



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación
Tesis Doctoral depositada en cumplimiento parcial de los requisitos para el
Título de Doctor por la Universitat Politècnica de València

**MoCIP: Un enfoque dirigido por modelos para el
aprovisionamiento de infraestructura en la nube**

Julio César Sandobalín Guamán

Director

Dr. Emilio Insfrán

Valencia, febrero de 2020

Tesis Doctoral

© **Julio César Sandobalín Guamán**, Valencia, España

MMXV-MMXX

Todos los derechos reservados en favor de sus respectivos propietarios.

Diseño de Portada: Mónica Sandobalín Guamán

Título: MoCIP: Un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube

Presentado por: Julio César Sandobalín Guamán
(julio.sandobalin@epn.edu.ec)

Director: Dr. Emilio Insfrán
(einsfran@dsic.upv.es)

Institución: Universitat Politècnica de València (UPV)

Departamento: Sistemas Informáticos y Computación (DSIC)

Programa de doctorado: Doctorado en Informática

Fecha de depósito: Valencia, noviembre de 2019

Defensa: Valencia, febrero de 2020

Comité de revisión: Dra. Marcela Genero Bocco, Universidad de Castilla-La Mancha, España
Dr. Juan de Lara Jaramillo, Universidad Autónoma de Madrid, España
Dr. Juan Manuel Murillo Rodríguez, Universidad de Extremadura, España

Tribunal:

Presidente:
Dr. Antonio Ruiz Cortés, Universidad de Sevilla, España

Secretario:
Dr. Juan de Lara Jaramillo, Universidad Autónoma de Madrid, España

Vocal:
Dr. Jesús García Molina, Universidad de Murcia, España

Agradecimientos

El camino de realización de una tesis doctoral resulta arduo, no siendo posible recorrerlo solo. A continuación, se citan a todas las personas que de alguna forma han contribuido en este trabajo.

A mi tutor de tesis, Emilio Insfrán por su confianza, apoyo y enseñanzas a lo largo de mis estudios de doctorado.

A Silvia Abrahão por su apoyo y enseñanzas sobre experimentos en Ingeniería de Software.

A mi madre, por todo su amor, paciencia, consejo y por ser mi inspiración de lucha y tenacidad para conseguir mis sueños.

A mi familia por todo su amor y apoyo durante todos estos años.

A mis amigos Carlos, Cecibel, Lenin, Nana, Jimena, Mabel, Ana, Ángel, Bea, José, Ximena, Patricia y Geovanny, por su amistad, apoyo, consejo, complicidad y sentido del humor que hicieron gratos e inolvidables todos estos años.

A la Escuela Politécnica Nacional y a mi país, por la confianza depositada en mí y por darme esta oportunidad que ha enriquecido mi vida en todo aspecto.

Y por último a todas las personas que de una u otra manera han contribuido a hacer posible este trabajo.

Resumen

DevOps (*Development & Operations*) es un nuevo movimiento que fomenta la colaboración entre los desarrolladores y el personal de operaciones a través de un conjunto de principios, prácticas y herramientas para optimizar el tiempo de entrega del software. En particular, la práctica del *despliegue continuo* de software es una gran fuente de problemas y genera mucha atención cuando los artefactos de software se entregan tarde o cuando un defecto crítico llega a producción. Al mismo tiempo, la práctica del despliegue continuo es la frontera entre los desarrolladores y el personal de operaciones en el ciclo de entrega del software. En consecuencia, se recomienda iniciar la implantación de DevOps con la práctica del despliegue de software. Para enfrentar este desafío, los profesionales e investigadores están utilizando la Infraestructura como Código (*Infrastructure as Code*, IaC), que es un enfoque para la automatización de la infraestructura basado en prácticas de desarrollo de software. El objetivo de IaC es definir en un script todas las instrucciones para crear, actualizar y ejecutar recursos de infraestructura. En este escenario, la automatización del aprovisionamiento de la infraestructura acelera la práctica del despliegue continuo en el ciclo de entrega del software.

La computación en la nube se ha convertido en el principal modelo de pago por uso utilizado por profesionales e investigadores para conseguir servicios en la nube en un corto período de tiempo. En este escenario, las compañías están pasando de generar potencia informática *in-house* hacia la obtención de recursos informáticos provistos en la nube a través de internet como servicios web. Al mismo tiempo, IaC y la computación en la nube están promoviendo algunos cambios en la industria. Por ejemplo, los equipos de operaciones pasan todo su tiempo trabajando en software, en lugar de configurar servidores y conectar cables de red. Existe una variedad de herramientas IaC que utilizan scripts para definir y ejecutar los recursos de infraestructura en diferentes proveedores de servicios IaaS (*Infrastructure as a Service*). Sin embargo, la diversidad de lenguajes de scripting de las herramientas IaC junto con la heterogeneidad del tipo de infraestructura que ofrece cada proveedor de servicios IaaS, han ocasionado que utilizar scripts IaC para el aprovisionamiento de infraestructura sea una actividad lenta y propensa a errores.

El objetivo del proyecto de doctorado es proponer una solución a la diversidad de los lenguajes de scripting —de las herramientas IaC— y a la heterogeneidad

del tipo de infraestructura que ofrece cada proveedor de servicios IaaS respecto al aprovisionamiento de infraestructura en la nube. Para afrontar estos desafíos se propone MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*).

