Universitat Oberta de Catalunya

uoc.edu

Reptes en DevSecOps

Raúl Suárez Dabó
Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions
M1.849 Seguridad empresarial_MISTIC

Jordi Guijarro Olivares
Victor Garcia Font

26/12/2020

FITXA DEL TREBALL FINAL

Títol del treball:	*Reptes en DevSecOps*
Nom de l'autor:	*Raúl Suárez Dabó*
Nom del consultor/a:	*Jordi Guijarro Olivares*
Nom del PRA:	*Víctor García Font*
Data de lliurament:	*12/2020*
Titulació o programa:	*Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions*
Àrea del Treball Final:	*M1.749 - TFM-Seguretat empresarial*
Idioma del treball:	*Català*
Paraules clau	*DevSecOp MultiCloud CICD*

Resum del Treball:

Aquest treball surt de la demanda de garantir seguretat en els processos de creació de les aplicacions dins d'un entorn *Multi Cloud*.

L'objectiu principal es determinar quines son les pràctiques més adequades que garanteixi una aplicació segura. Aquesta seguretat vincula dos factors. El primer, és garantir una aplicació que no tingui cap vulnerabilitat explotable. El segón, és adequar els procesos automatitzats d'entrega en un context d'entrega o actualització oportú, correcte i sense fer malbé el Sistema de la Informació.

En aquest treball s'ha analitzat les diverses operatives vinculades a un *DevOp* i s'ha desenvolupat el grau d'acceptació desde un punt de vista de seguretat. S'ha realitzat un ventall d'operatives vinculades a la qualitat de l'aplicació, en termes de seguretat, i s'ha remarcat la importància de realitzar una entrega amb garanties a un context *Multi Cloud*.

La conclusió a la que s'arriba és la importància de la seguretat en els processos de creació i entrega de les aplicacions. La cultura *DevOp* moltes vegades es queda a la superficie en termes de seguretat. Per aquest motiu, s'emfatitza una cultura *DevSecOp* que vincula el desenvolupament i les operatives amb la seguretat com a pont.

Abstract:

This work comes from the demand to ensure security in the application creation processes within the Multi Cloud environment.

The main goal is to determine the best practices to ensure a secure application. This security is based on two points. The first one, is to ensure an application that it hasn't any exploitable vulnerabilities. The second one, is to adapt the automated delivery processes in a timely, correct and undamaged context of delivery or updating of the Information System.

This work has analyzed some aspects related to *DevOp* and it has evaluated the security level. A range of operations related to the quality of the application, in terms of security, and the importance of making a delivery with guarantees in a Multi Cloud context has been remarked.

The conclusion has been arrived at is the importance of security at creation and delivering application's processes. The *DevOp* culture often stays on the surface in terms of security. For this reason, a *DevSecOp* culture emphasize that and links development and operations with security as a bridge.

Índex

1. Introducció

1.1 Context i justificació del Treball

El constant increment de demanda d'aplicacions que interactuen en diferents processos de negoci, a profilerat la necessitat de creació de serveis crítics amb garanties. El poder-ne oferir aquests serveis dins d'uns marges de temps adequats, requereix una serie de factors que donin la possibilitat de respondre a aquesta necessitat.

Els desenvolupament àgil (*Agile Software Development*[1]) va ser una de les respostes a aquesta proactivitat. Gràcies al seu *The Manifesto for Agile Software Development*[2] s'ha fet un dels canvis de paradigma més importants del desenvolupament de *software*. La seva metodologia es basa en el desenvolupament iteratiu i incremental, els requisits evolucionen al llarg del temps, fent una estrategia basada en l'adaptació i en la entrega a curt termini.

L'altre factor va ser el canvi de paradigma al disposar de la possibilitat d'abstracció a dos nivells.

Com primer nivell va aparèixer la computació virtual, ofereix la creació de recursos de computació més flexibles. Els proveïdors ofereixen les seves màquines virtualitzades -recolzades en els seus centres de processaments (CPDs)- implantant una política de pagament per ús.

Com según nivell d'abstracció van ser els contenidors, aquests es basen en una virtualització a nivell de Sistema Operatiu[3]. Ofereix una encapsulació que permet: portabilitat, lleugeresa i autosuficiencia. Portabilitat, al emprar integració amb el *kernel*[4] de *Linux*. Lleugeresa, al integrar les funcionalitats estrictament necessàries per executar l'aplicació i no un SO complet com succeeix en la virtualització. I autosuficiencia, al integrar totes les llibreries, configuracions i arxius necessaris per la seva execució. Aquest nivell d'abstracció permet a les empreses abstreures i executar les seves aplicacions sobre un *Platform as a Service*[5]. *PaaS* és un producte fruït de l'abstracció de la gestió de la infraestructura. On el proveïdor s'encarrega de mantenir la plataforma on es desplega els contenidors i el client pot gestionar els recursos segons les seves necessitats.

Encara que les bondats del *PaaS* són moltes, hi ha un problema. Vincular els Sistemes de la Informació (SI) a un proveïdor genera una dependencia condicionada a la gestió de les polítiques del proveïdor, impactant negativament en les empreses segons la decisió presa. Per aquest motiu les empreses s'estan acollint a solucions mixtes, conceptes com *Hybrid Cloud*[6] o *Multi Cloud*[7] permeten la migració o replicació de les aplicacions a un proveïdor

[1] https://en.wikipedia.org/wiki/Agile_software_development
[2] https://agilemanifesto.org/iso/ca/manifesto.html
[3] https://en.wikipedia.org/wiki/OS-level_virtualization
[4] https://en.wikipedia.org/wiki/Kernel_(operating_system)
[5] https://en.wikipedia.org/wiki/Platform_as_a_service
[6] https://en.wikipedia.org/wiki/Cloud_computing#Hybrid_cloud
[7] https://en.wikipedia.org/wiki/Multicloud

o a un altre. Això és possible al existir una homogeneïtzació de productes *PaaS* -el GKE de *Google* o AWS són un exemple- on tenim el *cluster* de servidors amb un orquestador instal·lat, *Kubernetes*.

Un cop identificat el desenvolupament actual, estem en l'obligació de contextualitzar la realitat en la que ens trobem avui dia.

Durant aquest any -2020- hem estat presents en la digitalització empresarial forçada per l'aparició del COVID-19[8]. Per tal de poder-ne mantenir el ritme de producció, les empreses han flexibilitat els llocs de treball del seus empleats de l'oficina a la seva casa. Això va implicar un augment significatiu en l'ús d'aplicacions remotes a través de la xarxa, internet.

Segons *IBM* en aquest 2020 tenim un increment del 40% en atacs en el primer trimestre[9], si indaguem més, trobem que Espanya està entre els sis països més atacats[10].

Com conseqüència de tot l'esmentat s'evidencia la criticitat dels procesos que busquen la conciliació entre la qualitat i l'entrega. *Continuous Integration*[11] i *Continuous delivery*[12] són les tècniques emprades que permeten la verificació del codi -mitjançant tècniques basades en *testing*- per garantir el funcionament de l'aplicació i automatitzar l'entrega del producte en l'entorn adient en el menor temps possible.

1.2 Objectius del Treball

L'objectiu d'aquest treball principal es estudiar la realitat per un *DevSecOp* dins un entorn *Multi Cloud*. Per poder-ne assolir aquest objectiu, es necessari distingir dues fites.

Per un costat, és necessari identificar els mètodes i eines capaços de donar una verificació de seguretat de cara a la custodia automatitzada per part d'un *DevSecOp*.

Per l'altre costat, identificar i estudiar les implicacions de seguretat que un *DevSecOp* ha d'afrontar dins d'un marc de *PaaS* multi-proveïdor. Per aquest motiu, aquest treball està fortament fitat en el cor de la cultura *DevOp*, les operatives CI/CD.

[8] https://es.wikipedia.org/wiki/COVID-19
[9] https://es.newsroom.ibm.com/announcements?item=122574
[10] https://cso.computerworld.es/cibercrimen/espana-entre-los-seis-paises-mas-ciberatacados-en-los-ultimos-dos-meses
[11] https://en.wikipedia.org/wiki/Continuous_integration
[12] https://en.wikipedia.org/wiki/Continuous_delivery

1.3 Enfocament i mètode seguit

L'enfocament i la metodología a emprar en aquest projecte comença desde un punt teòric fins arribar a un més pràctic.

La idea és començar desde la base, analitzar filosofía *DevOp*, una vegada definida es realitzarà un cerca de tècniques i eines que es pot emprar. Es donarà molta presència a les eines d'origen *Open-source,* el motiu serà -principalment- per la facilitat d'accés -tant a l'eina com a la documentació relacionada-.

Arribat a aquest punt, es tornarà a una vessant més teórica, identificar posibles punts on es pot qualificar com filosofía *DevSecOp*. A partir d'aquí, es tornarà a realitzar una cerca d'eines -amb preferència per les *open source*- amb les bondats de cadascuna. En aquest punt hi és possible realitzar un anàlisis comparatiu per tal d'identificar la més adient segons el context.

En la següent fase, s'estudiarà la informació *PaaS* dels diferents *Clouds* disponibles al seus recursos oficials. Amb aquesta informació s'analitzarà les implicacions tècniques i la idoneïtat de estar present la figura del *DevSecOp* i quines decisions de seguretat l'hi pertany en una gestió *Multi Cloud* com tendencia actual.

1.4 Planificació del Treball

La planificació d'aquest projecte disposa de tres fases. *DevOp* que es i que s'espera dins la filosofía. Valor afegit que aporta un *DevSecOp* i, finalment, com afecta una cultura *DevSecOp* dins d'un context *Multi Cloud*.

Amb tot això podem identificar els següents punts cronològics a aplicar durant el present projecte:

- Anàlisis *DevOp*.
  - Identificació de les operatives d'un *DevOp*.
  - Eines disponibles.
  - Poc de les metodologies més comunes.
- *DevOp* enfocat en seguretat, DevSecOp.
  - Identificar els punts on una filosofía *DevSecOp* es més idonea.
  - Analitzar eines disponibles per els punts identificats.
  - Poc de les metodologies ideals en un context de seguretat.
- *PaaS* en un entorn *Multi Cloud*.
  - Identificació dels proveïdors principals de *PaaS*.
  - Implicacions de gestionar la seguretat en un entorn *Multi Cloud*.

Per poder-ne assolir aquesta planificació s'emprarà informació bibliogràfica pertinent però jugarà un paper molt important els recursos disponibles a la xarxa. En especial la documentació de diferents productes, eines i *clouds* a analitzar.

1.5 Breu sumari de productes obtinguts

A aquest treball s'han obtingut els següents productes:

- L'automatització de proves estàtiques de seguretat.
- L'automatització de proves dinàmiques de seguretat
- L'automatització d'anàlisis de llibreries emprades a l'aplicació.
- L'automatització d'anàlisis d'imatges *Docker* que encapsulen l'aplicació.
- L'automatització de *Pentesting* de l'aplicació generada.
- *Delivery* sota un marc d'entrega segur.

1.6 Breu descripció dels altres capítols de la memòria

A aquest treball està desenvolupat en tres capítols:

- Capítol 2, s'analitza l'origen i la definició d'un *DevOp,* la seva vinculació amb conceptes com *GitOps* i quines són les operatives més comunes realitzades en l'actualitat.
- Capítol 3, és qüestiona la realitat d'un *DevOp* i es realitza una aproximació dels principis de seguretat que hauria de respondre i quines carències hi presenta en front a un *DevSecOp*. Finalment s'analitza quines són les operatives que hi hauria d'integrar un CI per considerar-se un CI que cerca la seguretat de les seves aplicacions.
- Capítol 4, s'introdueix la realitat del mercat i es fa un anàlisis de les dues opcions més adequades per un CD a un entorn *Multi Cloud*. S'evidencia la necessitat de disposar d'una entrega amb garanties i es realitza un anàlisis, explicació i aproximació d'una possible solució.

2. DevOp

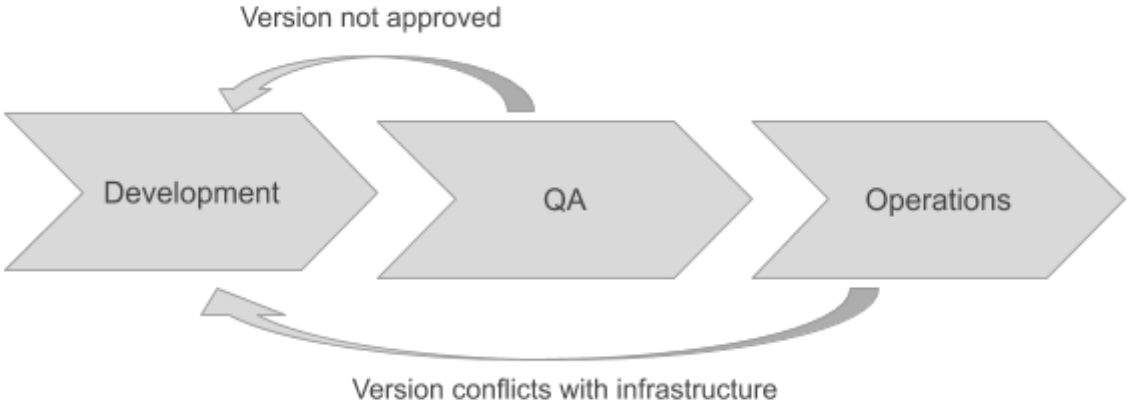
Abans de perfilar el concepte de *DevOp* estem en l'obligació d'aprofundir en ell a nivell històric i així entendre la amplitud d'aquest moviment.

Tradicionalment les empreses dividien el cicle de desenvolupament d'un producte en tres equips. L'equip de desenvolupament, l'equip de qualitat (QA) i l'equip d'operacions:

- L'equip de desenvolupament (*development*) era l'encarregat d'introduir noves característiques al producte.
- L'equip de qualitat (QA) era el responsable de garantir el correcte funcionament de la versió entregada per l'equip de desenvolupament.
- L'equip d'operacions (*Operations*) era qui coordinaba el desplegament de l'aplicació validada per QA.

Si parlem en termes de prioritats dels diferents equips, podem diferenciar dos. Per un costat tenim a l'equip de desenvolupament, aquest té com objectiu principal, entregar la versió demanada per negoci en el temps pactat. Per l'altre, l'equip d'operacions té la prioritat d'estabilitat, garantir el funcionament del servei per els clients.

En conseqüència, es pot denotar una falta de col·laboració entre els equips al disposar d'interessos diferents. Per ser més il·lustratius farem un exercici recolçant-nos en el diagrama d'acontinuació.



En una situació ideal, quan un desenvolupament es finalitza, pasa a ser validat per QA i, un cop verificat, passa a ser desplegat per l'equip d'operacions a producció. Aquesta situació deixar de ser tan ideal quan un desenvolupament, en la fase de proves, és detecta un *bug* i es reporta a desenvolupament perquè ho arregli. Aquest últims revisen la versió i fan els canvis necessaris per tornar al cicle original. Encara que aquesta situació pot endarrerir el flux ens podem trobar una situació més extrema.

