l'últim *build* o d'un d'anterior. Es pot fer de la branca Master per defecte o de qualsevol altra branca que es llegeixi del repositori de GitLab.
    - Fer *rollback* cap a una versió específica anterior.
    - Accedir directament als *artifacts* emprats en el *deploy*. Això permet veure realment què s'ha publicat, sent molt útil per a confirmar o descartar problemes en els *artifacts* generats.
- En general, accedir a la plana de visualització de l'estat de les cues de *build* i Deployment per saber si està en cua o no una petició en concret.

De forma indirecta, Bamboo pot enviar notificacions per correu electrònic sobre els diferents esdeveniments que es donen al servidor com resultats de processos de *builds* (incloent-hi els test) i inici i final de processos de publicació, entre d'altres.

La informació sobre l'estat i l'històric dels *builds* i de publicacions fetes, juntament amb les notificacions de correu seran de molta utilitat per al treball diari dels programadors i responsables de projectes. Es podrà tenir constància del que ha passat, què està succeint ara mateix i estar informat de qualsevol problema que es pugui donar quasi en temps real.

4.7 Metodologia desenvolupament

Amb la introducció d'un sistema de Integració Continua i amb l'ús de les eines basades en GIT que es faran servir, s'aconsella seguir una metodologia de desenvolupament definida i comuna per a tots els programadors. L'èxit de la IC està directament lligat amb la disciplina, bones pràctiques i procediments de tothom que col·labora i participa des de que es crea el software fins que es publica en producció.

En el capítol d'introducció a la IC ja s'han introduït conceptes i bones pràctiques a adoptar com: el manteniment de les branques, procediments de publicació, correcció immediata dels errors, introduir codi autoverificable, etc. Tot i així ara ens centrarem en un proposta de model de treball basada en **com** els desenvolupadors faran servir el repositoris, les branques i quines relacions i dependències hi haurà entre elles. El model proposat pels equips serà el conegut com a GitFlow.

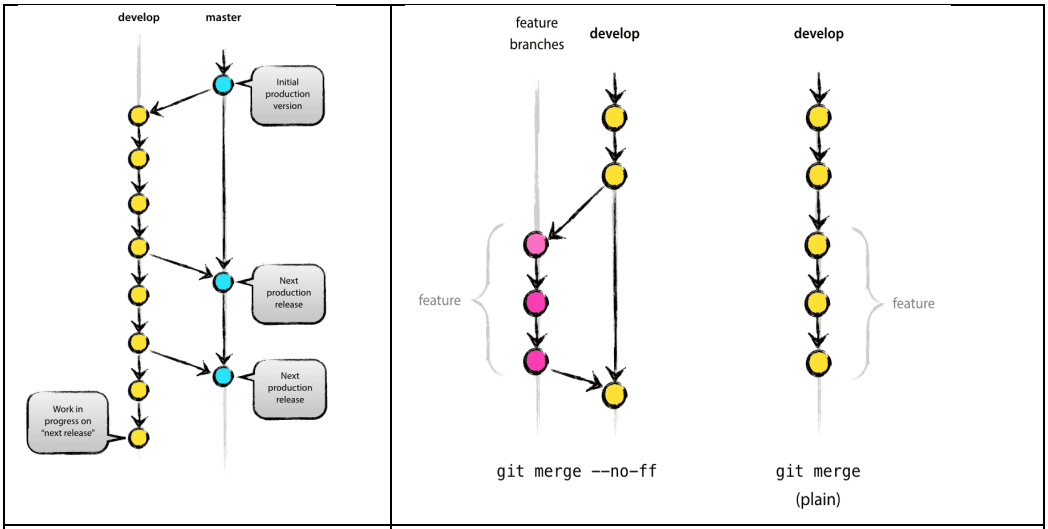
4.7.1 GitFlow

GitFlow és un disseny específic d'un *workflow* de Git que va ser publicat i fet popular per l'enginyer de software Vicent Driessen en un post en el seu blog personal [42]. Aquest *workflow* descriu un model estricte, amb rols ben definits per a cada branca, enfocat en el release de grans projectes.