MoCIP es un enfoque dirigido por modelos para el aprovisionamiento de la infraestructura en la nube que soporta IaC mediante la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering*, MDE). MoCIP utiliza los dos principios fundamentales de MDE: *abstracción* y *automatización*. En primer lugar, se desarrolló el lenguaje específico de dominio ArgonML para *abstraer* las capacidades de la computación en la nube, tales como cómputo, elasticidad, almacenamiento y redes. El lenguaje ArgonML permite modelar los recursos de infraestructura de la nube. En segundo lugar, se desarrolló la herramienta ARGON para *automatizar* el aprovisionamiento de infraestructura en la nube. ARGON realiza transformaciones de modelo a modelo para generar modelos que representan la infraestructura de diferentes proveedores de servicios IaaS. Además, ARGON realiza transformaciones de modelo a texto para generar scripts IaC con la información subyacente de cada proveedor de servicios IaaS y de cada herramienta de aprovisionamiento de infraestructura.

Con el fin de investigar la interacción de MoCIP en un contexto de aplicación se diseñaron y ejecutaron un conjunto de experimentos controlados. En primer lugar, se evaluó la percepción de los usuarios al utilizar MoCIP. Se encontró evidencia empírica para afirmar que los participantes percibieron que MoCIP es fácil de usar y útil para realizar el aprovisionamiento de infraestructura en Amazon Web Services. Además, los participantes tienen la intención de utilizar MoCIP en el futuro. En segundo lugar, se evaluó el rendimiento y la percepción de los usuarios al comparar la herramienta ARGON versus la herramienta Ansible respecto a la definición de la infraestructura de Amazon Web Services. Por un lado, se encontró evidencia empírica para afirmar que ARGON es más efectivo y eficiente que Ansible para especificar los recursos de infraestructura. Por otro lado, se encontró evidencia empírica para afirmar que los participantes percibieron que ARGON es más fácil de usar y útil que Ansible para definir la infraestructura. Además, los participantes tienen la intención de usar ARGON en el futuro.

DevOps (*Development & Operations*) is a new movement that encourages collaboration between developers and operation staff through a set of principles, practices, and tools in order to optimize the software delivery time. In particular, *continuous deployment* is a software practice, which is a great source of problems and generates a lot of attention when software artifacts are delivered late, or a critical defect comes to production. At the same time, continuous deployment is the border between developers and operation staff in the software delivery cycle. Consequently, it is recommended to start the DevOps implantation with continuous deployment practice. To address this challenge, practitioners and researchers are using Infrastructure as Code (IaC), which is an approach to infrastructure automation based on software development practices. The aim of IaC is to define in a script all the instructions to create, update and execute infrastructure resources. In this scenario, infrastructure provisioning automation accelerates the continuous deployment in the software delivery cycle.

Cloud computing has become the principal pay-as-you-go model used by practitioners and researchers to obtain cloud services in a short period of time. In this scenario, companies are moving from generating *in-house* computing power into obtaining computing resources provided by cloud computing through the Internet as web services. At the same time, IaC and cloud computing are promoting some changes in the industry. For instance, operation teams spend all their time working on software, instead of racking servers and plugging in network cables. There is a diversity of IaC tools that use scripts to define and execute infrastructure resources in different IaaS (*Infrastructure as a Service*) service providers. However, the diversity of scripting languages of IaC tools along with the heterogeneity of the type of infrastructure offered by each IaaS service provider have caused that the use of IaC scripts for infrastructure provisioning to be a time-consuming and error-prone activity.

The aim of the Ph.D. project is to propose a solution to the diversity of scripting languages of IaC tools along with the heterogeneity of the type of infrastructure offered by each IaaS service provider regarding the cloud infrastructure provisioning. To face these challenges, a Model-Driven Approach to Cloud Infrastructure Provisioning (MoCIP) is proposed.

MoCIP is a model-driven approach to cloud infrastructure provisioning that supports IaC through Model-Driven Engineering (MDE). MoCIP uses the fundamental principles of MDE: *abstraction* and *automation*. First, the domain-

specific language called ArgonML was developed to *abstract* the capabilities of cloud computing such as computing, elasticity, storage, and networking. The ArgonML language allows modeling the cloud infrastructure. Second, the ARGON tool *automates* the cloud infrastructure provisioning. ARGON performs model-to-model transformations to generate models that represent the infrastructure of different IaaS service providers. In addition, ARGON performs model-to-text transformations to generate IaC scripts with the underlying information of the IaaS service provider along with the infrastructure provisioning tool.

In order to investigate the interaction of MoCIP in an application context, a set of controlled experiments were designed and executed. First, the users' perception when using MOCIP was evaluated. Empirical evidence was found to state that participants perceived that MOCIP is easy to use and useful for infrastructure provisioning in Amazon Web Services. In addition, participants have the intention to use MoCIP in the future. Secondly, the users' performance and perception were evaluated when comparing the ARGON tool versus the Ansible tool regarding the infrastructure definition of the Amazon Web Services. On the one hand, empirical evidence was found to state that ARGON is more effective and efficient than Ansible to specify infrastructure resources. On the other hand, empirical evidence was found to state that participants perceived that ARGON is easier to use and useful than Ansible to define infrastructure. In addition, participants have the intention to use ARGON in the future.

DevOps (*Development & Operations*) és un nou moviment que fomenta la col·laboració entre els desenvolupadors i el personal d'operacions a través d'un conjunt de principis, pràctiques i eines per a millorar el temps de lliurament del programari. En particular, la pràctica del desplegament continu de programari és una gran font de problemes i genera molta atenció quan els artefactes de programari s'entreguen tard o quan un defecte crític arriba a producció. Al mateix temps, la pràctica del desplegament continu és la frontera entre els desenvolupadors i el personal d'operacions en el cicle de lliurament del programari. En conseqüència, es recomana iniciar la implantació de DevOps amb la pràctica del desplegament de programari. Per a enfrontar aquest desafiament, els professionals i investigadors estan utilitzant la Infraestructura com a Codi (*Infrastructure as Code*, IaC), que és un enfocament per a l'automatització de la infraestructura basat en pràctiques de desenvolupament de programari. L'objectiu de IaC és definir en un script totes les instruccions per a crear, actualitzar i executar recursos d'infraestructura. En aquest escenari, l'automatització de l'aprovisionament de la infraestructura accelera la pràctica del desplegament continu en el cicle de lliurament del programari.