Imaginem un desenvolupament realitzat i verificat per l'equip de QA hi arriba el moment de desplegar i dona problemes. Aquesta versió és rebutjada per operacions i retornada a desenvolupament -amb el conseqüent *rollback* a

producció-. L'equip de desenvolupament -al no disposar de contexte de la infraestructura- revisa la versió i indica que no hi troba cap tipus de problema en el lliurable. El conflicte està servit, ens trobem amb desenvolupament que dona per terminada la seva labor -prioritza el *delivery*- que hi choca amb l'equip d'operacions que es nega a posar en marxa aquesta versió -prioritza la estabilitat-.

Dins d'aquesta situació el consultor independent en IT, Patrick Debois, va realitzar una ponencia a la *Agile 2008* de Toronto. En ella va emfatitzar la frustració que hi sentía com a conseqüència de la divisió entre el desenvolupament i les operatives. Hi va proposar -el que ell va denominar- "Agile Infrastructure". Aquesta idea, va interesar a Andrew Shafer i va començar a treballar juntament amb Debois.

A l'any 2009 John Allspaw i Paul Hammond van fer una ponència a la *Velocity Conference* que s'anomenaba "10+Deploys Per Day: Dev and Ops Cooperation at Flickr", en ella enfatitzen la coordinació entre els dos equips per poder-ne realitzar desplegaments a producció constantment motivat per la necessitat de ser més àgils. Patrick Debois seguia l'event per *streaming* i començar un debat vía *Twitter*.

Entre el 30 i 31 d'octubre de 2009 Patrick Debois va coordinar la primera conferència "DevOpsDays" a Ghent, Bèlgica. Fent que el moviment *DevOps* guayes força sent un debat continu i orgànic.

2.1 Qué es un DevOp?

Abans d'entrar en matèria, és molt important aclarir certes confusions implantades per terces que no s'acaben d'entendre completament el concepte *DevOp*.

DevOp és una cultura que busca la cooperació entre el desenvolupament i les operacions. Busca l'equilibri entre l'entrega i l'estabilitat, fent que tots hi formint part del mateix equip. En definitiva, implantar una cultura *DevOp* implica buscar:

- *Fas time-to-market* (TTM).
- Poques fallades a producció.
- Alta capacitat de recuperació davant d'una fallada.

Per poder-ne assolir aquests punts s'ha de demolir pràctiques tradicionals com:

- Caixes negres, és molt important evitar la ofuscació d'informació per fomentar la col·laboració.
- Processos farragosos, un procés farragós és un procés lent que implica baixa entrega.
- Poca automatització, una baixa automatització en els processos de compilació i desplegaments implica baixa consistència.
- Poca capacitat d'identificar i fitxar errors, poca capacitat de reacció davant problemes i dona com conseqüència un baix coneixement.

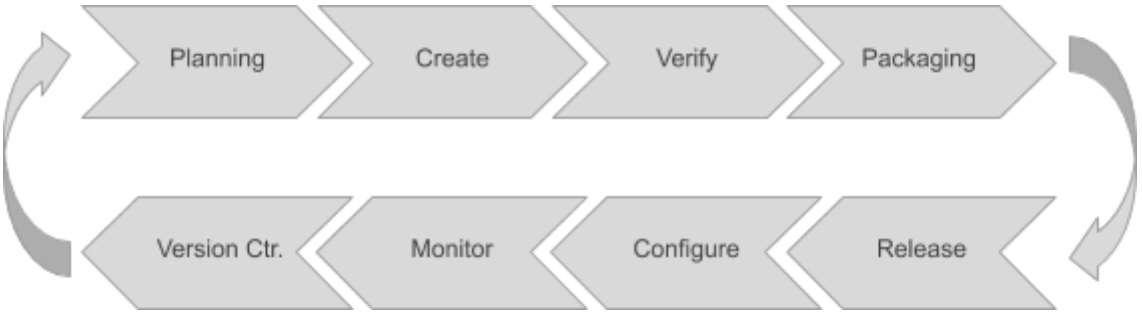
Com resum es pot dir que la cultura *DevOp* es basa en incentivar la proactivitat de l'equip, fent-lo sentir part d'un tot i no una part.

2.2 La toolchain d'un DevOp

Quan parlem de cultura *DevOp* estem parlant d'una cultura multidisciplinar i per això està composta per diverses disciplines.

Per definició, ser *DevOp* comporta un conjunt de capacitats que s'han de recolzar en un conjunt d'eines, una *toolchain*.

Un *DevOp toolchain*[13] és una combinació d'eines programatives que tenen com objectiu facilitar el desenvolupament, l'entrega i la gestió de l'aplicació. D'acord amb això una aplicació disposa de les següents fases de vida.



- *Planning*: ès la fase que resum la captació de requisits i la definició de l'estratègia a prendre per aplicar-los al *software*. Per definir l'estratègia s'ha de combinar el treball entre els *product owners*, l'equip de desenvolupament, l'equip de seguretat i els responsables de la gestió de la infraestructura.
- *Create*: fase en la que es comença el desenvolupament de l'aplicació codificant un conjunt de *release candidates* que seran cada cop més pròxima a la versió definitiva.
- *Verify*: punt en el que es realitza les comprovacions de qualitat del producte. Per poder-ne dur a terme aquesta feina es realitza automatització de testeig, anàlisi de codi estàtic, *smoking test* o anàlisi de seguretat.
- *Packaging*: fa referència al moment en que la versió està llesta per ser desplegada, normalment a entorn de proves com *staging* o *pre production*. És molt freqüent que aquest procés es dispara mitjançant l'acceptació d'un *pull request*.
- *Release*: és el punt on es planifica l'orquestració, aprovisionament i desplegament de la versió al entorn objectiu -normalment producció-.
- *Configure*: és el conjunt d'automatismes complementaris perquè l'aplicació disposi de les configuracions i serveis necessaris. Normalment es defineix com *Infrastructure as Code*, on es deriva un conjunt d'operatives vinculades a l'aplicació.
- *Monitor*: la capacitat de monitoritzar l'aplicació per tal de poder-ne garantir proactivitat d'avant les necessitats específiques es un dels punts

[13] https://en.wikipedia.org/wiki/DevOps_toolchain

crucials en el cicle de vida. Definir quins són i com impacta es una feina complexa que requereix temps.
- *Version Control*: és la gestió de la recol·lecció de informació vinculada a producte. Fent especial èmfasi en el control de les versions, el qual permet realitzar un desenvolupament no lineal i distribuït.

Aquestes vuit fases son cícliques, de manera que la gestió del projecte és incremental i adaptativa segons les necessitats de negoci.

2.3 DevOps vs. CI/CD

La Integració Contínua (CI) és la pràctica de la ingeniería del software que realitza les integracions automàtiques. Quan parlem d'integracions automàtiques estem parlant del procés de compilació i execució de proves del projecte extret d'un control de versions. L'objectiu d'aquest procés es detectar possibles problemes de forma autònoma i ràpida del treball dels desenvolupadors.

El Desplegament Continu (CD) és un conjunt de processos automàtics que permeten la entrega continuada del *software* en el menor temps possible un cop la versió està terminada o fitada.

La CI i CD són part en el desenvolupament de la cultura *DevOp*, és molt freqüent confondre CI/CD amb *DevOp* però la relació no és exactament un a un.

DevOp es un moviment cultural que sobrepasa els àmbits de aquestes dues pràctiques de desenvolupament de *software*. És cert que la integració continuada i el desplegament continu són parts molt important en la cultura, però no és un tot, sino una part.

La filosofía *DevOp* es basa en tres grans blocs: *Continuous Feedback*, *Continuous Integration* i *Continuous Deployment*. Dins d'aquest tres grans blocs podem agrupar el conjunt de vuit fases numerades en l'apartat anterior de la següent manera:

Continuous Feedback	*Continuous Integration*	*Continuous Deployment*
Planning Monitor Version Control	Create Verify	Packaging Release Configure

Arrel d'aquest quadre resum podem determinar que:

- CF (*Continuous Feedback*) és el punt de la reciprocitat per terceres parts del *software* desenvolupat. On la planificació és el punt crucial, ja que la coordinació de tots està fortament relacionada amb la monitorització -captació de comportaments i possibles errors- i la documentació l'aplicació-.
- CI (*Continuous Integration*) és el procés de creació i verificació automatitzat dins del cicle de vida d'una aplicació.

- CD (*Continuous Deployment*) és el conjunt d'automatitzacions que poden desplegar la versió pertinent a l'entorn adequat amb les seves configuracions.

2.4 Eines matrius d'un DevOp

Un cop que hem explicat que el procés CI/CD és part de la cultura *DevOp*, farem un petit repàs del conjunt d'eines disponibles al mercat que cobreixen aquesta operativa.

És fàcil trobar a la xarxa una taula comparativa[14] del conjunt de diferents eines que hi ha disponibles per realitzar els processos de CI/CD. Dins de les moltes opcions hi podem destacar dos grans conjunts de solucions que hi disposem al mercat.

Per un costat tenim eines basades en *SaaS*, ens permet cobrir les necessitats més comunes d'un *DevOp* sense realitzar processos d'instal·lació i contractació de màquines, simplifica la operativa al tractar-se d'un servei. Dins d'aquest conjunt ens trobem solucions com Circle CI o Travis CI.

Per l'altre costat disposem d'opcions que requereixen la contractació, instal·lació i configuració de l'eina per poder-ne utilitzar-la. Aquesta alternativa requereix un esforç extra al tindre que configurar les màquines i l'eina. Dins d'aquest conjunt ens trobem amb l'eina referència dins del sector, Jenkins.

Jenkins és la continuació del projecte *Hudson*[15] *d'Oracle,* és l'eina més extesa. Un dels motius principals és la seva arquitectura escalable i la seva alta capacitat d'integració de complements. Al tractar-se d'un projecte *Open Source* el suport i col·laboració de la comunitat és molt ampli. Fent que *Jenkins* sigui un estàndard i referent per aquestes operatives.

Encara que hem esmentat les eines principals que defineixen una de les parts fonamentals d'un *DevOp,* no són les úniques. Podem ampliar el conjunt d'eines segons les necessitats de cada projecte. Principalment necessitem eines que facilitin els tres grans blocs explicats en l'apartat anterior.

Si ens focalitzem en la part de *continuous feedback* podem parlar d'eines com pot ser *Jira* per la gestió de tasques i versionat. Si parlem de monitorització de l'aplicació podem optar per *Prometheus* amb *Grafana*. Sempre i quan no optem per solucions *cloud* com pot ser *Cloud Monitoring* o *Cloud Logging* de *Google*, per exemple.

Si continuem l'enfoc dels tres grans blocs veiem que *Continuous Integration* i *Continuous Deployment* engloben gran part de la operativa.

La *suite* d'eines que utilitza un *DevOp* és amplia i diferent, no és l'objectiu d'aquest projecte realitzar-ne un anàlisi del conjunt de solucions. El nostre objectiu és analitzar certs aspectes que desenvoluparem en l'apartat tercer. Per aquest motiu centrarem un esforç més gran en analitzar les operatives

[14] https://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software
[15] https://en.wikipedia.org/wiki/Jenkins_(software)

vinculades a la CI/CD ja que tenen una especial relevancia en l'àmbit de la seguretat.

2.5 GitOps i la definició operacional

L'any 2017 Alexis Richardson va publicar un article en el que parla d'un nou concepte, *GitOps*.

En el seu article va emfatitzar que part de les bones pràctiques que hi estava realitzant el seu equip era emprar fitxers descriptors on es reflectia les operatives a realitzar. Aquestes s'integren en els fluxos del gestor de versions.

Aquesta manera d'entendre la gestió operacional va influir la forma d'administrar les operatives dins de la filosofía *DevOp*. Convertint-se en una pràctica molt comuna la gestió dels fitxers semàntics -normalment fitxers en format *yaml* o *json*- per determinar els automatismes a aplicar. En especial dins del flux CI/CD que hem explicat als punts anteriors.

Encara que a l'article de Alexis Richardson no es fa una referència explícita, *GitOps* es una adaptació de les operatives a la gestió de branques de Vincent Driessen a l'any 2010, el *Gitflow*.

La filosofía *DevOp* integra el desenvolupament i les operatives. De manera que els integrants d'aquesta cultura son desenvolupadors. Els desenvolupadors han adoptat *Git* com el control de versions de referència. És una part molt important del seu treball. Per aquest motiu, el concepte *GitOp* es una iniciativa orgànica dins de la cultura *DevOp*, l'adaptació a fitxers semàntics que declaren la gestió de les operatives a realitzar faciliten la comprensió i la gestió d'aquest amb el control de versions dins del flux de desenvolupament proposat per Vincent Driessen.

2.6 Exemple pràctic d'operatives

Dins de l'apartat d'annex d'aquest treball s'adjunta l'explicació d'una POC basada en tecnología *Java*. La idea és mostrar un petit exemple de les operatives més comunes d'un DevOp com punt d'anàlisi inicial.

Hem de diferenciar que s'ha realitzat dues adaptacions. Per un costat, s'ha aplicat les operatives més típiques d'un DevOp emprant un *SaaS* com es *Travis CI*,. i per l'altre costat, s'ha realitzat una operativa idèntica per *Jenkins.* El motiu d'haver-ne escollit aquestes dues opcions és per ser eines de referència dins del sector.

Les operatives aplicades sobre les dues eines es la mateixa:

- Descarrega de dependències.
- Tests unitaris.
- Test d'integració.
- Publicació de la *Release Candidate*.
- Publicació de la versió.
- Deploy, es delegar a l'últim apartat.

Cal remarcar que la realització dels test unitaris s'ha realitzat amb *JUnit* emprant *Jacoco* com eina de *reporting* de cobertura. A més, els test d'integració s'ha realitzat sobre una la mateixa màquina virtual de *Java* per simplificar el flux. També s'ha de tenir present que per el *Docker Registry* s'ha optat per Docker Hub. I, finalment, el desplegament es delega a pròximes fases d'anàlisis dins d'una coyuntura *Multi Cloud*.

Si ens fixem en els fitxers semàntics de *Travis CI* i *Jenkins* són molt semblants. Tots dos es basen en *stages* que permeten condicionar la seva execució depenent de la branca que s'està executant, també permeten invocar comandes *shell* per interactuar amb el sistema operatiu *host* i, finalment, disposa d'un conjunt de variables d'entorn per defecte, a la qual es pot afegir les que sigui necessària per l'operativa.

Si per el contrari volem parlar de les diferencies de treballar amb *Jenkins* i *Travis CI*, podem resaltar un conjunt a analitzar molt interessant.