El model es basa en 2 branques principals:

- **Master**. Branca habitual dels repositoris git que conté el codi oficial de producció i l'històric de tot el projecte.
- **Develop**. Nova branca introduïda com a còpia de la *master*, que conté tot l'històric del projecte i que es farà servir com a branca d'integració de totes les noves *features* fetes pels programadors.

Fins aquí es té un model clàssic en el qual es treballa directament en la branca de *develop* introduint aquí totes les noves funcionalitats. Un cop els canvis són finals, es van traslladant a la *master* (com es pot veure en la figura de sota de l'esquerra)



Model clàssic	Model clàssic + *feature*

Figura 49: Models GitFlow amb feature

- **Feature branches.** En aquest model s'introdueix ara el nou concepte de *feature*. Cada feature és correspon a una nova funcionalitat que s'incorpora al projecte. Cada nova funcionalitat s'ha de programar en una branca nova dedicada exclusivament a la nova funcionalitat. Les *features* es creen com a còpia de la *develop* i contenen els últims canvis de la branca. Un cop s'han fet tots els desenvolupaments, es fa un *merge* cap a *develop*. Ara la branca *develop* conté tota la *feature*, juntament amb altres funcionalitats d'altres equips. La idea és no fer servir *develop* directament com a branca de desenvolupament, sinó sempre *features* independents per no comprometre la feina d'altres equips que desitgin començar noves *features*. Les branques poden tenir com a nom qualsevol nom excepte els reservats: *master, develop, release-*,* o *hotfix-**.
- **Relase Branch.** Un cop la *feature*, o un conjunt de *features*, han sigut correctament integrades a la branca *develop*, es pot generar una nova versió que serà el *release* a publicar més tard Es crearà una nova branca anomenada release-XXX que servirà de suport per a preparar el codi que anirà a producció. Durant el període que es realitza el release, no es poden afegir noves *features* al projecte mentre no es finalitzi aquest. En aquest moment s'afegirà un TAG de versió per a aquell release. Possibles correccions poden ser necessàries per a solucionar petits detalls. Per a cada nou *commit* fet, cal fer el *merge* contra *develop* i *master* per a tenir-los sempre perfectament alineats.

- **Hotfix Branch.** A vegades pot ser necessari en un moment donat, corregir un error que es dona en producció amb la màxima urgència. Les correccions fetes al codi de producció es faran en branques de tipus *hotfix*. Aquestes branques agafen el codi de la *master* i no de *develop* com es fa en el cas de les *features* o els releases. Per a cada correcció feta s'han de propagar els canvis cap a *master* (afegint el corresponent tag) i *develop,* i si hi ha una release en curs, també cap a aquest. La idea del *hotfix* és permetre correccions sota necessitat sense parar el cicle dels altres programadors que estan treballant paral·lelament en *features* o *releases*.

La següent figura mostra a nivell visual el conjunt de les branques i diferents fluxos que es donen en el model definit per GitFlow.