La computació en el núvol s'ha convertit en el principal model de pagament per ús utilitzat per professionals i investigadors per a aconseguir serveis en el núvol en un curt període de temps. En aquest escenari, les companyies estan passant de generar potència informàtica *in-house* cap a l'obtenció de recursos informàtics proveïts en el núvol a través d'internet com a serveis web. Al mateix temps, IaC i la computació en el núvol estan promovent alguns canvis en la indústria, per exemple, els equips d'operacions passen tot el seu temps treballant en programari, en lloc de configurar servidors i connectar cables de xarxa. Existeix una varietat d'eines IaC que utilitzen scripts per a definir i executar els recursos d'infraestructura en diferents proveïdors de serveis IaaS (*Infrastructure as a Service*). No obstant això, la diversitat de llenguatges de scripting de les eines IaC juntament amb l'heterogeneïtat del tipus d'infraestructura que ofereix cada proveïdor de serveis IaaS, han ocasionat que utilitzar scripts IaC per a l'aprovisionament d'infraestructura siga una activitat lenta i propensa a errors.

L'objectiu del projecte de doctorat és proposar una solució a la diversitat dels llenguatges de scripting —de les eines IaC— i a l'heterogeneïtat del tipus d'infraestructura que ofereix cada proveïdor de serveis IaaS respecte a l'aprovisionament d'infraestructura en el núvol. Per a afrontar aquests

desafiaments es proposa MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*).

MoCIP és un enfocament dirigit per models per a l'aprovisionament de la infraestructura en el núvol que suporta IaC mitjançant l'Enginyeria de Programari Dirigida per Models (*Model-Driven Engineering*, MDE). MoCIP utilitza els dos principis fonamentals de MDE: *abstracció* i *automatització*. En primer lloc, es va desenvolupar el llenguatge específic de domini ArgonML per a *abstraure* les capacitats de la computació en el núvol tals com còmput, elasticitat, emmagatzematge i xarxes. El llenguatge ArgonML permet modelar els recursos d'infraestructura del núvol. En segon lloc, es va desenvolupar l'eina ARGON per a *automatitzar* l'aprovisionament d'infraestructura en el núvol. ARGON realitza transformacions de model a model per a generar models que representen la infraestructura de diferents proveïdors de serveis IaaS. A més, ARGON realitza transformacions de model a text per a generar scripts IaC amb la informació subjacent de cada proveïdor de serveis IaaS i de cada eina d'aprovisionament d'infraestructura.

Amb la finalitat d'investigar la interacció de MoCIP en un context d'aplicació es van dissenyar i van executar un conjunt d'experiments controlats. En primer lloc, es va avaluar la percepció dels usuaris en utilitzar MoCIP. Es va trobar evidència empírica per a afirmar que els participants van percebre que MoCIP és fàcil d'usar i útil per a realitzar l'aprovisionament d'infraestructura en Amazon Web Services. A més, els participants tenen la intenció d'utilitzar MoCIP en el futur. En segon lloc, es va avaluar el rendiment i la percepció dels usuaris en comparar l'eina ARGON versus l'eina Ansible respecte a la definició de la infraestructura de Amazon Web Services. D'una banda, es va trobar evidència empírica per a afirmar que ARGON és més efectiu i eficient que Ansible per a especificar els recursos d'infraestructura. D'altra banda, es va trobar evidència empírica per a afirmar que els participants van percebre que ARGON és més fàcil d'usar i útil que Ansible per a definir la infraestructura. A més, els participants tenen la intenció d'usar ARGON en el futur.

Contenido

Capítulo 1	Introducción	1
1.1	Motivación	2
1.2	Planteamiento del problema	5
1.3	Solución propuesta	7
1.4	Contexto de la investigación	9
1.5	Contribuciones	10
1.5.1	Soporte a la infraestructura como código utilizando MDE	10
1.5.2	Metamodelos de infraestructura y sus mapeos	10
1.5.3	Un pipeline para el aprovisionamiento de infraestructura	11
1.5.4	Evidencia empírica sobre MoCIP y ARGON	11
1.6	Publicaciones	11
1.6.1	En revistas científicas	11
1.6.2	En conferencias internacionales	12
1.6.3	En conferencias nacionales de España	12
1.6.4	En un Blog científico divulgativo	12
1.7	Estructura de la tesis	13
Capítulo 2	Metodología de Investigación	15
2.1	Proyecto <i>Design Science</i>	16
2.2	Objetivos de investigación	18
2.2.1	Objetivo de diseño del artefacto	19
2.2.2	Objetivos de conocimiento	19
2.2.3	Objetivos de diseño de los instrumentos	20
2.2.4	Objetivos de predicción	20
2.3	Preguntas de investigación	21
2.4	Ciclos de ingeniería, diseño y empírico	23
2.5	Resumen	28
Capítulo 3	Marco Conceptual	31
3.1	Conceptos básicos	32
3.2	Un marco conceptual para la Computación en la Nube	33
3.3	Un marco conceptual para DevOps y la Infraestructura como Código	35
3.4	Un marco conceptual para la Ingeniería de Software Dirigida por Modelos	36
3.5	Un marco conceptual para experimentos en Ingeniería de Software	38
Capítulo 4	Estado del Arte	41
4.1	Librerías y herramientas IaC	42
4.2	Trabajos de investigación que utilizan IaC y MDE	44
4.3	Mapeo sistemático sobre IaC	47
4.3.1	Categorización de las publicaciones relacionadas con IaC	50
4.3.1.1	Herramientas o marcos para IaC	50
4.3.1.2	Adopción de IaC	51
4.3.1.3	Estudios empíricos relacionados con IaC	51