Si parlem dels temps de *setup* hi ha una gran distinció. L'experiència d'usuari de *Travis CI* permet ser molt més productiu que no pas amb *Jenkins*. El motiu és l'enfoc, *Travis CI* està pensat per un conjunt de casuístiques molt predeterminat. Parteix de la premisa d'integrar-se amb un conjunt de *VC,* disposa d'un conjunt de *workflows* que integren les operatives més comunes per un grup de llenguatges de programació. Permet centrar-se en definir les operatives mitjançant el fitxer semàntic i delegar tasques com la definició de *WebHook* i *tokens* d'accéss al servei.

Per la banda contraria tenim *Jenkins,* és una eina que ofereix un gran conjunt d'opcions. Motiu per el qual fa el procés de *setup,* molt més complicat. Desde la configuració del servidor, que allotja el servei, fins la seva instal·lació, i passant per la seva configuració, fa que l'operativa d'aquesta sigui molt més exigent i requereix planificació i administració.

Com hem esmentat *Jenkins* es una eina que ofereix un gran conjunt d'opcions, aquest és el seu gran reclam. Permet realitzar tot tipus d'operatives amb restrincions molt petites. Aquest és el punt on *Travis CI* manca. Si recapitulem en la semàntica d'operacions declarades al seu fitxer. Veiem que l'allotjament dels informes de cobertura de testeig depèn d'una tercera eina, Codecov. Aquest fet comporta cert descontrol de la informació, la informació no està disposada a un sol proveïdor sino que està distribuïda en diversos, motiu per el qual gener cert desconeixement de qui és responsable de que.

Un altre aspecte a tenir present és mantenir una informació fluida, la gestió d'enviament de correus informatius sobre els diferents processos automàtics, requereix cert esforç per tots dos serveis. És cert que *Travis CI* sols necessita la declaració de la política d'enviament, per el contrari *Jenkins* requereix un esforç extra al tindre que integrar-lo amb un servei d'enviament de correus i gestionar la política d'enviament.

Com conclusió inicial podem afirmar que *Jenkins* es una eina multidisciplinar que permet molta flexió a l'hora de gestionar les operatives dels nostres projectes. Permet delegar a terces les responsabilitat si l'equip ho considera necessari, de igual manera que permet centralitzar-la.

Per el contrari, és cert que requereix un esforç extra, la seva configuració i manteniment obliga a administrar un servei. Si no es vol o no requereix gran flexibilitat per la manca de requisits tècnics, serveis com *Travis CI* es una solució accesible i àgil que permet una primera aproximació sense un gran esforç.

3. DevSecOp

En l'apartat anterior hem fet un repàs del conjunt de disciplines que agrupen la cultura *DevOp*. Encara que aquesta cultura agrupa un conjunt molt dispar d'inquietuds, moltes vegades s'obvia un aspecte crític de les aplicacions que s'estan desenvolupant, la seguretat.

La seguretat informàtica, o ciberseguretat, es l'àrea vinculada a la protecció del conjunt d'actius, el qual es pot resumir en:

- La **infraestructura computacional**, vigila el correcte funcionament dels equips i actuar davant qualsevol tipus d'amenaça.
- Els **usuaris**, consumidors de la tecnologia facilitada als que se'ls ha de garantir idoneïtat en la informació gestionada.
- La **informació**, actiu principal gestionat per la infraestructura computacional i consumida per l'usuari.

Un cop definits el conjunt d'actius que són susceptibles de ser amenaçats per terceres parts podem passar a parlar d'amenaces.

Una amenaça pot sorgir en diferents moments del cicle de vida d'una aplicació, un defecte en el disseny o implementació pot donar conseqüències desastroses. No obstant això, no són els únics condicionals que poden identificar-se com un conjunt de possibles amenaces a tindre present:

- **Usuaris**, són el problema més gran a l'hora de gestionar la seguretat. Les seves decisions poden causar problemes de seguretat per una manca de perfilat en els seus permisos o un comportament inesperat.
- ***Malware***, són aplicacions destinades a realitzar accions il·lícites dels recursos del sistema informàtic.
- **Errors de programació**, són la porta d'accés a *exploits* per accedir al sistema informàtic objectiu. Pot estar en una aplicació o al pròpi sistema operatiu.
- **Intrusos**, accessos no autoritzats a les dades o aplicacions allotjades al sistema informàtic.
- Un **sinistre**, pèrdua de material o arxius.
- **Personal tècnic**, conjunt de treballadors de l'entitat que poden tenir desavinences amb altres individus dins del mateix entorn.
- **Fallades electròniques**, els equips són dispositius electrònics que poden tenir errors fruit de l'ús o del seu pròpi disseny.

Dins d'un ecosistema en que la integració amb tercers i l'accessibilitat dels sistemes de la informació es fa mitjançant internet. És fàcil arribar a la conclusió que el teu sistema de la informació serà atacat- en major o menor mesura-, segons la importància que representa la informació gestionada per els teus aplicatius a terceres parts.

Davant aquesta situació, s'ha determinat un conjunt d'accions que tracten de minimitzar el risc de vulnerabilitat davant el conjunt d'amenaces exposats.

- *Security by design*, tota aplicació ha d'estar dissenyada desde el seu principi sota una premisa de ser segura.

Les tècniques vinculades a aproximar aquest principi són:

- *Principle of least privilege*[16] (PoLP), consisteix en donar -estrictament- accés als recursos necessaris per a cada part del sistema funcional.
- *Automated theorem proving*[17] (ATP), teoremes automatitzats que demostren el correcte funcionament del programari realitzat.
- *Code reviews*, tècnica que consisteix en la revisió del codi per integrants de l'equip cercant una bona qualitat del mateix.
- *Unit Testing*, proves unitàries que verifiquen el correcte funcionament d'un mòdul específic.
- *Defense in depth*, pràctica informàtica que consisteix en aplicar múltiples capes de control de seguretat a un sistema de la informació.
- *fail safe*, davant fallades l'aplicació ha de garantir un comportament segur.
- *Audit trails*, sistema de control d'activitat que analitza i reacciona davant una bretxa de seguretat.
- *Full Disclosure* o auditoria de seguretat independent, conjunt de persones que cerquen possibles errors i que no disposen cap relació amb l'equip que l'ha desenvolupat.

- *Security architecture*, una arquitectura segura es pot definir com un conjunt d'artefactes dissenyats en descriure els controls de seguretat. Aquest controls han de servir per garantir una integritat, confidencialitat, disponibilitat, responsabilitat i seguretat del servei.
  Tot servei amb un plantejament de Arquitectura Segura, consta d'un conjunt de punts crucials que permeten definirla:
  - La relació dels diferents components i les seves dependències.
  - Les polítiques de controls definides i basades en l'avaluació.
  - La estandardització dels controls proposats.
- Mesures de Seguretat, la determinació de les mesures de seguretat a aplicar es basen en la prevenció, la detecció i la resposta. Existeix un conjunt molt ampli d'eines i mecanismes que permeten aplicar aquestes mesures. Totes elles es basen en l'aplicació de les polítiques següents:
  - Control d'accés de l'usuari.
  - Tècniques criptogràfiques per protegir el sistema de la informació.
  - Firewalls
  - IDS
  - Pla de resposta predefinit.
- Gestió de vulnerabilitats, fa referència a la gestió de les vulnerabilitats detectades -i no detectades- del sistema de la informació desenvolupat.
- Mecanismes de protecció del *Hardware*, dins d'un context de governança física de la infraestructura s'ha de proveir un conjunt de mecanismes de seguretat que garantitzin la integritat del *hardware*.
- Securitzar els Sistemes Operatius, correspon a la revisió dels sistemes operatius emprats per l'entitat i qüestionar com de segurs són.
- *Secure Coding*, és la pràctica de desenvolupar *software* emprant algoritmes que no generin cap tipus de vulnerabilitat. Els atacs més

[16] https://en.wikipedia.org/wiki/Principle_of_least_privilege
[17] https://en.wikipedia.org/wiki/Automated_theorem_proving

comuns dins d'aquest context són: *buffer-overflow*[18], *format-string attack*[19], *integer-overflow*[20] i *directory traversal*[21].

- *Capability-based security* i *Access Control List,* és un principi del modelat de la seguretat basat en la utilització d'aquest dos que poden eludir el "problema de diputat confús"[22]. Es tracta de disposar de les bondats de aquest dos i cap dels seus inconvenients.
- Educació de l'usuari, emfatitzar l'educació de l'usuari per tal de evitar incidents o bretxes de seguretat. Dins d'aquesta educació es pot incluir conceptes com la higiene digital que cerca l'assumpció per part de l'usuari en aplicar les recomanació de seguretat més bàsiques.
- Resposta a la bretxa, en el moment que es detecta una bretxa de seguretat s'ha de aplicar el pla d'acció esmentat en l'altre punt. Aquest ha d'aplicar un conjunt d'accions que minimitzin els possibles danys.

3.1 DevOp vs. Computer Protection

Al llarg de la introducció d'aquest capítol s'ha realitzat un recorregut dels principals actius, amenaces i possibles accions que minimitzen el risc davant la seguretat dels nostres sistemes de la informació. Però, com s'aplica això dins d'una cultura *DevOp*?

El concepte de *DevOp* és un principi orgànic, tracta d'emprar diferents metodologies, disciplines, tècniques o eines per arribar a un objectiu. Disposar d'altes garanties al consolidar un alt grau de coneixement del producte desenvolupat. En altres paraules, la conjunció de desenvolupament i operacions busca un alt grau de *expertis* en el producte desenvolupat.

Prenent com punt de partida els aspectes de seguretat esmentats a la introducció d'aquesta secció, farem un repàs de quines accions de seguretat sí estan cobertes en una cultura *DevOp* tal i com hem definit fins aquest punt.

3.1.1 DevOp vs. Security by design

Un desenvolupament segur des dels seus principis més bàsics es un dels aspectes fonamentals dins d'un desenvolupament segur. Però sin enfrontem aquest principi amb la cultura *DevOp*. Garantim un disseny segur?

Per poder-ne respondre aquesta pregunta ens hem de preguntar el grau d'implicació de l'equip. Un equip basat en la cultura *DevOps* té que està implicada desde el principi. Aquest inici, inclou la fase del disseny de l'aplicació. Per tant, és molt important que tots els integrants de l'equip siguin conscients de la importància d'aquesta fase desde un punt de vista de seguretat.

És molt fàcil identificar certes tendències que han estat adoptades dins de la cultura *DevOp*. Pràctiques com el *Code Review* o *Unit Testing* o ATP, totes elles són pràctiques fortament vinculades a la cultura.

---

[18] https://en.wikipedia.org/wiki/Buffer_overflow
[19] https://en.wikipedia.org/wiki/Uncontrolled_format_string
[20] https://en.wikipedia.org/wiki/Integer_overflow
[21] https://en.wikipedia.org/wiki/Directory_traversal_attack
[22] https://en.wikipedia.org/wiki/Confused_deputy_problem

Per altre costat, un control d'auditoria és una pràctica que queda relegada a una fase més de maduresa i suport. Per aquest motiu és molt comú realitzar les primeres versions sense un ampli concepte d'auditoria, fruit d'una mala planificació o manca de claredat en el funcional del propi producte.

3.1.2 DevOp vs. Security architecture

Una Arquitectura Segura requereix tenir molt clar els conceptes de integritat, confidencialitat, disponibilitat i responsabilitat.

Per poder-ne dur a terme una Arquitectura Segura es requereix disposar d'un coneixement clar de l'aplicació a entregar. Per tant, per poder-ner afirmar que un equip realitza una Arquitectura Segura té que tenir molt clara la relació dels components, les polítiques de control i l'estàndard emprat. Sense aquest tres grans punts, difícilment un equip *DevOp* pot afirmar que la seva arquitectura és segura.

3.1.3 DevOp vs. Mesures de Seguretat

Dins d'un entorn *DevOp* prenent mesures de seguretat? La resposta ràpida sería si, es prenent mesures de seguretat. Encara que moltes vegades cal remarcar que dins d'una cultura *DevOp* en un context de contenidors orquestables són ells els que defineixen les xarxes de comunicacions, protocols i criptografia a aplicar. Per aquest motiu, es pot dir que sí s'apliquen dins d'una cultura DevOp un principi de mesures de seguretat.

3.2 "The 16 Gates" i la cultura DevOp

L'any 2017 el director de "Capital One", Tapabrata Pal, va realitzar una ponència a la conferència "DevOps Enterprise Summit" on va començar a parlar de "The 16 Gates". L'any 2018 va plasmar aquestes ideas a un post on explica la seva experiència.

A aquest article emfatitza la necessitat d'entregar un producte d'alta qualitat i ràpid.

Per poder-ne arribar a aquest objectiu el Pipeline té un paper decisiu, engloba el concepte CI/CD. En el seu article el setze punts que defineix són:

1. Utilització d'un control de versions.
2. Estratègia òptima en gestió de branques.
3. Anàlisi estàtic.
4. Una cobertura d'un >80%
5. Escaneix de vulnerabilitats.
6. Escaneix de Open source
7. Control d'artefactes per versions.
8. Auto aprovisionament, IaC
9. Servidors immutables
10. Integration testing
11. Test de rendiment.
12. Build, deploy i testing automàticament a cada commit.
13. Rollback automàtic.

14. Ordres automàtiques segons canviï.
15. Zero downtime a cada release
16. Característiques per palanca.

Com es pot veure, tots els seus punts cercan l'automatisme extrem, on cada canvi al producte "dispara" un conjunt d'operacions que garanteixin el correcte funcionament de l'aplicació amb una capacitat d'entrega molt alta. Fent que el Pipeline hi jugui un paper crític dins de la cultura DevOp.

3.3 DevSecOp vs. DevOp

A causa de la importància fonamental que hi juga la seguretat dins dels serveis informàtics. Hi han aparegut als darrers anys un apelatiu nou, *DevSecOp* o *SecDevOp*. Això comporta una confusió a la gent, ja que qüestiona el rol del *DevOp* com paper principal, fent preguntes com; qué és un *DevSecOp*? O, quines diferències hi ha entre un *DevOp* i un *DevSecOp*?

Hem estat parlant del conjunt d'iniciatives que hi corresponen al *DevOp* per passar a analitzar la seguretat de les aplicacions. Hem vist la part més teòrica i hem contrastat les accions de seguretat que queden cobertes per la cultura *DevOp*. Malauradament, no sempre cobreix tot l'àmbit de seguretat com deuria.

Quan hem parlat de *Security by Design* hem comentat que àmbits com el *testing* o la revisió de codi eran pràctiques consolidades. En canvi, pràctiques com PoLP o *Defense in depth* o *fail safe*, són pràctiques menys comunes. Requereix un esforç de disposar un conjunt de casos extrems. Qüestionar-se com es comporta la nostra aplicació sota un comportament enòleg, o definir un principi de drets mínims o disgregació per capes. Comporta una inquietud que moltes vegades queda fora de l'abast de la nostra aplicació i requereix una visió global. En el mateix camí ens trobem quan parlem de *Security Architecture*, un dels punts claus es el coneixement de les interaccions dels diferents components. I continuant en aquest camí ens trobem *Capability-based security* i ACL, o modelat de la seguretat que requereix una maduresa en el mateix aspecte.