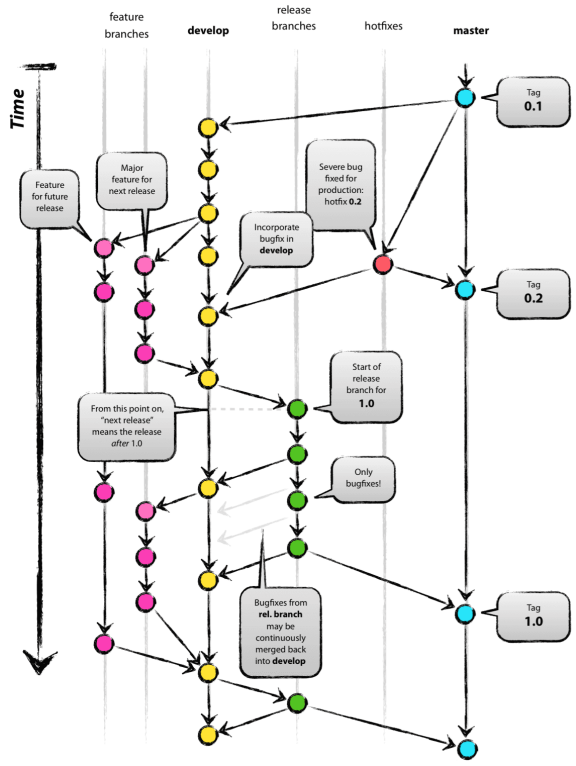


Figura 50: Resum general model de branques en GitFlow

Aquesta metodologia de treball s'haurà d'integrar gradualment en l'equip de desenvolupament. Paral·lelament també es configuraran totes les eines i recursos de la plataforma de IC per a suportar-la i fer-ne el màxim ús.

4.7.2 Convenció comentaris Commits

Per a cada *commit*, cal afegir una descripció associada als canvis realitzats. Aquest *commit*, un cop s'apliqui a les branques del repositori, Bamboo en farà el *build* corresponent. Amb la finalitat de facilitar el vincle des dels *builds* fets amb Bamboo i les tasques Jira associades als canvis del *commit*, sempre que interessi cal posar com a prefix del comentari l'identificador de la tasca entre claudàtors. Per exemple:

"[TEAM1-123] – Canvis en mòdul de logs de l'API"

on l'identificador d'equip Jira es TEAM1 i el número de tasca a Jira és 123. Aquests identificadors els genera Jira automàticament cada cop que es crea una nova tasca.

4.7.3 Conveni de versions del codi

Com a punt addicional al model de treball, cal acordar i proposar un conveni de versions per al codi font. Hi ha diferents metodologies i estàndards possibles per a versionar el codi. En el nostre cas es proposarà establir com a conveni l'estàndard SEMVER v 2.0 (*Semantic Versioning*) [43]. Bàsicament es faran servir 3 codis de versió: X.Y.Z (nombres enters, positius) on X és la versió *Major*, Y la versió *Minor* i Z la versió de *patch.* Serà responsabilitat dels programador de seguir correctament aquest model i d'incrementar les diferents versions a mida que s'introdueixen noves funcionalitats/APIs o es solucionen *bugs*. Mes informació sobre l'especificació del l'estàndard es pot consultar a [43].

4.7.4 Plugin Visual Studio GitFlow

En VisualStudio es pot instal·lar un *plugin* que farà d'assistent en la implementació del flux de treball de GitFlow. El *plugin* pot crear automàticament totes les branques i fer els *merges* corresponents un cop tancades les *features*, releases i *hotfixs* en curs. El *plugin* inicialitza el repositori fent servir els estàndards de la nomenclatura que defineix GitFlow.

Tot seguit uns exemples visuals de les opcions que ofereix el *plugin*:

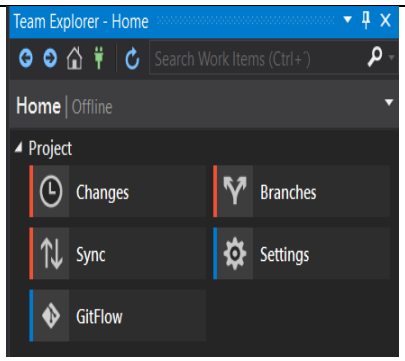
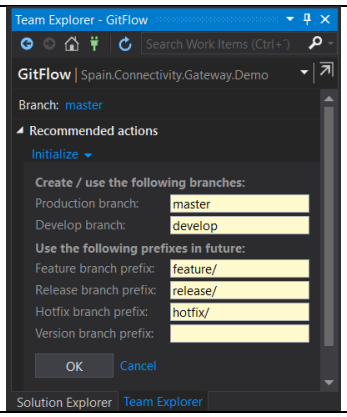
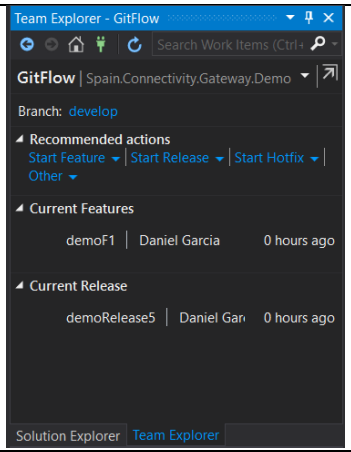
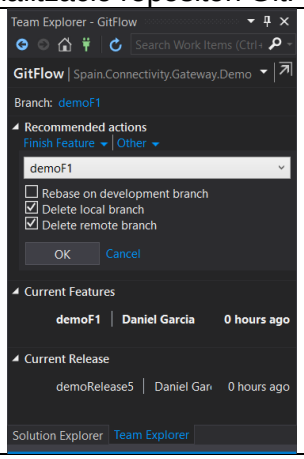
<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Menú d'accés a GitFlow</td><td>Inicialització repositori GitFlow</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Accions de GitFlow</td><td>Finalitzar Feature</td></tr>
</table>

Figura 51: Exemple pantalles plugin GitFlow per a VisualStudio

L´ús d'aquest *plugin* serà obligatori per a facilitar i garantir una unicitat en el procés de treball de tots els programadors. L'organització haurà de prendre les mesures necessàries per a garantir-ho.

4.8 Formació d'equips

La nova plataforma d'Integració Continua afegeix nous recursos, tecnologies i competències dins el domini de l'organització que hauran de ser adquirits per a fer-ne ús i suportar les operacions. La IC serà un canvi transversal que afectarà en major o menor mesura a la majoria d'equips de treball. Tant els comercials de l'empresa com els programadors hauran de conèixer la IC des de diferents punts de vista.

4.8.1 Equip desenvolupadors i caps de projecte

Començant per l'equip de desenvolupadors i pel cap de projecte caldrà organitzar formació i documentació per a tractar els següents punts:

- Aprofundiment gestió Repositoris Git i adopció GITFLOW.
- Implementació específica en Visual Studio.
- Operacions bàsiques amb GitLab.
- Operacions bàsiques amb ProGet.
- Model IC amb Bamboo.
    - Comprensió, arquitectura i funcionament.
    - Entendre quin tipus de projectes i plans es poden configurar.
    - Operacions:
        - Build and deploy on demand.
        - Cerca d'informació històrica.
        - Entendre els logs i la informació mostrada.
    - Com es publiquen les aplicacions als servidors.

Aquestes tasques estaran supervisades pel responsable de l'equip tècnic de desenvolupadors, el cap d'equip i pel responsable del departament de sistemes. Es preveu un durada de formació entre 1 setmana i mitja - 2 setmanes màxim. En aquesta estimació no està considerat el temps necessari per a fer l'estandardització i normalització dels projectes tai i com s'ha explicat en la fase d'anàlisis. Caldrà programar una actuació especifica per a dur a terme aquesta activitat.

4.8.2 Administradors de sistemes

L'equip d'administradors de sistemes és qui haurà d'absorbir més càrrega i formació degut a que s'incorporaran a l'organització noves aplicacions i servidors que caldrà administrar. Caldrà organitzar i fer formació sobre:

- Instal·lació i administració de GitLab.
- Instal·lació i administració de les BBDD PostGreSQL.
- Instal·lació i administració de ProGet.
- Instal·lació i administració de Bamboo. Caldrà aprendre:
    - Administració de projectes i plans.
    - Configuració *plugins* i eines.
    - Investigació de com implementar totes les necessitats particulars de les aplicacions de l'organització i de la seves respectives instal·lacions.
- En general, gestió de les llicències de tot el software.