4.3.1.4	Pruebas con IaC.....	52
4.4	Resumen	53
Capítulo 5 El Enfoque MoCIP		55
5.1	¿Por qué un enfoque dirigido en modelos?	56
5.2	Requisitos para el enfoque MoCIP	58
5.3	MoCIP: Un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube.....	59
5.3.1	Elicitación de requisitos	60
5.3.2	Modelado de infraestructura	61
5.3.3	Aprovisionamiento de infraestructura.....	62
5.4	Soporte de metamodelos para MoCIP	63
5.4.1	Metamodelo abstracto de infraestructura	64
5.4.2	Metamodelo de Amazon Web Services	65
5.4.3	Metamodelo de Microsoft Azure	67
5.5	Resumen	69
Capítulo 6 Soporte de Herramientas		71
6.1	El lenguaje de modelado ArgonML.....	72
6.1.1	Sintaxis abstracta	72
6.1.2	Sintaxis concreta.....	75
6.2	La herramienta ARGON	78
6.2.1	La arquitectura de ARGON.....	79
6.2.1.1	Requerimientos	80
6.2.1.2	Modelo independiente de la plataforma (PIM).....	80
6.2.1.3	Modelo específico de la plataforma (PSM).....	81
6.2.1.4	Instancias.....	83
6.2.2	Generación de modelos de infraestructura.....	83
6.2.2.1	Transformaciones M2M para un Máquina Virtual	84
6.2.2.2	Transformaciones M2M para un Balanceador de Carga	85
6.2.2.3	Transformaciones M2M para un escalado automático	86
6.2.2.4	Módulo de transformaciones M2M	88
6.2.3	Generación de scripts de aprovisionamiento	89
6.3	Un pipeline para el aprovisionamiento de infraestructura en la nube	91
6.3.1	Proyecto de infraestructura	93
6.3.2	Gestión de la configuración	94
6.3.2.1	Sistema de control de versiones.....	95
6.3.2.2	Repositorio de artefactos	96
6.3.3	Integración continua	97
6.3.3.1	Generación de scripts	98
6.3.3.2	Conjunto de pruebas automatizadas para los scripts.....	99
6.3.4	Despliegue continuo	100
6.3.4.1	Conjunto de pruebas automatizadas para la infraestructura	102
6.4	Resumen	103
Capítulo 7 Validación de MoCIP		107
7.1	MoCIP.....	108
7.1.1	Elicitación de requisitos	108
7.1.1.1	La universidad CEC	108
7.1.1.2	La empresa MODAFIN.....	109

7.1.2	Modelo de infraestructura de Amazon.....	109
7.1.3	Aprovisionamiento de infraestructura.....	110
7.2	Experimentos controlados	112
7.2.1	Objetivo	112
7.2.2	Selección del contexto.....	113
7.2.2.1	Objetos experimentales	113
7.2.2.2	MoCIP y su soporte de herramientas	114
7.2.2.3	Selección de participantes	114
7.2.3	Diseño de los experimentos individuales.....	114
7.2.3.1	Diseño del experimento de línea base	115
7.2.3.1.1	Selección del contexto.....	115
7.2.3.1.2	Participantes	117
7.2.3.1.3	Selección de variables	117
7.2.3.1.4	Formulación de hipótesis.....	120
7.2.3.1.5	Diseño	120
7.2.3.1.6	Operación	121
7.2.3.2	Segundo experimento (UPV2)	122
7.2.4	Tareas experimentales y materiales	123
7.3	Resultados.....	125
7.3.1	Estadística descriptiva y análisis de datos exploratorios	125
7.3.2	Prueba de hipótesis	127
7.3.2.1	Respondiendo las preguntas de investigación	128
7.4	Amenazas a la validez del experimento.....	129
7.4.1	Validez de la conclusión.....	129
7.4.2	Validez interna	130
7.4.3	Validez del constructo.....	130
7.4.4	Validez externa	130
7.5	Conclusiones.....	131

Capítulo 8 Validación de ARGON.....133

8.1	Herramientas de aprovisionamiento de infraestructura	134
8.1.1	Ansible: Una herramienta centrada en el código.....	134
8.1.2	ARGON: Una herramienta dirigida por modelos.....	136
8.1.3	Comparación de herramientas	138
8.2	La familia de experimentos.....	139
8.2.1	Objetivo	139
8.2.2	Selección del contexto.....	140
8.2.2.1	Objetos experimentales	140
8.2.2.2	Herramientas IaC.....	141
8.2.2.3	Selección de participantes	141
8.2.3	Diseño de experimentos individuales.....	142
8.2.3.1	Experimento de línea base	143
8.2.3.1.1	Selección del contexto.....	143
8.2.3.1.2	Participantes	145
8.2.3.1.3	Selección de variables	145
8.2.3.1.4	Formulación de hipótesis.....	148
8.2.3.1.5	Diseño	148
8.2.3.1.6	Operación	150
8.2.3.2	Segundo experimento (UPV2)	151
8.2.3.3	Tercer experimento (UPV3)	151
8.2.4	Tareas experimentales y materiales	152