Si parlem del conjunt de mesures de seguretat aplicades veiem que es comú la manca de pla d'acció -o de resposta- davant una bretxa de seguretat. Es una situació molt extrema que requereix un pla clar i concís, ja que el no tindre-ho comporta una pèrdua de temps molt gran que sols hi incrementa el problema.

Altres aspectes que repasa el conjunt d'accions de seguretat són l'anàlisi de vulnerabilitats -tant del nostre aplicatiu com del Sistema Operatiu on s'executa-, com de segur és el nostre codi davant atacs i educar al consumidor -l'usuari- en l'ús de les nostres aplicacions.

Per tot l'esmentat anteriorment, quan parlem de *DevSecOp* el que s'intenta enfatitzar és la vinculació entre desenvolupament, seguretat i operatives, ja que tant el desenvolupament com les operatives estan fortament relacionades amb la seguretat.

En altres paraules, el moviment *DevSecOp* fa una crida d'atenció a l'equip, concienciant-los que tots són responsables de la seguretat. Una enfatització que s'ha d'integrar dins la cultura *DevOp* i no eclipsar-la.

Els aspectes que implica són el desenvolupament amb una perspectiva de *Security by Design, Security Architecture* i modelatge de la seguretat, estigui present en totes les fases del cicle de vida de l'aplicació. D'igual manera que dins del conjunt de mesures hi hagi un punt de vista que referencia la seguretat de les decisions preses. Punts com: la definició del control d'accéss d'usuari, les técniques criptogràfiques emprades i el pla de resposta davant una bretxa, són punts que necessiten resposta i un seguiment cíclic. A remarcar l'aspecte important de disposar d'un control exhaustiu d'anàlisis de vulnerabilitats i una verificació d'un codi segur. Finalment, fonamentar la inquietud de l'equip amb tercers mitjançant la gestió d'una auditoria de seguretat independent i, per l'altre costat, la educació del conjunt d'usuaris que hi consumeixen els nostres serveis de la informació.

3.4 DevSecOp i el Pipeline

El conjunt d'operatives que ha d'adoptar un *DevOp* forçosament passa per la declaració d'un *Pipeline*. Aquest permet determinar un conjunt d'operatives que administra i gestiona la nostra aplicació cercant la qualitat i l'entrega esmentades anteriorment. Però encara que les operatives de *DevOp* estiguin definides, com afecta a un Pipeline les operatives d'un *DevSecOp*?

Un *DevSecOp* tractar de garantir termes de seguretat en el software entregat o com menciona Tapabrata Pal (2018), un *software* ha de disposar d'una "alta qualitat, és a dir, sense defectes de seguretat, en compliment, defectes mínims, etc.". En altres paraules, la seguretat és un dels pilars de la qualitat del producte entregat.

Per tant, aplicar principis de seguretat al nostre Pipeline es una necessitat de qualitat que no podem eludir. Hem de cercar la verificació automàtica del nostre codi sense entrar en procesos automàtics que generin coll d'ampolla o un temps de verificació massa ampli. Aleshores, quina estratègia apliquem al nostre Pipeline que garanteixi seguretat?

3.4.1 El Pipeline i les tècniques SAST i DAST

El SAST o *Static Application Security Testing*, és l'eina d'anàlisi estàtic del codi font que permet esbrinar possibles errors de vulnerabilitat. El DAST o *Dynamic Application Security Testing*, és l'eina d'anàlisi dinàmic que permet determinar possibles vulnerabilitats en la nostra aplicació.

En una filosofía *DevOp* necessitem realitzar els anàlisis amb un temps mínim. No ens podem permetre parar al desenvolupador per causa d'operatives d'anàlisis que no són fluides. Per aquest motiu, la definició de quin es l'ordre d'utilitzar aquestes eines no és aleatori.

Si cerquem en el flux d'un desenvolupador, el més coherent sería començar per l'anàlisi estàtic. El principal motiu d'incloure tècniques d'anàlisi de codi font

a un dels primers pasos del Pipeline és per la proximitat que hi disposa el canvi realitzat per el programador i l'anàlisi estàtic del mateix. Fent una vinculació de test unitaris i anàlisi estàtic del codi. Un cop verificar el correcte funcionament podem delegar l'anàlisi dinàmic a una fase posterior ja que són recursos més exigents en potencia de CPU i temps.

3.5 Una ToolChain segura

Dins de la figura d'un *DevOp* hi ha una relació vertebral entre les seves tasques i la definició del *Pipeline*. Aquest mateix disposa de la mateixa vertebralitat per la vesant d'un *DevSecOp*. Totes dues filosofies depenen de la definició del seu *Pipeline*, hi condiciona la seva qualitat i la seva capacitat d'entrega.

Les "16 Gates" que ha de disposar un *Pipeline* és una projecció de la seva vinculació amb les tècniques SAST i DAST, però quines eines poden donar aquest principis d'anàlisi estàtic i dinàmic?

Per un costat quan parlem d'anàlisi estàtic estem parlant d'un context en el que hi tenim accés al codi font. Aquest accéss ens permet realitzar mètriques i definir umbrals de qualitat del percentatge de codi testejat i de si els seus algorismes són segurs.

Encara que hi ha moltes formes d'afrontar aquest tipus de desafiaments, aquí parlarem de l'eina més utilitzada per poder garantir l'anàlisi estàtic tal i com hem definit, l'eina és Sonarqube. *Sonarqube* permet realitzar la unificació de l'anàlisi estàtic segons les necessitats del nostre projecte. En primer lloc permet l'anàlisi estàtic d'un ampli conjunt de llenguatges, analitza els patrons de codi per evitar vulnerabilitats. Per un altre costat, disposa d'un ampli conjunt de *plugins* d'integració amb terces, de forma que pot mostra anàlisi de la cobertura de testeig realitzada o les vulnerabilitats de les llibreries emprades a l'aplicació, entre d'altres. A més, disposar d'una àmplia integració amb Jenkins, de forma que els umbrals de qualitat es pot definir a Sonarqube sense intervenir en la definició del *Pipeline*, facilitant el manteniment de la qualitat del nostre codi font.

Per l'altre costat tenim l'anàlisi DAST, aquí estem en un contexte de *blackbox*, no hi podem tenir la mentalitat del pas anterior. Necessitem pensar desde un punt de vista de consum, pròxim a la visió qui hi tendría un *hacker* cercant febleses. A diferencia de l'anterior fase, no hi disposem d'una eina capaç d'agrupar tota la informació. Hem de realitzar una composició de diferents eines que cobreixin la immutabilitat dels nostres serveis davant un atac. El primer d'aquest passos sería una fase de *PenTesting*, l'eina de referència a l'hora de realitzar aquest tipus de tasques es Zap del projecte OWASP aquesta eina permet realitzar un atac o escaneig de vulnerabilitats segons es configuri.

Encara que l'execució de *PenTesting* permet veure si hi ha codi potencialment vulnerable, hi cau la possibilitat de disposar de vulnerabilitats a nivel de Sistema Operatiu. Dins d'aquest punt hi ha dues opcions mayoritaries al mercat, Trivy i Clair. Totes dues opcions el que hi realitzen es un anàlisi de la imatge del contenidor generat i cerca possibles vulnerabilitats registrades.

3.6 Exemple pràctic de DevSecOp

Dins de l'apartat d'annex d'aquest treball s'adjunta l'explicació d'una POC basada en tecnología *Java*. La idea és mostrar un petit exemple de les operatives més comunes d'un *DevSecOp* com punt ilustratiu.

El procés de desenvolupament d'operatives d'un *DevSecOp* dins d'un context de CI/CD a un entorn de *Jenkins* es troba explicat a l'annex "Exemple de filosofía DevSecOp amb Jenkins".

La pregunta que podem fer-nos sería, el desenvolupament d'un *Pipeline* que integra anàlisi estàtic i dinàmic de la nostra aplicació aporta algun benefici?

Sí, aporta. El motiu d'aquesta afirmació es pot argumentar en dos punts:

- Vulnerabilitats detectades estàticament: A nivell estàtic hem guanyat en capacitat d'anàlisi ja que som conscients de la qualitat del nostre codi i de les vulnerabilitats de les seves dependències.
- Vulnerabilitats detectades dinàmicament: A nivel dinàmic hem fet un *PenTesting* que cerca vulnerabilitats i -com segona fase- analitza la nostra imatge creada valorant possibles vulnerabilitats a nivell de S.O.

D'igual manera que es necessita la captació d'informació per prendre les decisions de negoci més adients. Les modificacions realitzades a aquesta versió del *Pipeline* dona la informació necessària per prendre les decisions més adients segons la realitat de la nostra aplicació. En altres paraules, el valor afegit que dona la vesant d'un *DevSecOp* te una repercussió directa en la qualitat del producte entregat al ser un producte més segur.

4. MultiCloud deployment

En els capítols anteriors hem estat parlant de la importància de la seguretat vinculada al desenvolupament de les aplicacions. En esencia hem parlat de la seva importància i les possibles repercussions negatives que hi pot implicar la manca de seguretat en una aplicació. Aquest concepte s'agrupa en la figura d'un *DevSecOp*, el qual tracta d'integrar un enfoc més emfatitzat en la seguretat durant el procés de creació i entrega.

Al llarg dels capítols segón i tercer hem parlat de diverses operatives que permeten afegir diferents pràctiques que cerquen garantir una bona qualitat de l'aplicació. Però en cap moment hem parlat del *delivery*, pràcticament tot el temps hem estat parlant de la integració continua. El motiu és la complexitat del cas d'ús que aquí plantejarem al parlar de entrega continuada.

Tal i com s'esmenta a la introducció el CD avui en día està fortament condicionat a les despeses de disponibilitzar les nostres aplicacions a qualsevol *cloud*. La forta competitivitat entre els diferents proveïdors pot portar -a una entitat- voler migrar la seva entrega a un altre proveïdor.

4.1 Eines de desplegament Cloud

Avui día es molt comú necessitar una infraestructura virtualizada per un ampli ventall de beneficis que van desde economics fins a flexibilitat de recursos.

Si ens enfoquem a nivell tècnic podem parlar de IaC (*Infrastructure as Code*), el qual ens permet programa la nostra infraestructura facilitant les tasques al poder automatitzar la seva construcció i escalat.

Al trobar nos a un context altament mutable, hem de cercar eines que facilitin les operatives de creació i manteniment de la nostra infraestructura. Dins d'aquest rang ens centrarem en dues eines, *Terraform* i *Rancher*. Acotarem el nostre cas d'ús a una infraestructura dockeritzada sobre un *PaaS* de tipus *Kubernetes*.

Si parlem de *Terraform* estem parlant d'una eina de *HashiCorp* pensada per la construcció, canvi i versionat de la infraestructura d'una manera segura i eficient. El foc principal de *Terraform* es la gestió de: *clouds* -públics o privats-, xarxes, software com a servei o plataforma com a servei. Aquesta gestió la realitza mitjançant la definició de proveïdors.

Terraform es focalitza en una infraestructura declarada emprant la sintaxis de configuració HCL (*HashiCorp Configuration Language*).

Si analitzem *Rancher* veiem que es una eina amb un enfoc molt diferent, està enfocat a la gestió de *Multi-Cluster*s que poden ser de diferents *Clouds*. Les principals característiques de *Rancher* són:

- agnostic: implementat desde una vessant *cloud* agnostic, s'integra amb els principals proveïdors dins d'un marc de *Kubernetes*.
- *Hybrid cloud*, al tractar-se de ser una eina agnòstica permet integració amb infraestructura privada.

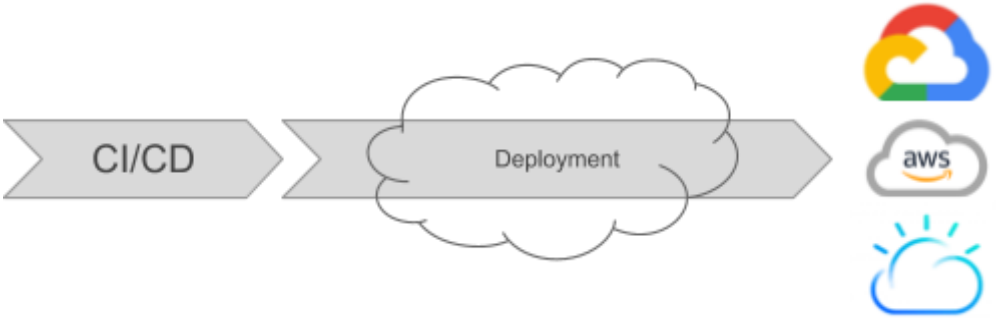
- centralització de polítiques, dins d'un marc de gestió de diversos *clusters*, *Rancher* es presenta com un punt comú de gestió.
- centralització de gestió d'incidències i monitorització amb eines com Prometheus.

Salta a la vista que dins d'un context de gestió *Multi Cloud Rancher* es l'opció que més s'adapta a l'entorn que estem analitzant. Mentres que amb *Terraform* es necessita una declaració per cada proveïdor que adapti el desplegament a la seva infraestructura, amb Rancher podem gestionar tant l'entrega com les polítiques dels diferents *clusters*.

Aleshores, si integrem al nostre *delivery* amb *Rancher* podem assegurar que les nostres aplicacions s'implanten amb garanties d'integritat a un context tan heterogeni com hem descrit?

4.2 Bases per un desplegament segur

El nostre punt de partida es el delivery d'una aplicació a un entorn *Multi Cloud*, quan parlem d'aquest entorn estem parlant d'un entorn altament flexible. Imaginem la nostra situació amb l'ajuda del diagrama d'acontinuació.



Per accedir a algun dels diferents *cloud* proveïdors, hem de realitzar les operatives de desplegament distribuït -ja sigui emprant *Rancher* o qualsevol altre producte- mitjançant Internet com pont de comunicació. Indirectament estem exposant les actualitzacions dels nostres serveis de la informació a la xarxa. Podem ser víctimes de diversos tipus d'atacs en el moment de realitzar els desplegaments o actualitzacions. Per aquest motiu, necessitem cercar marcs de treball que ens permetin gestionar la seguretat de tot el conjunt d'actors que garanteixin una gestió correcta en el nostre entorn. Els dos marcs de treball que es proposa són *The Update Framework* (TUF) i *Open Policy Agent* (OPA).

4.2.1 The Update Framework (TUF)

TUF pertany al *Cloud Native Computing Foundation* (CNCF) i és un marc de treball que tracta de detallar les pràctiques per garantir una actualització segura en un sistema de la informació. Segons ells mateixos una actualització de *software* es segura si:

- coneix les últimes actualitzacions de manera oportuna.
- els arxius que es descarrega són els correctes.
- no es produeix cap dany al comprovar o descarregar els arxius.