- Entendre els requisits de comunicacions dels nous servidors i serveis. Caldrà entendre com afegir aquesta nova infraestructura a l'organització, revisar els Firewall locals, de les seus i del DataCenter per a permetre i filtrar adequadament el transit generat per tots els nous serveis.
- En general, entendre i avaluar les necessitats de *backup* dels sistemes de fitxers dels nous servidor, aplicacions i bases de dades.

Es preveu que es necessitarà una formació d'unes 3 setmanes per aprofundir i analitzar correctament tota la documentació i manuals dels nous productes. Aquest procés estarà coordinat i supervisat pel responsable tècnic de l'organització (CTO o equivalent) i el cap de sistemes. Es farà ús també dels serveis de suport dels productes amb llicències que ho suportin.

Els administradors del sistema seran els responsables màxims de la plataforma de IC. Tot i així, s'ha de tenir en compte que les noves eines introduïdes estan força lligades a com és el procés de desenvolupament dels projectes informàtics i com estan estructurats internament. Bamboo, per exemple, necessita saber informació sobre l'estructura del projecte, fitxers de configuració, dades que depenen de cada entorn i en general particularitats de com funcionen o com s'han d'instal·lar en els servidors. Per aquest motiu caldrà també organitzar unes jornades de formació entre els desenvolupadors i administradors. Es parlarà de com programen, com estan fets els projectes, quins recursos externs es fan servir, com han de ser instal·lats, etc. Això facilitarà la feina de com fer una millor i correcta configuració dels plans de *build* i publicació a Bamboo i en definitiva obtindré millors resultats.

4.8.3 Equips Comercials i de Marketing

L'equip comercial i de màrqueting de l'organització ha de conèixer com afecta la IC a la forma de treballar de la companyia i en conseqüència la relació amb els clients. Caldrà fer formació específica per a cobrir els següents aspectes:

- Entendre a nivell general que és la IC.
- Els seus avantatges.
- Els nous temps de lliurament dels productes i/o de noves funcionalitats requerides.
- Els nous entorns de proves que podran fer-se servir per a les *demos* amb els clients.
- La nova flexibilitat per a introduir canvis que poden ser ràpidament publicats als entorns de proves o producció.

D'ara en endavant cal canviar l'estratègia de com es venen i ofereixen els productes als clients. Ara es podran avaluar fàcilment versions no finals del producte, validar paquets de funcionalitats en comptes d'esperar a integracions de producte completes, etc. Caldrà incentivar la cultura de provar amb el client en els entorns de proves i validar i definir amb ells les funcionalitats finals.

La formació pot anar a càrrec de personal tècnic, per exemple un cap de projectes, un cop hagin adquirit tota la formació prèvia tècnica necessària. Es preveu un durada de formació de 3 dies amb *demos* i explicacions.

4.9 Monitoratge

A mida que l'equip de desenvolupadors vagi introduint nous canvis en les aplicacions i les vagi integrant amb la plataforma d'IC, s'aniran publicant en els diferents entorns. Tot i que els canvis que van a producció han passat per un procés d'integració, amb totes les seves respectives validacions, els nous canvis poden causar que l'entorn de producció deixi de funcionar o bé que ho faci de mode incorrecte. Per exemple, es pot donar algun error en el procés de publicació, un *bug* no esperat en el codi, etc. Per aquest motiu es recomana complementar la plataforma de IC amb algun mecanisme de monitorització que sàpiga monitoritzar els servidors de producció, els seus recursos i les seves aplicacions.

Hi ha moltíssimes eines i plataformes al mercat per a monitoritzar servidors. En el nostre cas, per exemple, es proposarà l'eina OpenSource Nagios [44]. Aquesta eina ens pot ajudar a revisar l'estat dels servidors cada cop que es faci una publicació. De forma senzilla es poden revisar paràmetres dels servidors com l'ús de CPU, de memòria, estat d'execució de processos i servei, interfícies de xarxa, connectivitat amb altres servidors, fer ping, etc. Qualsevol publicació que faci disparar algun d'aquests indicadors pot significar que algun problema s'ha introduït en l'última publicació.