8.2.5	Análisis de los datos de los experimentos y metaanálisis	154
8.3	Resultados	155
8.3.1	Estadística descriptiva y análisis de datos exploratorio	155
8.3.2	Prueba de hipótesis	162
8.3.3	Análisis de datos de la familia de experimentos	166
8.3.3.1	Metaanálisis	167
8.3.3.2	Respondiendo las preguntas de investigación	169
8.4	Amenazas a la validez	173
8.4.1	Validez de la conclusión	173
8.4.2	Validez interna	174
8.4.3	Validez del constructo	175
8.4.4	Validez externa	175
8.5	Conclusiones	176
Capítulo 9	Conclusiones y Trabajos Futuros.....	179
9.1	Conclusiones	180
9.1.1	Objetivo de conocimiento sobre IaC y MDE.....	180
9.1.2	Objetivo de diseño del artefacto MoCIP	182
9.1.3	Objetivo de conocimiento sobre la investigación de MoCIP	182
9.2	Trabajos futuros	185
9.2.1	Ampliar el soporte de MoCIP.....	185
9.2.2	Ampliar el soporte de las herramientas MDE	186
9.2.3	Mantenibilidad de los metamodelos de infraestructura.....	186
9.2.4	Pruebas a los modelos de infraestructura	186
9.2.5	Implantación de MoCIP	187
9.2.6	Experimentos controlados	187
Bibliografía.....	189

Índice de Figura

Figura 1-1 Interés en "DevOps" como palabra de búsqueda en Google Trends	2
Figura 1-2 Interés en "Infrastructure as Code" como frase de búsqueda en Google Trends	4
Figura 2-1 Proyecto <i>Design Science</i> para el enfoque MoCIP	16
Figura 2-2 Marco para el proyecto <i>Design Science</i> para el enfoque MoCIP	18
Figura 2-3 Estructura de los objetivos de investigación del proyecto de doctorado	19
Figura 2-4 Ciclo de diseño del proyecto de doctorado	24
Figura 2-5. Ciclo empírico para validar MoCIP	25
Figura 2-6 Ciclo empírico para valida la herramienta ARGON	26
Figura 2-7 Ciclo de ingeniería, ciclo de diseño y ciclos empíricos aplicados a la tesis	28
Figura 5-1 El enfoque MoCIP para el aprovisionamiento de infraestructura en la nube	60
Figura 5-2 Metamodelo abstracto de infraestructura	64
Figura 5-3 Metamodelo de Amazon Web Services	66
Figura 5-4 Metamodelo de Microsoft Azure	67
Figura 6-1 Sintaxis abstracta del lenguaje ArgonML	74
Figura 6-2 Extracto de la definición de la sintaxis concreta de lenguaje ArgonML	77
Figura 6-3 Sintaxis concreta del lenguaje ArgonML	78
Figura 6-4 Arquitectura de la herramienta ARGON	80
Figura 6-5 Transformaciones M2M para una máquina virtual	84
Figura 6-6 Transformaciones M2M para un balanceador de carga	85
Figura 6-7 Transformaciones M2M para un escalado automático	87
Figura 6-8 Extracto del módulo de transformaciones M2M amazon2azure.atl	88
Figura 6-9 Plantilla de transformación M2T para Ansible	90
Figura 6-10 Plantilla de transformación M2T para Terraform	90
Figura 6-11 Descripción general del <i>pipeline</i> de aprovisionamiento de infraestructura	92
Figura 6-12 Proyecto de infraestructura creado por la herramienta ARGON	93
Figura 6-13 Extracto de código del archivo POM del proyecto de infraestructura	94
Figura 6-14 Descripción general del control de versiones con GitHub	95
Figura 6-15 Proyecto de infraestructura en GitHub	96
Figura 6-16 Artefactos del proyecto de infraestructura en Nexus	96
Figura 6-17 Descripción general del repositorio de artefactos	97
Figura 6-18 Descripción general del pipeline de CI	98
Figura 6-19 Extracto de código del archivo POM para el motor de transformaciones M2T ...	98
Figura 6-20 Panel de proyectos del servidor Jenkins	99
Figura 6-21 Pruebas de verificación de sintaxis de un script de aprovisionamiento	100
Figura 6-22 Pruebas de comprobación estática de un script de aprovisionamiento	100
Figura 6-23 Descripción general del despliegue continuo de infraestructura	101
Figura 6-24 Máquina virtual (<i>webservice</i>) en Amazon Web Services	101
Figura 6-25 Pruebas en la infraestructura creada en Amazon Web Services	103
Figura 7-1 Diagrama de infraestructura modelado con el lenguaje ArgonML	110
Figura 7-2 Extracto del script de aprovisionamiento para la herramienta Ansible	111
Figura 7-3 Extracto de los mensajes de la consola de la herramienta Ansible	111
Figura 7-4 Verificación del funcionamiento de una máquina virtual	112
Figura 7-5 Resumen de los experimentos controlados	115
Figura 7-6 Proceso de operación del experimento de línea base	121

Figura 7-7 Diagrama de caja para las variables basadas en la percepción.....	127
Figura 8-1 Un <i>Playbook</i> de Ansible para la Empresa MODAFIN.....	135
Figura 8-2 Un modelo de infraestructura de ARGON para la Universidad CEC	137
Figura 8-3 Resumen de la familia de experimentos	142
Figura 8-4 Resumen del proceso de operación del experimento de línea base	150
Figura 8-5 Diagramas de caja para la efectividad del tratamiento	156
Figura 8-6 Diagramas de caja para la eficiencia del tratamiento	157
Figura 8-7 Diagramas de caja para la efectividad de la secuencia.....	158
Figura 8-8 Diagramas de caja para la eficiencia de la secuencia	159
Figura 8-9 Diagramas de caja para la efectividad del período.....	160
Figura 8-10 Diagramas de caja para la eficiencia del período	161
Figura 8-11 Diagramas de caja para las variables basadas en la percepción	161
Figura 8-12 Diagrama de bosque para el metaanálisis de la familia de experimentos	169