Aquest marc de treball tracta de garantir una actualització de *software* amb la suficient seguretat de no fer cap malbé al sistema de la informació, per això els principals atacs o debilitats que tractar de combatre són:

- **Arbitrary software installation,** un atacant pot proporcionar fitxers aleatoris tractant de respondre a la petició de descarga de *software* que un client ha sol·licitat.
- **Rollback attacks,** un atacant pot presentar fitxers obsolets a un sistema per tractar de mantenir una versió que presenta vulnerabilitats i explotar-les en un futur immediat.
- **Fast-forward attacks,** un atacant pot incrementar la versió del projecte de forma que qualsevol tipus d'actualització el sistema el pot interpretar com un *rollback* o una versió més darrera o *out-of-date*. Amb aquesta tècnica s'aconsegueix un *freezing* de la versió.
- **Indefinite freeze attacks,** l'atacant ofereix -de manera continuada- informació d'actualització al sistema de manera que aquest deixa al sistema obsolet al no poder accedir a les actualitzacions disponibles.
- **Endless data attacks,** un atacant respon a una petició de descàrrega amb un final del *stream* de informació incorrecte. De forma que hi causa dany a aquest.
- **Extraneous dependencies attacks,** un atacant indica a un client que necesita descargar *software* no relacionat. Aquest pot donar un punt d'accéss a l'atacant.
- **Mix-and-match attacks,** un atacant pot presentar a un client un conjunt de fitxers inexistents a un repositori. Amb el resultat d'instal·lar dependencies no necessàries amb vulnerabilitats explotables.
- **Wrong software installation,** un atacant proveeix al client una versió confiable que no es la desitjada per el client.
- **Malicious mirrors preventing updates,** un atacant controla un *repository mirror* i intercepta les actualitzacions legítimes d'altres repositoris que no disposen de codi maliciós.
- **Vulnerability to key compromises,** un atacant que pot comprometre la clau única del sistema, comprometent als clients.

TUF tracta de combatre un conjunt d'atacs centrats en crear un *out-of-date* del sistema o la inclusió d'una versió del *software* inapropiada. Per garantir una adequació en la versió del *software* disposa de quatre principis de seguretat que són:

- **Trust:** credibilitat en la descàrrega dels arxius implica que aquest no disposen cap tipus de *software* maliciós. Aquesta credibilitat està basada en:
    - La credibilitat no és per sempre, aquesta pot expirar sino es renovada.
    - La credibilitat no es la mateixa per tot. Un model de credibilitat compartimentada permet aplicar rols de confiança segons sigui estipulat.
- **Compromise-Resilience:** La signatura criptografica es un component necessari en la gestió d'actualitzacions. La seguretat de les claus que generan les signatures comprometen la seguretat dels sistemes. Per aquest motiu no s'assumeix que les claus privades estan sempre

segures. Una actualització de *software* ha d'anticipar com mantenir als seus clients segurs quan es compromet la privacitat d'aquestes claus. Mantenir als clients segurs de qualsevol compromís de les claus implica: una gestió ràpida i segura en la substitució i revocació, una minimització del risc de la localització de les claus i una gestió multi-clau en el seu ús i el llindar i el quòrum de la signatura.

- **Integrity:** La integritat no implica únicament al fitxer d'actualització, també inclou al repositori que allotja la informació. És molt important una correcta visió d'aquest per garantir la provisió correcta de les versions.
- **Freshness:** és una pràctica molt comuna la gestió d'actualitzacions que cerquen fitxar errors de seguretat. Garantir *freshness* implica no aceptar fitxers més antics dels vistos i reconèixer quan hi ha un problema per obtenir una actualització.

4.2.2 Open Policy Agent (OPA)

És un projecte *open source* que defineix un motor de polítiques de propòsit general que les unifica al llarg de tota l'arquitectura. Utilitza un llenguatge declaratiu, que pot especificar les polítiques d'un conjunt molt dispar com són els microserveis, *Kubernetes*, CI/CD, *pipelines* o APIs. Sent un motor de polítiques agnòstic al domini i que es centra en evaluar les peticions d'entrada amb la declaració de polítiques i les dades.

La filosofía de OPA és basa en un conjunt de regles que determinen el comportament de l'aplicatiu, una política. Aquesta pot determinar diversos aspectes de la plataforma, desde la confianza dels servidors fins al direccionament de la xarxa. La seva base agnòstica permet abstreures i determinar les regles que poden comprometre el correcte funcionament del sistema de la informació. Aquest és un dels punts forts de OPA, permet determina una política desacoplada per tot el nostre sistema de la informació, no és necessari revisar tot el conjunt de polítiques de l'arquitectura per realitzar un canvi, tan sols, actualitzar la política que necessitem sense impactar en cap altre. Aquest desacoblament permet realitzar actualitzacions més ràpides que juntament amb la seva facilitat semàntica facilita la gestió i la declaració. A més, OPA disposa d'un principi de integració amb les dades de tercers, fet que permet gestionar -entre d'altres- el *token* de seguretat JWT o la validesa d'un TTL.

La gestió de OPA es basa en exposar un interfície que permet la integració en la gestió i l'execució de les polítiques. Aquestes interfícies estan definides en:

- **Avaluació:** interfície focalitzada en demanar la política de decisió a aplicar en el servei, eina, etc. El resultat és un desacoblament entre l'aplicació i la política.
- **Management:** interfície focalitzada en la distribució, status o actualització de la política.

OPA és la base per la gestió de seguretat de molts sistemes, *Kubernetes* disposa del *Admission Controller*, que es el punt on s'aplica les polítiques de gestió. L'*admission Controller* és l'interceptor de peticions de la API de *Kubernetes* abans de realitzar una persistencia al sistema aquest també

disposa d'integració de tercers -mitjançant *webhooks*- que permeten la integració dins del *Dynamic Admission Control*.

L'objectiu d'aquest component es la intercepció de qualsevol interacció amb l'API de *Kubernetes* i aplicar la política adequada a la declarada per l'administrador.

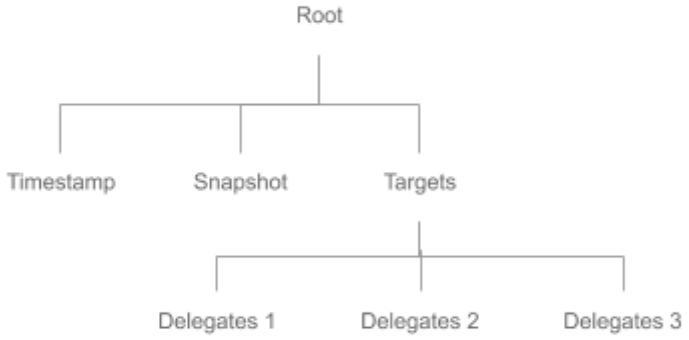
4.3 Les dues cares per un desplegament segur

En la darrera secció hem estat parlant de la part més teòrica, TUF és un marc que cercar garanties en les actualitzacions i OPA és la sintaxi declarativa de les polítiques de cadascun dels aspectes de seguretat a gestionar i que s'integra amb els diferents *stacks* tecnològics. En el nostre cas *Kubernetes* integra OPA mitjançant l'*Adminission Controller,* punt on s'aplica els criteris de política declarades.

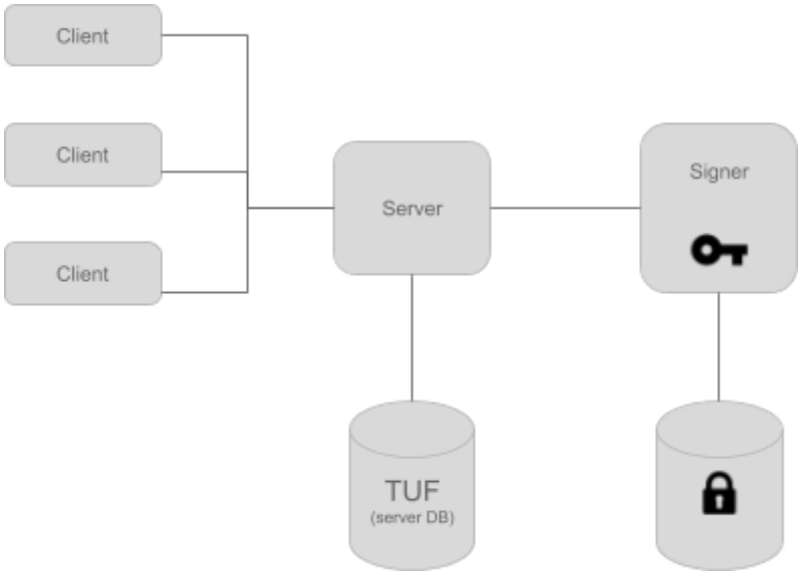
Disposem de dues eines per gestionar la seguretat en els nostres deplegaments. Això implica la gestió de dos actors que seran els responsables de garantir la seguretat. Malauradament TUF és un marc de treball però no és una especificació i l'*Admission Controller* de *Kubernetes* requereix integrar un tercer per gestionar les polítiques de signatura que requerim. Per aquest motiu parlarem de *Notary* com implementació de TUF i *Portieris* com *Admission Controllers* de verificació d'imatges.

4.3.1 Notary

Notary es la implementació de TUF que gestiona la seguretat en les actualitzacions. Aquesta seguretat està basada en la signatura criptografica de les actualitzacions sota un principi de credibilitat basada en rols.



Notary implementa l'arquitectura de rols proposta per TUF, permetent el perfilatge de la credibilitat i "protegint" la informació criptogràfica al compartimentar la credibilitat en rols. La seva arquitectura és la següent:



Els actors són:

- *Notary Client* que pot generar la informació de la imatge i facilitar-la al servidor *Notary*.
- *Notary Server,* responsable de guardar la informació facilitada per el client.
- *Notary Signer* que custodia les claus xifrades i signar la informació facilitada per el *Notary Server*.

Exemple de cas d'ús:

1. El client facilita la informació -*metadata*- al *Notary Server*, el qual soporta JWT com mètode d'autorització.
2. El *Notary server* la verifica i contrasta si existeix conflicte amb versions prèvies, verifica la signatura y *checksums*.
3. *Notary Server* sella amb un *timestamp* la informació i la envia la *Notary Signer* per ser signada.
4. El *Notary Signer* rep la sol·licitud i cerca les claus a la seva base de dades per realitzar la signatura. Si és satisfactori retorna la signatura al *Notary Server.*
5. *Notary Server* gestiona la combinació de la informació facilitada per el *Notary Client* i el *Notary Signer*. Un cop integrat tots dos notifica al client que l'actualització de la informació ha estat correcta.
6. El client té disponible la informació per descarregar-se.

4.3.2 Portieris

Portieris és un producte *open source* de IBM que defineix un *Admission Controller* instalable en el *cluster* de *Kubernetes*.

Portieris es pot instal·lar mitjançant *Mutating Admission Webhook* que permet modificar el *Kubernetes resources* al punt de creació. De forma que ratifica la credibilitat de les imatges al crearse al cluster. Aquesta gestió de polítiques pot realitzar mitjançant *namespaces*, nivell de cluster o segons la imatge que es vol desplegar.

La gestió de la confiança de les imatges que es vol instal·lar al *cluster* es pot fer mitjançant la connexió directa que ofereix Portieris amb el servidor de *Notary*, centrant-nos en la gestió de les polítiques i no en tasques operatives.

Bàsicament *Portieris* utilitzarà la informació de *Notary Server* mitjançant el seu client integrat per accedir a les claus públiques de la imatge i admetre o denegar la seva creació.

4.3.3 Diagrama per un desplegament segur

A les darreres seccions hem estat parlant de *Notary* i *Portieris*. Ara parlarem de com s'integra en el nostre ecosistema de CI/CD.

Si parlem dels actors que intervenen en la gestió de la signatura de la informació podem fer la primera aproximació en el nostre CI.

Un *Notary Server* pot donar servei a diferents clients facilitant la coordinació de la informació de les diferents imatges generades. Per tant, en el nostre *pipeline* sería integrar un client de *Notary* per poder-ne realitzar les signatures de les versions susceptibles de ser desplegades en un entorn, quedant un *stage* que gestiona la *metadata* amb el client que cerca la credibilitat en el desplegament.

Un cop confirmada aquesta informació es pot crear el *stage* de desplegament, aquest realitzaría la integració amb *Rancher*, eina que integra dos conceptes claus per la gestió d'un desplegament *Multi Cloud.* Per un costat, gestiona diferents proveïdors i facilita el desplegament. I per l'altre costat, tenim la seva capacitat de gestionar les polítiques dels diversos *clusters*. Aquesta gestió facilita la gestió de l'entorn que pasaría per instal·lar *Portieris* i actualitzar la política a aplicar.

5. Conclusions

A aquest capítol es recull les conclusions, els objectius complerts i possibles línies futures obtingudes de desenvolupar aquest treball.

5.1 Conclusions

El *DevSecOp* és la resposta a un desenvolupament més segur, no es tracta d'introduir un nou rol a l'equip. Sino de fomentar la mateixa inquietud de la cultura *DevOp* i que elimina els silos d'una organització tradicional, per qüestionar-se com de segures són les nostres aplicacions i els seus procesos de creació i entrega.

Per aquest motiu un *DevSecOp* no és més que un *DevOp* que relaciona el desenvolupament i les operatives mitjançant la seguretat. Fent que la seguretat estigui present a tots dos llocs i sigui un nexe d'unió.

Per aquest motiu un *DevSecOp* es troba amb un conjunt de reptes a afrontar tres punts fonamentals. Incentivar una aplicació segura desde el seu disseny, fomentar mides d'anàlisis de seguretat estàtiques i dinàmiques i garantir una entrega segura a la plataforma destí. Aquest últim, implica adoptar mides que doni garanties d'una entrega oportuna, correcta i que no contingui cap tipus de dany al sistema de la informació.

5.2 Objectius complerts

L'objectiu d'aquest treball era estudiar la realitat d'un *DevSecOp* a un entorn *Multi Cloud*. Malauradament l'amplitud del tema és molt i afirma que s'ha aconseguit és pecar d'incrèdul. S'ha identificat un ventall de reptes que ha d'afrontar un *DevSecOp* que van desde la integració continuada fins a un *delivery* segur. Però els reptes per un *DevSecOp* a sobre la taula s'amplien segons s'estudia la matèria i evoluciona la tecnologia.

5.3 Línies per un treball futur

Les línies principals que es vol proposar són:

- Aprofundir en un *delivery* segur en termes de la gestió criptogràfica al emprar The Update Framework com marc d'ús.
- Analitzar una gestió segura de les eines emprades a les diferents operatives CI/CD i el seu impacte.
- Analitzar un entorn adequat per albergar tot els serveis necessaris per les operatives CI/CD a un context *Multi Cloud*.