De la mateixa forma, es poden monitoritzar URLs de les aplicacions o serveis Web de l'organització per a interpretar si donen error o deixen d'estar accessibles per algun motiu. Amb Nagios, també es poden afegir *plugins* que poden llegir fitxers de logs i trobar patrons dins d'ells que facin disparar alguna alarma. Nagios quan detecti algun problema pot invocar programes des de línia de comandes, pot llençar una petició HTTP a alguna URL per a notificar algun esdeveniment i pot enviar notificacions per correu immediatament.

En resum, sense formar part d'un plataforma de Integració Continua, una bona aplicació de monitoratge ens pot ajudar encara més a garantir la correcta integració i detectar problemes que la plataforma de IC no pot detectar, en la majoria de casos, per si mateixa.

5 Conclusions

En aquest PFC s'ha introduït el concepte de la Integració Continua. S'han descrit les seves característiques principals i s'han comparat a fons 5 productes del mercat, tant comercials com open source. A continuació, amb la informació recollida, s'han valorat aquests productes i s'han definit uns criteris que permetin a una organització triar quina és la millor opció per a les seves necessitats.

A partir d'aquí, per una empresa tipus, s'ha analitzat l'entorn de partida i s'ha triat un producte per a la seva implantació. A la vegada, s'han introduït nous components i productes addicionals com gestors de repositoris de codi font i de paquets Nuget, BBDD i *plugins* que complementaran la funcionalitat del producte d'IC triat. Es descriuen amb detall tots ells així com el procés d'instal·lació, administració i ús de la nova plataforma.

També es proposen tots els canvis a nivell organitzatiu, de procediments i metodologies de treball necessaris per implantar un producte d'aquestes característiques en l'empresa.

Es posa de relleu la importància de les bones pràctiques associades a la disciplina de la IC així com la necessitat de seguir una metodologia de treball clara i comuna per a tot l'equip de desenvolupament. L'actitud dels programadors serà tan important com el bon ús de les eines de la plataforma.

Els beneficis introduïts en l'organització amb la IC comportaran un augment general de la qualitat del codi i de la quantitat de lliuraments. Els mecanismes d'integració automàtics seran capaços de detectar els errors molt abans dins de la fase de desenvolupament. Això significarà un cost menor en temps per a solucionar els errors i més temps útil per a publicar noves funcionalitats. L'avaluació a curt termini de petites funcionalitats de codi serà més eficient que no pas fer-ho per a grans lliuraments que es fan amb poca freqüència.

En definitiva cada cop més, mitjanes i grans organitzacions del sector de les TIC, estan adoptant la pràctica de la IC per a donar suport a les seves demandes de creixement i d'operacions. Serà una inversió per a garantir la sostenibilitat futura del procés productiu de l'organització.

6 Bibliografía

[1] «Grady_Booch,» [En línea]. Available: https://en.wikipedia.org/wiki/Grady_Booch.

[2] «Introducing Continuous Integration,» [En línea]. Available: http://cdn.ttgtmedia.com/searchDataManagement/downloads/Continuous_Integration.pdf.

[3] «Agile Aliance,» [En línea]. Available: https://www.agilealliance.org/glossary/continuous-integration/.

[4] M. F. -. C3. [En línea]. Available: https://www.martinfowler.com/bliki/C3.html.

[5] M. Fowloer. [En línea]. Available: https://www.martinfowler.com/.

[6] M. Fowler, «Continuos Integration,» [En línea]. Available: https://www.martinfowler.com/articles/continuousIntegration.html.

[7] CruiseControl. [En línea]. Available: http://cruisecontrol.sourceforge.net/.

[8] «Impact of Continuous Integration for the Effectiveness of Automated Testing,» [En línea]. Available: https://www.linkedin.com/pulse/impact-continuous-integration-effectiveness-automated-kaluarachchi/.