Índice de Tablas

Tabla 4-1 Lista de publicaciones que utilizan IaC y MDE	45
Tabla 4-2 Lista de publicaciones del mapeo sistemático (Rahman <i>et al.</i> , 2019)	48
Tabla 4-3 Mapeo entre tema y publicación (Rahman <i>et al.</i> , 2019).....	50
Tabla 6-1 Conceptos de modelado de ArgonML respecto a la capacidad de cómputo	73
Tabla 6-2 Conceptos de modelado respecto a la capacidad de cómputo y sus elementos gráficos	75
Tabla 6-3 Conceptos de modelado respecto a la capacidad de almacenamiento y sus elementos gráficos	76
Tabla 6-4 Conceptos de modelado respecto a la capacidad de elasticidad y sus elementos gráficos	76
Tabla 7-1 Requisitos para aprovisionar la infraestructura en Amazon Web Services	115
Tabla 7-2 Resumen de variables dependientes	118
Tabla 7-3 Ítems para medir las variables basadas en la percepción	119
Tabla 7-4 Diseño del experimento de línea base.....	121
Tabla 7-5 Resumen de estadísticas descriptivas.	125
Tabla 7-6 Resultados del aprovisionamiento de infraestructura en Amazon Web Services	126
Tabla 7-7 Resumen de estadísticas inferencial.....	128
Tabla 8-1 Comparación entre los elementos de infraestructura de ARGON y los módulos de Ansible	138
Tabla 8-2 Requisitos para definir los recursos de infraestructura de la nube.....	144
Tabla 8-3 Resumen de las variables dependientes de la familia de experimentos	146
Tabla 8-4 Ítems para mediar las variables basadas en la percepción	147
Tabla 8-5 Diseño cruzado AB/BA para configura el experimento de línea base	149
Tabla 8-6 Estadísticas descriptivas de la familia de experimentos	155
Tabla 8-7 Resumen de la prueba Shapiro-Wilk para los residuos del LMM.....	163
Tabla 8-8 Prueba Tipo III de Efectos Fijos para la Efectividad	163
Tabla 8-9 Prueba Tipo III de Efectos Fijos para la Eficiencia	164
Tabla 8-10 Resumen de las pruebas Alfa de Cronbach	164
Tabla 8-11 Prueba Tipo III de Efectos Fijos para la PEOU	165
Tabla 8-12 Prueba Tipo III de Efectos Fijos para la PU	165
Tabla 8-13 Prueba Tipo III de Efectos Fijos para la ITU	166
Tabla 8-14 Resumen de la <i>d</i> de Cohen para las variables dependientes de los experimentos..	167
Tabla 8-15 Resumen del resultado de las hipótesis de la familia de experimentos	170

Acrónimos

ARGON	<i>An infrastructure modelinG tool for clOud provisioniNg</i> / Una herramienta de modelado de infraestructura para el aprovisionamiento en la nube
ArgonML	<i>ARGON Modeling Language</i> / Lenguaje de modelado de ARGON
CD	<i>Continuous Deployment</i> / Despliegue Continuo
CI	<i>Continuous Integration</i> / Integración Continua
CIM	<i>Computation Independent Model</i> / Modelo Independiente de Computación
DevOps	<i>Development & Operations</i> / Desarrollo y Operaciones
EMF	<i>Eclipse Modeling Framework</i>
IaaS	<i>Infrastructure as a Service</i> / Infraestructura como un Servicio
IaC	<i>Infrastructure as Code</i> / Infraestructura como Código
KP	<i>Knowledge Question</i> / Pregunta de Conocimiento
JAR	<i>Java Archive</i> / Archivo Java
M2M	<i>Model to Model</i> / Modelo a Modelo
M2T	<i>Model to Text</i> / Modelo a Texto
MDA	<i>Model-Driven Architecture</i> / Arquitectura Dirigida por Modelos
MDE	<i>Model-Driven Engineering</i> / Ingeniería Dirigida por Modelos
MEM	<i>Method Evaluation Model</i> / Método para la Evaluación de Modelos
MoCIP	<i>A Model-Driven Approach to Cloud Infrastructure Provisioning</i> / Un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube
PaaS	<i>Platform as a Service</i> / Plataforma como un Servicio
PIM	<i>Platform Independent Model</i> / Modelo Independiente de Plataforma
PSM	<i>Platform Specific Model</i> / Modelo Específico de Plataforma
RQ	<i>Research Question</i> / Pregunta de Investigación
SaaS	<i>Software as a Service</i> / Software como un Servicio
TAM	<i>Technology Acceptance Model</i> / Modelo de Aceptación Tecnológica

TI	Tecnología de la Información
TRP	<i>Technical Research Problem</i> / Problema de Investigación Técnica
XML	<i>eXtensible Markup Language</i> / Lenguaje de Marcas Extensible

Capítulo 1

Introducción

En este capítulo se contextualiza el trabajo de investigación realizado en esta tesis. En primer lugar, se presenta las particularidades de la automatización de la infraestructura en la computación en la nube. A continuación, se plantea el problema a resolver y, por último, se describe la solución propuesta.

En la Sección 1.1 se presenta una introducción a DevOps y a la Infraestructura como Código (IaC). Además, se presenta la motivación a la automatización de la infraestructura en el despliegue de aplicaciones software.

En la Sección 1.2 se presenta el planteamiento del problema sobre el aprovisionamiento de infraestructura en la nube.