6. Glossari

ACL: Access-control list, llistat de permisos associats a un objecte.
API: és una interfície que declara com els diferents aplicatius informàtics interaccionen.
Artifactory: Argrupació d'un conjunt d'arxius que simplifiquen la seva gestió al disposar de metainformació sobre ell mateix.
ATP: *Automated theorem proving* o deducció automàtica, consisteix en realitzar teoremes automatitzats que demostren el correcte funcionament del programari.
CD: Continuous Delivery, tècnica del desenvolupament de *software* on es busca cicles curts per poder-ne garantir la entrega en el menor temps possible.
Checksum: técnica basada en la suma de verificació d'un fitxer per protegir la integritat del mateix.
CI: Continuous Integration, tècnica de desenvolupament de *software* que busca falladas del mateix emprant proves automàtiques.
CLI: *Command Line Interface*, mètode que permet donar instruccions a un programa informàtic mitjançant líneas de text simples.
CPD: Acrònim de Centre de processament de dades, és l'espai on es localitzen els recursos necessaris per el processament de la informació d'una organització.
Cluster: Conjunt d'ordinadors que es comporten com un sol.
Host: Fa referència a la màquina que fa un rol de amfitrió.
Malware: Aplicacions malicioses amb l'objectiu de realitzar accions nocives en el sistema informàtic objectiu.
IDS: Intrusion Detection System, és un servei -pot ser un dispositiu o un *software*- que monitoritza a nivell de xarxa o sistema, cercant violacions de la política seguretat o activitat maliciosa.
Namespace:
Pipeline: Un Jenkins Pipeline és un conjunt de connectors que permet implementar i integrar canonades de lliurament continu.
POC: Proof Of Concept, acrònim de prova de concepte.
PoLP: *principle of least privilege*, principi que consisteix en donar -estrictament- accés als recursos necessaris per a cada part del sistema.
Repository: és una localització on s'emmagatzema i manter la paquetería d'un programa.
SAST: Static application security testing, process que tractar de garantir la qualitat del *software* analitzant el codi font.
SDK: *Software Development Kit* és un conjunt d'eines per el desenvolupament de *software* per un ecosistema específic.
TimeStamp: Marca temporal o seqüència de caràcters que denoten hora i data d'un event.
VC: acrònim de Control de Versions, eina emprada per el desenvolupament de *software* per disposar un dietari de versions.
WebHook: mètode per el qual es realitza crides personalitzades a tercers dins del seu flux d'execució.

7. Bibliografia

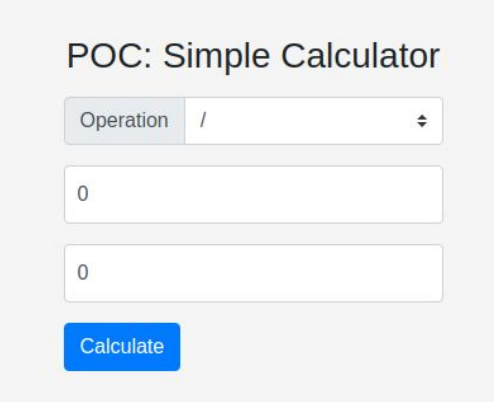
- Jordi Guijarro Olivares, Joan Caparrós Ramírez i Lorenzo Cubero Luque, DevOps y seguridad Cloud, primera edició, Editorial UOC, Barcelona, 2019.
- Robert C. Martin, Clean Code, A Handbook of Agile Software Craftsmanship, Prentice Hall, Boston, 2009
- Patrick Debois, Agile infrastructure and operations: how infra-gile are you? 2008.
- https://waterplacid.files.wordpress.com/2019/02/periodic-table-of-devops-tools.pdf, 12 d'octubre de 2020.
- https://www.weave.works/blog/gitops-operations-by-pull-request 21 d'octubre de 2020.
- https://nvie.com/posts/a-successful-git-branching-model/ 23 d'octubre de 2020.
- https://www.jenkins.io/doc/ 25 d'octubre de 2020.
- https://docs.travis-ci.com/ 25 d'octubre de 2020.
- http://www.opensecurityarchitecture.org/cms/definitions/it-security-architecture 14 de novembre de 2020.
- https://en.wikipedia.org/wiki/Capability-based_security 14 de novembre 2020.
- https://en.wikipedia.org/wiki/Computer_security 14 de noviembre de 2020.
- https://owasp.org/www-project-proactive-controls/ 15 de noviembre de 2020.
- https://medium.com/capital-one-tech/focusing-on-the-devops-pipeline-topopal-833d15edf0bd 16 de noviembre de 2020.
- https://theupdateframework.io/ 14 de decembre de 2020.
- https://www.openpolicyagent.org/ 14 de decembre de 2020.
- https://github.com/IBM/portieris 22 de decembre de 2020.
- https://github.com/theupdateframework/notary 22 de decembre de 2020.
- https://kubernetes.io/es/docs/home/ 22 de decembre de 2020.

8. Annexos

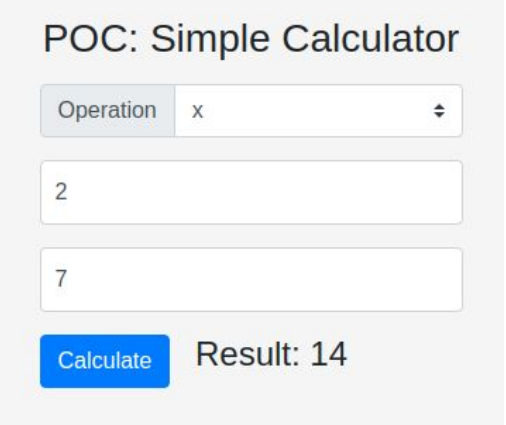
8.1 Explicació de la *POC*

En aquest annex farem una petita explicació de l'aplicació POC emprada al llarg de tot el document. El punt de partida ha estat una aplicació basada en *Spring Boot* amb el *actuator* de *Spring MVC* per oferir una *POC* en format *web*. Per evitar qualsevol tipus de soroll en l'exemple, l'aplicació no disposa de cap tipus de servei extern. Es dir, no hi utilitza cap tipus de servei de base de dades o API externa.

La seva operativa és molt senzilla; realitza operacions aritmètiques de suma, resta, multiplicació i divisió. Disposa d'un selector d'operació i dos *inputs fields* per introduir els valors per l'operació.



Com es pot observar el formulari es simple y permet operar amb resultats amb la disposició de la informació de la manera següent:



Cal resaltar el projecte disposa de les bones pràctiques de desenvolupament més comunes. Emprar una política de proves unitàries i d'integració adients al llenguatge i a les estructures més comunes per un projecte estructurat en *maven*.

8.2 Exemple de filosofía *DevOp* amb *Travis CI*

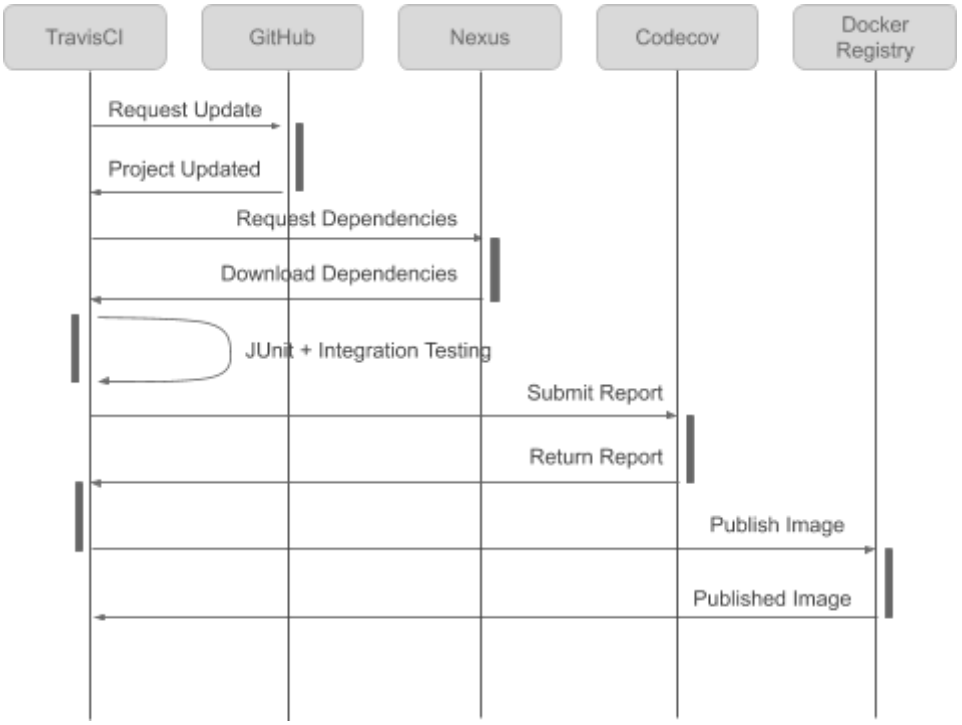
Emprant l'aplicació explicada en l'apartat anterior, farem un exercici de CI/CD emprant una de les solucions *SaaS* més comunes, *Travis CI*.

La idea es realitzar les operatives més comunes dins d'un equip de desenvolupadors que volen implementar un flux de CI/CD basat en *Github*, *Travis CI* i desplegant al *GKE* de *Google* mitjançant l'ús del *Docker Registry* de *Docker Hub.*

Com tot *Pipeline* que es vol desenvolupar es necessita un servei de control de versions. Dins del conjunt d'opcions *Github* és de les més esteses. La idea es realitzar un ús basat en fitxers descriptors semàntics dins del context del proveïdor, *Travis CI*.

TravisCI s'encarrega de gestionar l'entorn de CI/CD i integrar-se amb *Github*, de forma que vincular tot dos no pasa de realitzar un parell d'acceptacions de permisos. La definició del comportament que volem realitzar està definit al fitxer *travis.yml* a l'arrel de projecte. Dins d'aquest fitxer tenim definits sis fases o "stages" que són:

- Descarrega de dependències.
- Tests unitaris.
- Test d'integració.
- Publicació de la *Release Candidate*.
- Publicació de la versió.
- Deploy -desenvoluparem aquesta fase en propers capítols-.



8.2.1 Descarrega de dependències

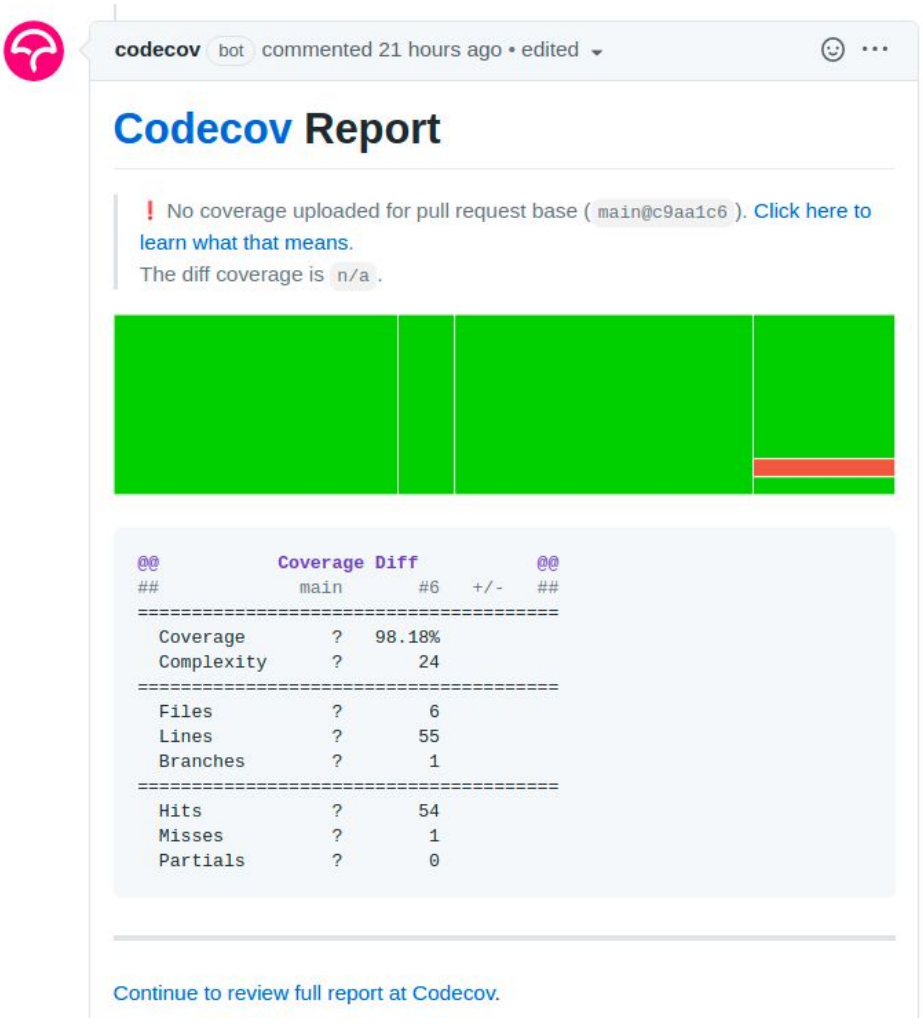
Aquesta fase està molt vinculada a la gestió de dependències que hi ofereix *maven*. Per aquest motiu abans de realitzar qualsevol pas és necessari descarregar les dependències de *maven* con primer pas.

8.2.2 Tests unitaris

Un cop descarregada les dependències es passa a realitzar les proves unitàries per garantir el correcte funcionament.

Part del treball d'un desenvolupador es garantir la cobertura de les seves proves. El percentatge de cobertura tractar de buscar garanties en l'ús de les diferents funcionalitats de l'aplicació. En el nostre cas l'aplicació realitza la cobertura amb el *plugin Jacoco*[23]. Per aquest motiu la integració que proposa la documentació de *Travis CI* per garantir un *reporting* adequat passa per utilitzar Codecov[24]. Aquest últim permet mostrar les dades obtingudes de forma que el *Pull Request* es pot visualitzar de la següent manera.