[9] S. C. Hudson. [En línea]. Available: http://hudson-ci.org/.

[10] M. L. OpenSource. [En línea]. Available: https://opensource.org/licenses/MIT.

[11] D. Syntax. [En línea]. Available: https://jenkins.io/doc/book/pipeline/syntax/.

[12] WYSIWYG. [En línea]. Available: https://en.wikipedia.org/wiki/WYSIWYG.

[13] J. B. O. Editor. [En línea]. Available: https://jenkins.io/blog/2016/05/26/introducing-blue-ocean/.

[14] A. Tomcat. [En línea]. Available: http://tomcat.apache.org/.

[15] Docker. [En línea]. Available: https://www.docker.com/.

[16] K. K. B. V. Machine. [En línea]. Available: https://www.linux-kvm.org/page/Main_Page.

[17] J. Plugins. [En línea]. Available: https://plugins.jenkins.io/.

[18] W. Jenkins. [En línea]. Available: https://wiki.jenkins.io/collector/pages.action?key=JENKINS.

[19] J. Jenkins. [En línea]. Available: https://issues.jenkins-ci.org/projects/JENKINS/issues.

[20] J. Documentation. [En línea]. Available: https://jenkins.io/doc/.

[21] Jira. [En línea]. Available: https://www.atlassian.com/software/jira.

[22] Confluence. [En línea]. Available: https://www.atlassian.com/software/confluence.

[23] B. M. &. Documentation. [En línea]. Available: https://confluence.atlassian.com/bamboo/bamboo-documentation-289276551.html.

[24] A. Bbitbucket. [En línea]. Available: https://bitbucket.org/.

[25] «JetBrains,» [En línea]. Available: https://www.jetbrains.com/.

[26] «Team City - Runners support,» [En línea]. Available: https://confluence.jetbrains.com/display/TCD10/Build+Runner.

[27] «TeamCity Platform support documentation,» [En línea]. Available: https://confluence.jetbrains.com/display/TCD10/Supported+Platforms+and+Environments#SupportedPlatformsandEnvironments-Platforms(OperatingSystems).

[28] «Repositori CircleCI imatges Docker,» [En línea]. Available: https://circleci.com/docs/2.0/circleci-images/.

[29] «GO CD,» [En línea]. Available: https://www.gocd.org/.

[30] «GoCD plugins,» [En línea]. Available: https://www.gocd.org/plugins/.

[31] «Versions i dependencies .NET Framework,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/bb822049(v=vs.110).aspx.

[32] «Paquets NuGet,» [En línea]. Available: https://docs.microsoft.com/en-us/nuget/what-is-nuget.

[33] «Gestor Nuget ProGet,» [En línea]. Available: https://inedo.com/proget.

[34] «GitLab,» [En línea]. Available: https://about.gitlab.com/.

[35] «GitLab, Installation support,» [En línea]. Available: https://about.gitlab.com/installation/.

[36] «GitLab , system requirements,» [En línea]. Available: https://docs.gitlab.com/ce/install/requirements.html.

[37] «Git per a Windows,» [En línea]. Available: https://git-scm.com/download/win.

[38] «PostgreSQL,» [En línea]. Available: https://www.postgresql.org/.

[39] «Wix Tooset for VisualStudio,» [En línea]. Available: https://marketplace.visualstudio.com/items?itemName=RobMensching.WixToolsetVisualStudio2017Extension.

[40] «Web Deployment Tool,» [En línea]. Available: https://technet.microsoft.com/en-us/library/dd569058(v=ws.10).aspx.

[41] «Windows Package Manager Chocolatey,» [En línea]. Available: https://chocolatey.org/.

[42] V. Driessen, «Git Flow branching model,» [En línea]. Available: http://nvie.com/posts/a-successful-git-branching-model/.

[43] «Semantic Versioning,» [En línea]. Available: https://semver.org/.

[44] «Nagios Monitoring,» [En línea]. Available: https://www.nagios.com/.