En la Sección 1.3 se presenta un resumen de la solución propuesta basada en un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube.

En la Sección 1.5 se presentan las contribuciones de la tesis.

En la Sección 1.6 se presentan las publicaciones que soportan los resultados de la investigación en el aprovisionamiento de infraestructura en la nube.

En la Sección 1.7 se presenta la estructura de la tesis junto con un resumen de cada capítulo.

1.1 Motivación

En muchas compañías, uno de los desafíos más críticos es cómo entregar una nueva idea o artefacto de software a los clientes lo más rápido posible. Además, como los requisitos y los plazos de entrega cambian constantemente debido principalmente al tiempo de comercialización, la información entre el equipo de desarrollo y el personal de operaciones debe ser precisa, estar fácilmente disponible, ser fácil de encontrar e, idealmente, entregarse de forma continua. Para enfrentar a estos desafíos, un nuevo movimiento denominado DevOps (*Development & Operations*) está promoviendo una colaboración entre los desarrolladores y el personal de operaciones a través de un conjunto de principios, prácticas y herramientas para optimizar el tiempo de entrega del software (Humble *et al.*, 2010). En este escenario, DevOps significa una transformación significativa en la cultura de TI (*Tecnología de la Información*), centrándose en la entrega rápida de los servicios de TI a través de la adopción de metodologías ágiles y prácticas Lean en el contexto de un enfoque orientado al sistema (Gartner, 2018).

DevOps se ha convertido en una palabra de moda, como muestran los datos de Google Trends relacionados con el término de búsqueda "*DevOps*". La Figura 1-1 presenta la evidencia —a partir de 2014— sobre el interés creciente en DevOps. El término DevOps fue acuñado por John Allspaw y Paul Hammond (empleados de Flickr) en 2009 en la O'Reilly Velocity Conference, y desde entonces, DevOps aboga porque el desarrollo de software y las actividades de operaciones sean compactas, transparentes y totalmente integradas.

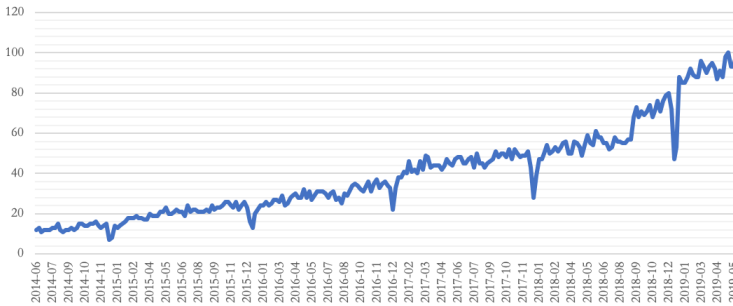


Figura 1-1 Interés en "*DevOps*" como palabra de búsqueda en Google Trends

Compañías como Puppet (Puppet Labs, 2005) y Splunk (Baum *et al.*, 2003) han encuestado a más de 30 000 profesionales técnicos en todo el mundo para explorar las relaciones entre el desempeño de TI, las prácticas de DevOps, la cultura y otros elementos que afectan los resultados comerciales de las compañías. El resultado de la encuesta es el Informe del Estado de DevOps de

2018 (Mann *et al.*, 2018). Este informe describe la automatización de la infraestructura como una etapa en la evolución hacia DevOps: “*La automatización de las tareas de aprovisionamiento de la infraestructura y de la configuración de los sistemas operativos es un resultado de alta prioridad*” (Mann *et al.*, 2018). Por un lado, la automatización del aprovisionamiento de infraestructura mejora la capacidad de implementar aplicaciones de software. Por otro lado, la automatización de la configuración de los sistemas operativos ayuda al personal de operaciones a preparar y entregar sistemas que coincidan en diferentes entornos.

Las prácticas de DevOps más utilizadas por profesionales e investigadores son la *integración* y la *entrega continuas*. Estas prácticas garantizan una entrega de software oportuna y de alta calidad. La *integración continua* es una práctica de integrar y realizar pruebas con frecuencia en todos los cambios de un sistema a medida que se desarrollan los artefactos de software, mientras que la *entrega continua* garantiza que todos los artefactos de software, sistemas e infraestructura se validen continuamente para garantizar que estén listos antes de pasar a la etapa de producción (Morris, 2016). En cambio, un *despliegue continuo* implica realizar automáticamente las pruebas correspondientes sobre los cambios de software incrementales y con frecuencia implementarlos en los entornos de producción (Parnin *et al.*, 2017). En este contexto, la entrega continua permite que un artefacto de software esté en un estado viable para su implementación todo el tiempo, mientras que el despliegue continuo es un proceso automático para colocar una versión del software en el entorno de producción.

Particularmente, el despliegue continuo está causando un cambio en gran parte del panorama de la ingeniería de software. Para estudiar este cambio, varios investigadores participaron en la Cumbre de Despliegue Continuo en el campus de Facebook en julio de 2015 (Parnin *et al.*, 2017). A esta cumbre asistieron compañías destacadas como Cisco, Facebook, Google, LexisNexis, Microsoft, IBM, Mozilla, Netflix, Red Hat y SAS. Los resultados de la cumbre establecieron enfoques y creencias para guiar las actividades del despliegue continuo y establecer objetivos tangibles para la validación empírica por parte de la comunidad científica. Adicionalmente, los investigadores informaron que el *despliegue de software* es una gran fuente de problemas y generan mucha atención cuando los artefactos de software se entregan tarde o cuando un defecto crítico llega a producción, y los clientes lo notan (Mann *et al.*, 2018).