[23] https://www.eclemma.org/jacoco/trunk/doc/maven.html
[24] https://codecov.io/



codecov bot commented 21 hours ago • edited

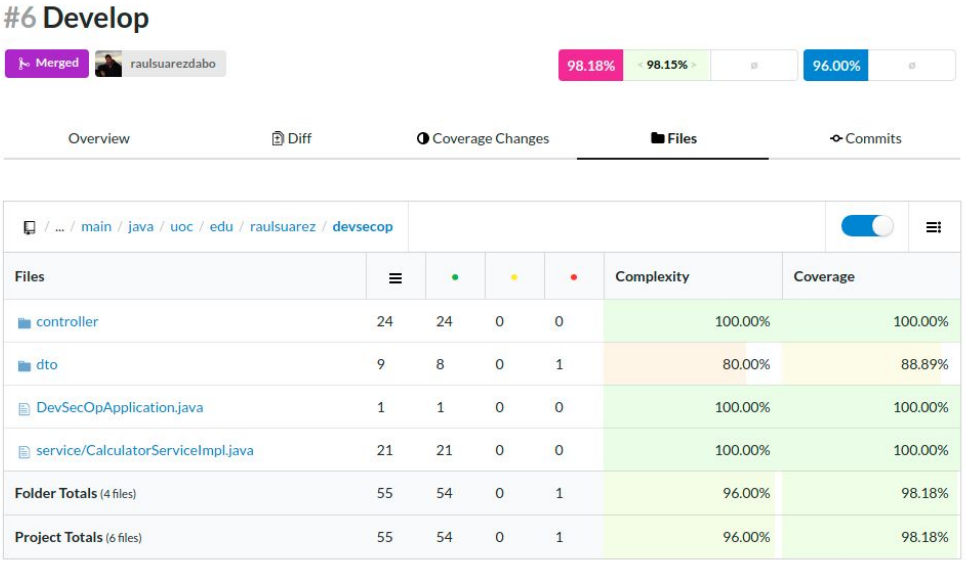
Codecov Report

! No coverage uploaded for pull request base (`main@c9aa1c6`). Click here to learn what that means.
The diff coverage is `n/a` .

```
@@           Coverage Diff           @@
##             main       #6   +/-   ##
=======================================
  Coverage        ?   98.18%
  Complexity      ?       24
=======================================
  Files           ?        6
  Lines           ?       55
  Branches        ?        1
=======================================
  Hits            ?       54
  Misses          ?        1
  Partials        ?        0
```

Continue to review full report at Codecov.

D'aquesta manera es pot explotar les dades i valorar si el *Pull Request* compleix amb les polítiques de *testing* de l'equip. D'igual manera permet visualitzar la informació facilitada per l'extensió segons sigui necessari.



#6 Develop

Merged raulsuarezdabo

98.18% 98.15% 96.00%

Overview | Diff | Coverage Changes | Files | Commits

/ ... / main / java / uoc / edu / raulsuarez / devsecop

Files					Complexity	Coverage
controller	24	24	0	0	100.00%	100.00%
dto	9	8	0	1	80.00%	88.89%
DevSecOpApplication.java	1	1	0	0	100.00%	100.00%
service/CalculatorServiceImpl.java	21	21	0	0	100.00%	100.00%
Folder Totals (4 files)	55	54	0	1	96.00%	98.18%
Project Totals (6 files)	55	54	0	1	96.00%	98.18%

8.2.3 Test d'integració

Per tal de garantir un correcte funcionament disposem d'una classe de *Java* que realitza una sèrie de proves funcionals per garantir el correcte funcionament del flux de "negoci" dins d'un context de la *JVM* de *Java*.

8.2.4 Publicació de la Release Candidate

Un cop tenim la conformació de la qualitat de la nostra aplicació és necessari generar un contenidor amb l'aplicació. Aquest *stage* és l'encarregat de generar el contenidor amb la versió adequada.

Abans d'explicar el flux d'aquest pas és necessari remarcar que -basant-nos en la gestió de branques de *Gitflow*[25] proposat per Vincent Driessen- considerem versió candidata a tota versió disposada a la branca de *develop*. Per tant, aquesta fase es exclusiva a aquesta branca per tal de garantir un control de versionat coherent amb la filosofía *Gitflow*.

Un cop realitzat es publica al *Docker Registry* de *Docker Hub* amb l'etiquetat basat en les variables de ofereix el proveïdor seguit dels caràcters "-RC".

8.2.5 Publicació de la versió

Continuant amb la filosofia *Gitflow*, tota versió a la branca *main* es una versió que s'ha de considerar versió tancada i potencialment desplegable a l'entorn. En el nostre cas aquest *stage* realitza el versionat doble, per un costat la versió que l'hi pertoca amb les variables que disposa *Travis CI* i la versió *latest.* Considerada com la darrera versió disponible.

Aquest canvis s'apliquen i es publica a *Docker Hub* al igual que el pas anterior.

[25] https://nvie.com/posts/a-successful-git-branching-model/

8.3 Exemple de filosofía *DevOp* amb *Jenkins*

Emprant l'aplicació explicada en l'apartat "Explicació de la POC", farem un exercici de CI/CD emprant el producte estrella, *Jenkins.*

La idea es realitzar les operatives més comunes dins d'un equip de desenvolupadors que volen implementar un flux de CI/CD basat en *Github*, *Jenkins* i un desplegant futur a un entorn *Cloud* de moment indeterminat i que es realitzarà mitjançant l'ús del *Docker Registry* de *Docker Hub.*

És molt important tenir present que la configuració del servei ha implicat la creació del laboratori sota la composició de contenidors que ofereix *Docker Compose*[26]. És crea un servei de *Jenkins* amb accés -mitjançant *socket*- al *daemon* de *Docker* del *host*. Amb aquesta especificació és crearan els contenidors sota demanda per cadascun de les fases definides al *Pipeline*.

Un cop realitzat el *setup* necessari s'ha optat per es realitzar un ús basat en fitxers descriptors semàntics dins del context de *Jenkinsfile.* Per dur a terme això s'ha d'emprar un *pipeline* de tipus *Multibranch Pipeline* el qual ens permet realitzar les tasques de CI/CD sobre totes les branques del repositori.

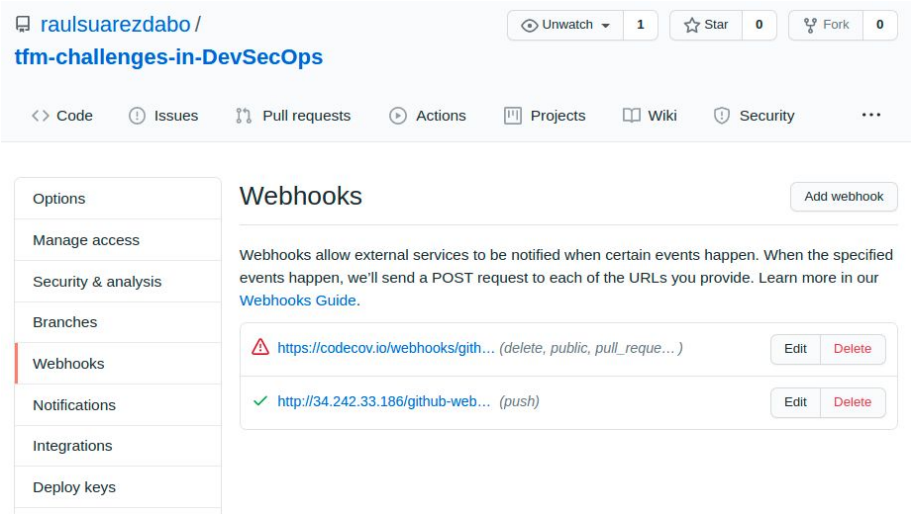
Com que el servei no està integrat nativament amb *Github* és necessari realitzar unes tasques per vincular *Jenkins* i *Github*.

Per un costat és necessari declarar a quin repositori ha de fer *checkout Jenkins* per descarregar-se el codi font.

Per l'altre costat es necessari declarar un *WebHook* a *Github* amb la definició pertinent perquè informi a a *Jenkins* els canvis aplicats al control de versions.

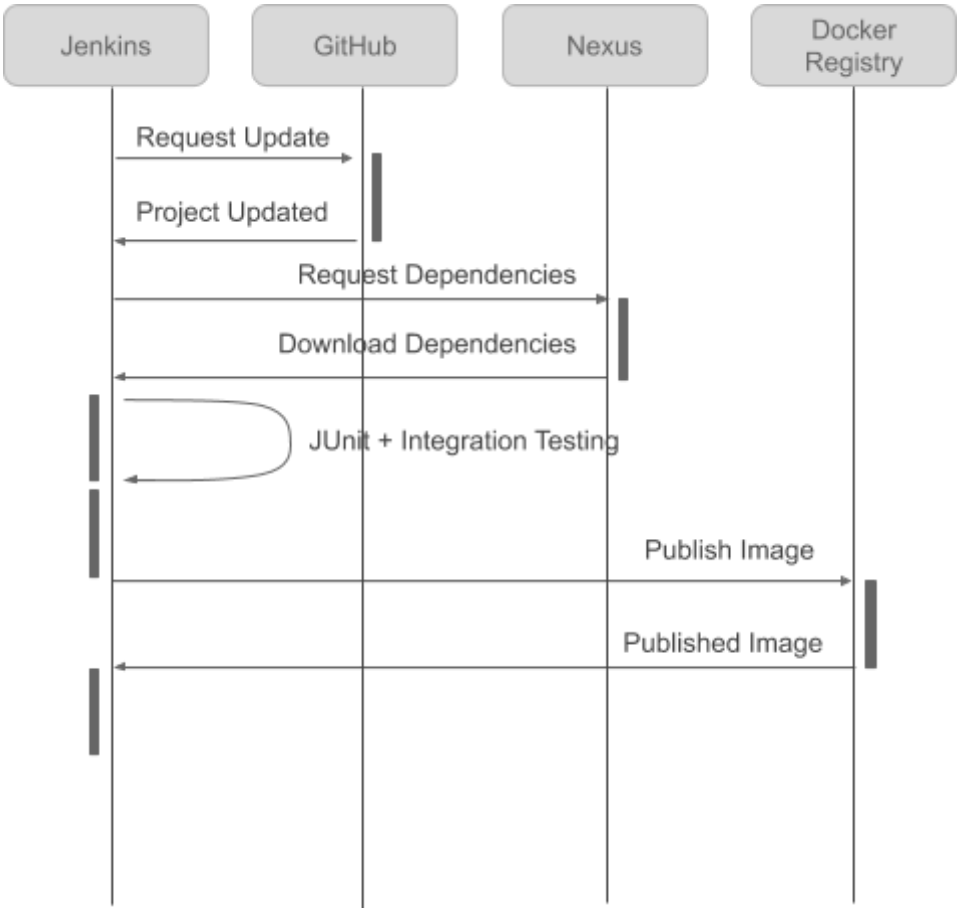
El fitxer descriptor semàntic Jenkinsfile s'encarrega d'indicar de definir el comportament que volem realitzar.

[26] https://docs.docker.com/compose/



Amb aquestes darreres operacions ja ens podem focalitzar en explicar les sis fases o “stages” amb un diagrama de flux:

- Descarrega de dependències.
- Tests unitaris.
- Test d’integració.
- Publicació de la *Release Candidate*.
- Publicació de la versió.
- Deploy -desenvoluparem aquesta fase en propers capítols-.

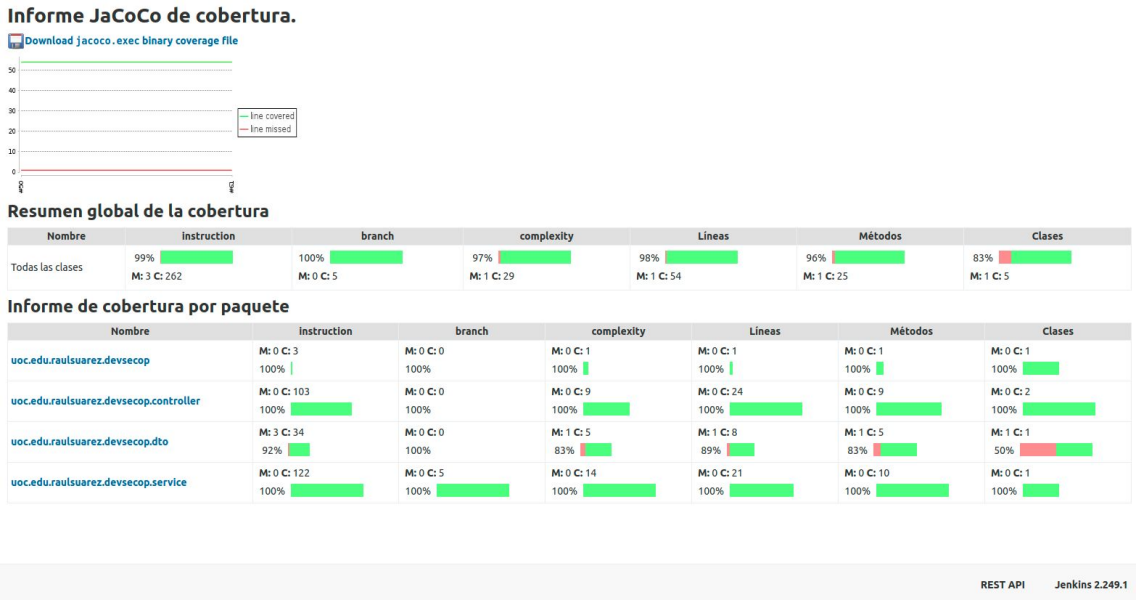


8.3.1 Descarrega de dependències

Al igual que en l'annex anterior vincula la gestió de dependències de maven com primer pas.

8.3.2 Tests unitaris

La definició de testeig està fortament vinculada al propi servei de *Jenkins.* El motiu és a causa que el *reporting* de *Jacoco* queda integrat amb el seu *plugin*[27]. Fent que sols sigui necessari indicar en el descriptor la localització del fitxer *jacoco.exe* el qual conté la informació de cobertura vinculada als tests.



8.3.3 Test d'integració

Els tests d'integració funcionen sota la premissa d'una classe de *Java* que realitza una sèrie de proves funcionals per garantir el correcte funcionament del flux de "negoci" dins d'un context de la *JVM* de *Java*.

8.3.4 Publicació de la Release Candidate

Un cop validat la qualitat de la nostra aplicació és necessari generar un contenidor amb aquesta. Aquest *stage* és l'encarregat de generar el contenidor amb la versió adequada.

Al igual que el seu homoleg, realitzarem les candidates a *release* en base a la versió disponible a la brancha *develop.* Fent que aquesta fase sigui exclusiva a aquesta.

La publicació al *Docker Registry* de *Docker Hub* amb l'etiquetat basat en el número de job seguit dels caràcters "-RC".

[27] https://plugins.jenkins.io/jacoco/

8.3.5 Publicació de la versió

Tota versió a la branca *main* es una versió potencialment desplegable a l'entorn. En el nostre cas aquest *stage* realitza el versionat doble, per un costat la versió que l'hi pertoca amb les variables que disposa *Jenkins*.

Aquest canvis s'apliquen i es publica a *Docker Hub* al igual que el pas anterior.

8.3.6 Desplegament

El desplegament es el punt que hi realitza la implantació del contenidor dins l'entorn *Kubernetes* de tercers. Com que l'estratègia de desplegament està molt vinculada a l'administració d'un entorn *Multi Cloud*. Deixarem aquesta part per desenvoluparla més endavant i ens fixarem que en aquest cas no s'utilitza cap tipus de proveïdor extern.

La resta de credencials necessaris per el pipeline, com són *Github* o *Docker Hub*, també estan sota el govern de *Jenkins* dintre de la opció *Manage Credentials*.

8.4 Exemple de filosofía *DevSecOp* amb *Jenkins*

Emprant l'aplicació explicada en l'apartat "Explicació de la POC", farem un exercici de CI/CD emprant tècniques de seguretat.

La idea es realitzar les operatives més comunes dins d'un equip de desenvolupadors que volen implementar un flux de CI/CD basat en *Github* i *Jenkins* amb una filosofia de seguretat o *DevSecOp*.

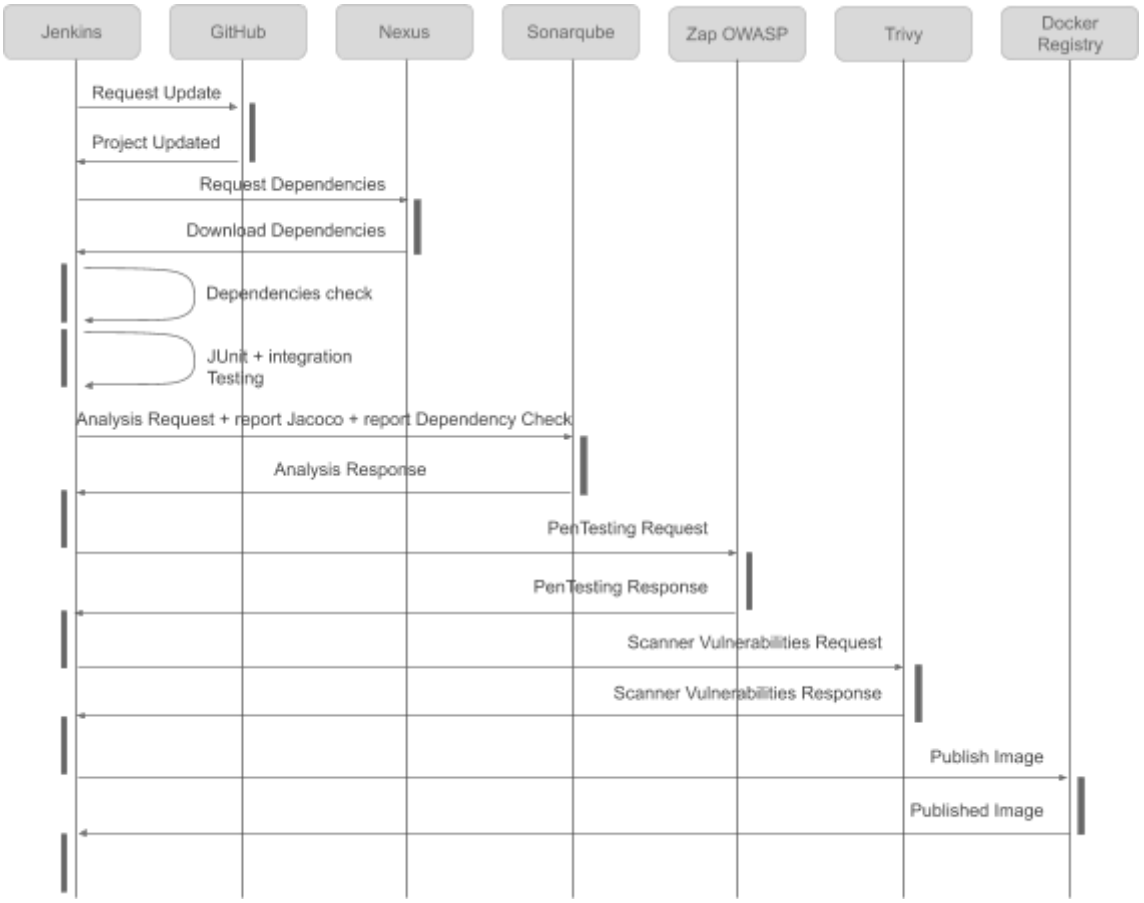
El primer pas es remarcar que aquest exemple continua el exemple de l'apartat "Exemple de filosofía DevOp amb Jenkins", però en aquesta ocasió ampliem el flux proposat de forma que queda:

- Descarrega de dependències.
- Escaneix de vulnerabilitats en les dependències del projecte.
- Tests unitaris.
- Test d'integració.
- Anàlisi de codi i reporting.
- Realització de *PenTesting*.
- Escaneix de vulnerabilitats de la imatge *Docker*.
- Publicació de la *Release Candidate*.
- Publicació de la versió.
- Deploy -desenvoluparem aquesta fase en propers capítols-.

En el nostre flux apliquem una filosofía basada en les tècniques SAST i DAST. El conjunt d'operatives que corresponen a cadascun són:

SAST	*DAST*
Escaneix de vulnerabilitats de dependències del projecte.	Realització de PenTesting.
Tests unitaris.	Escaneix de vulnerabilitats de la imatge Docker
Test d'integració	
Anàlisi de codi i reporting.	

El diagrama general del nostre flux quedaría com es mostra a continuació:



8.4.1 Escaneix de vulnerabilitats de dependències del projecte

El escaneig de vulnerabilitats d'un projecte queda fortament condicionat al llenguatge de programació emprat. En el nostre cas ens hem fet servir del projecte *OWASP Dependency-Check*[28] que disposa d'un plugin[29] d'integració amb el nostre projecte gestionat per *Maven* el qual ens permet realitzar un anàlisi exhaustiu de les dependències utilitzades. Aquest últim ens realitza un reporting de les seves conclusions per propers pasos valorar si prendre accions o no.

8.4.2 Tests unitaris i Test d'integració

Aquesta secció no disposa de cap tipus de novetat. Al igual que en exemple de l'apèndix anterior, utilitzem *Jacoco* per les seves mètriques i, al igual que en el punt anterior, deixem el report per més endavant valorar si prendre accions o no.

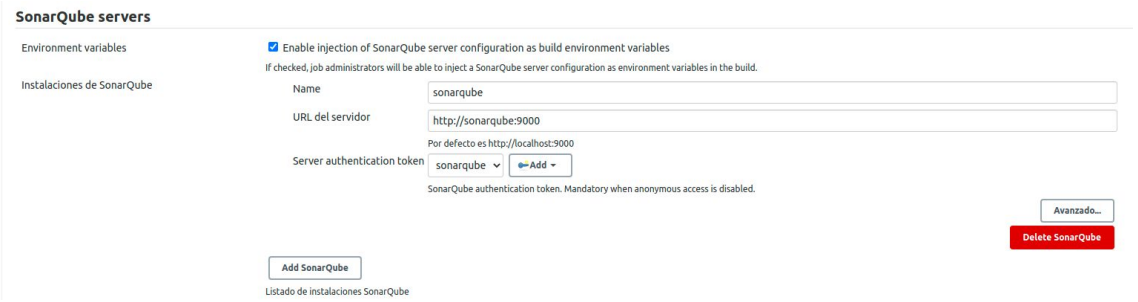
8.4.3 Anàlisi de codi i reporting

En aquesta secció necessitem una eina que ens analitza la qualitat del nostre codi -i si aquest- disposa d'alguna mala pràctica que vulneri la nostra aplicació. La opció més utilitzada -per ser la més completa- és Sonarqube[30]. Aquesta eina

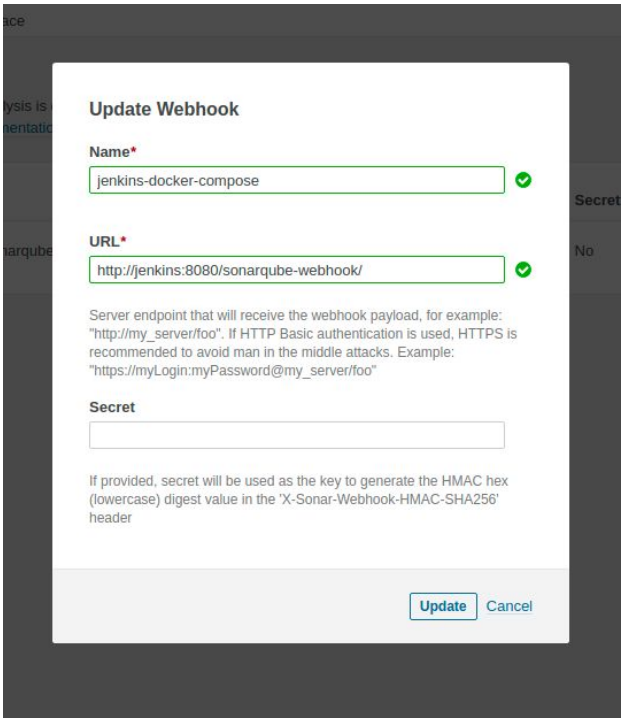
[28] https://owasp.org/www-project-dependency-check/
[29] https://jeremylong.github.io/DependencyCheck/dependency-check-maven/index.html
[30] https://www.sonarqube.org/

és capaç d'analitzar diversos llenguatges de programació i té un ampli conjunt de *plugins* que hi permeten diverses estratègies segons es necessiti. En el nostre cas definim *Sonarqube* com l'eina d'anàlisi i d'estimació del reporting generat per *OWASP Dependency-Check* i *Jacoco*. De manera que es valorarà si els resultats obtinguts de cobertura i de vulnerabilitats registrades a les nostres dependències de *Maven* són motiu de bloqueig del *Pipeline* o no.

En primer lloc necessitem relacionar el nostre servidor de *Jenkins* amb el nostre servidor de *Sonarqube*. Per aquest motiu declarem un nou *plugin,* el SonarQube Scanner[31], de *Jenkins* que permet identificar la localització del mateix.



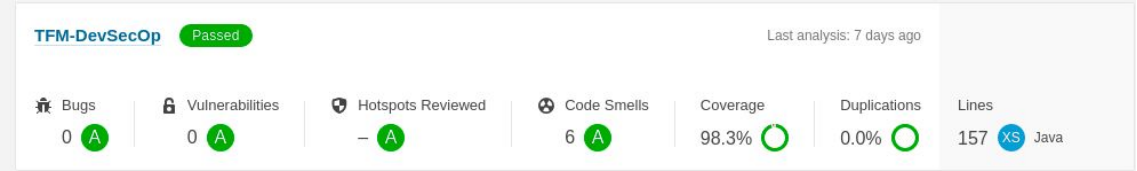
Per l'altre costat declarem la localització del servidor de *Jenkins* mitjançant un *WebHook* perquè Sonar sigui capaç d'indicar les conclusions a les que hi arriba.



Sonarqube per defecte incorpora integració amb els informes de cobertura de *Jacoco*. En canvi, no disposa per defecte l'extensió de *OWASP*

[31] https://plugins.jenkins.io/sonar/

Dependency-Check[32]. Afegim el *plugin* per integrar la funcionalitat. A continuació es mostra una captura de les mètriques d'un dels panells.



8.4.4 Realització de PenTesting

La realització del *PenTesting* es emprant un servei anomenat OWASP Zed Attack Proxy (Zap)[33]. Eina referència en la realització d'aquest tipus de tasques. Per poder-ne realitzar la invocació dins del *Pipeline* de *Jenkins* s'ha utilitzar la execució de comandes mitjançant del Zap-Cli[34] el qual ens facilita la interacció amb el servidor. Per tant, els passos realitzats són:

1. Disposició del servidor Zap per el port 8000
2. Creació de la imatge a realitzar-l'hi el *PenTesting*.
3. Arrancar la imatge a una xarxa definida amb el àlias "app".
4. Execució de la CLI de Zap per realitzar un escaneig amb un criteri definit per la branca en la que s'està efectuant el *PenTesting*.
5. Parem la imatge "app" per liberar recursos de la infraestructura.

Tots aquest pasos estan disponibles al fitxer *Jenkinsfile* dins el *stage* "Test the image (Pen Testing)"

8.4.5 Escaneix de vulnerabilitats de la imatge Docker

Un dels punts a contemplar es les vulnerabilitats a nivell de Sistema Operatiu. En el nostre cas, a nivell d'imatge del contenidor. Per tal d'evitar publicacions de imatges no adequades al *Docker Registry* es realitzarà una revisió de la imatge a registrar abans. L'eina triada es *Trivy*[35], realitza anàlisis de vulnerabilitats per determinar si la imatge creada es adecuada o no. Per tal de mantenir una filosofía dockeritzada, donarem access al *socket* de Docker a la imatge de *Trivy* de manera que permet realitzar interaccions amb el dimoni de *Docker* i, així, poder-ne realitzar l'anàlisi pertinent.

Al igual que en el cas del *PenTesting*, el anàlisi de vulnerabilitats queda condicionat la branca estem executant. El motiu és la relació d'entorns que hem predefinit inicialment. Amb aquesta tècnica tractem d'evitar disponibilitar vulnerabilitats a producció sense afectar al *delivery*.

[32] https://www.sonarplugins.com/owaspdependencycheck
[33] https://www.zaproxy.org/
[34] https://github.com/Grunny/zap-cli
[35] https://github.com/aquasecurity/trivy

```
raulsuarezdabo/tfm-devsecop-jenkins:latest (alpine 3.9.4)
==========================================================
Total: 4 (CRITICAL: 4)

+-------------+-----------------+----------+-------------------+---------------+--------------------------------+
|   LIBRARY   | VULNERABILITY ID | SEVERITY | INSTALLED VERSION | FIXED VERSION |             TITLE              |
+-------------+-----------------+----------+-------------------+---------------+--------------------------------+
| libbz2      | CVE-2019-12900  | CRITICAL | 1.0.6-r6          | 1.0.6-r7      | bzip2: out-of-bounds write in  |
|             |                 |          |                   |               | function BZ2_decompress        |
+-------------+-----------------+          +-------------------+---------------+--------------------------------+
| musl        | CVE-2019-14697  |          | 1.1.20-r4         | 1.1.20-r5     | musl libc through 1.1.23       |
|             |                 |          |                   |               | has an x87 floating-point      |
|             |                 |          |                   |               | stack adjustment imbalance,    |
|             |                 |          |                   |               | related...                     |
+-------------+                 +          +                   +               +                                |
| musl-utils  |                 |          |                   |               |                                |
|             |                 |          |                   |               |                                |
|             |                 |          |                   |               |                                |
|             |                 |          |                   |               |                                |
+-------------+-----------------+          +-------------------+---------------+--------------------------------+
| sqlite-libs | CVE-2019-8457   |          | 3.26.0-r3         | 3.28.0-r0     | sqlite: heap out-of-bound read |
|             |                 |          |                   |               | in function rtreenode()        |
+-------------+-----------------+----------+-------------------+---------------+--------------------------------+
[Pipeline] }
```

El *Pipeline* d'anàlisi queda inclòs dins del *stage* de "Publish Release" de forma que abortara l'execució si troba vulnerabilitats del nivell definit segons la branca seleccionada.

Finalment, tota la infraestructura necessària per executar aquesta Poc està inclosa dins del fitxer docker-compose a l'arrel del repositori.