En consecuencia, debido a que el proceso de despliegue de software es la frontera entre los desarrolladores y el personal de operaciones en el ciclo de entrega del software, se recomienda iniciar la implantación de DevOps con el proceso de despliegue de software (Mann *et al.*, 2018). En este escenario, la automatización del aprovisionamiento de la infraestructura acelera el proceso de

despliegue en el ciclo de entrega del software. Para enfrentar este desafío, los profesionales e investigadores están utilizando la Infraestructura como Código (*Infrastructure as Code*, IaC), que es un enfoque para la automatización de la infraestructura basada en prácticas de desarrollo de software (Morris, 2016).

El enfoque IaC tiene como objetivo definir en un script todas las instrucciones para crear, actualizar y ejecutar recursos de infraestructura. El término en inglés “*Infrastructure as Code*” fue acuñado por Mark Burgess, quien creó la herramienta CFEngine a principios de la década de 1990 (Johann, 2017). El enfoque IaC también se ha convertido en una práctica de moda, como muestran los datos de Google Trends relacionados con la búsqueda del término “*Infrastructure as Code*”. La Figura 1-2 muestra la evidencia —a partir de 2014— sobre el interés creciente en el enfoque IaC.

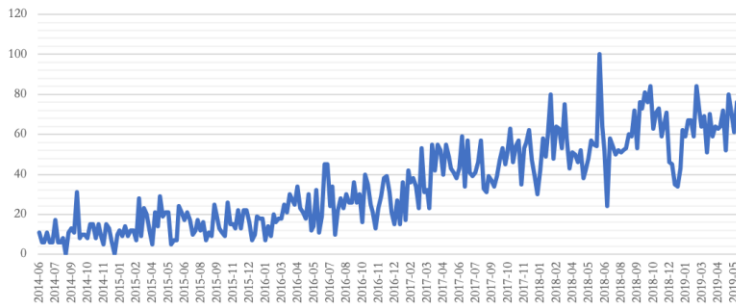


Figura 1-2 Interés en “*Infrastructure as Code*” como frase de búsqueda en Google Trends

Los principios del enfoque IaC (Morris, 2016) ofrecen muchas ventajas para definir, actualizar y ejecutar recursos de infraestructura, tales como:

- **Los sistemas se pueden reproducir fácilmente.** Es posible reconstruir de manera fácil y confiable cualquier elemento de la infraestructura.
- **Los sistemas son desechables.** La infraestructura se puede crear, destruir, reemplazar, redimensionar y mover fácilmente.
- **Los sistemas son consistentes.** Es posible implementar el principio de reproducibilidad para construir rápidamente múltiples elementos idénticos de infraestructura.

Por otra parte, la computación en la nube se ha convertido en el principal modelo de pago por uso que utilizan los profesionales e investigadores para obtener recursos de infraestructura en poco tiempo. Existen procesos DevOps que aprovechan los servicios ofrecidos por los proveedores de la nube, tales como computación, redes, almacenamiento y elasticidad. En particular, el aprovisionamiento de infraestructura en la nube utiliza el enfoque IaC para

definir la creación, actualización y ejecución de recursos de infraestructura en diferentes proveedores de la nube. La computación en la nube y el enfoque IaC están causando algunos cambios en la industria (Brikman, 2017), tales como:

- En lugar de administrar centros de datos, muchas empresas se están moviendo su infraestructura a la nube.
- En lugar de invertir en hardware, los equipos de operaciones pasan todo su tiempo trabajando en software.
- En lugar de configurar servidores y conectar cables de red, muchos administradores de sistemas están escribiendo código.

En la actualidad, las organizaciones que utilizan una cantidad considerable de tecnología están lidiando con la complejidad de cómo lograr que el desarrollo de software y las actividades de operaciones sean compactas, transparentes y totalmente integradas (Mann *et al.*, 2018). Además, los investigadores están estudiando y realizando estudios empíricos sobre las implicaciones de DevOps en la industria. Un estudio de caso exploró la implantación de DevOps en una compañía de desarrollo de software en Nueva Zelanda (Senapathi *et al.*, 2018) y explicó los facilitadores tecnológicos de DevOps, tales como la automatización del despliegue de software, la automatización del aprovisionamiento de la infraestructura y la gestión de la configuración del código. Otro estudio de caso realizado con tres empresas de desarrollo de software en Finlandia (Riungu-kalliosaari *et al.*, 2016) describe que los principales desafíos en la adopción de DevOps son la comunicación insuficiente y los entornos heterogéneos.

Si bien los investigadores y profesionales han hecho una enorme contribución a DevOps, todavía existen desafíos y problemas por resolver relacionados con el despliegue de aplicaciones de software. En particular, se debe abordar el desafío de mejorar el aprovisionamiento de infraestructura en la nube para apoyar el despliegue continuo de aplicaciones de software y reducir las barreras entre los desarrolladores y personal de operaciones.

1.2 Planteamiento del problema

En los últimos años comenzó un importante cambio tecnológico. Estamos dejando una era en la cual para implementar una aplicación de software era necesario instalar físicamente el hardware y un sistema operativo. Hoy en día, estamos afianzándonos en la era de la computación en la nube, en donde con unos pocos clics y en poco tiempo, podemos obtener una máquina virtual con todo el software necesario para su adecuado funcionamiento. La computación en la nube se compone de servicios basados en hardware, donde la

administración del hardware es altamente abstracta y la capacidad de la infraestructura es altamente elástica (Buyya *et al.*, 2011).

Los servicios en la nube se dividen de acuerdo con el nivel de abstracción de su modelo de servicio, es decir, Infraestructura como Servicio (*Infrastructure as a Service*, IaaS), Plataforma como Servicios (*Platform as a Service*, PaaS) y Software como Servicio (*Software as a Service*, SaaS). En particular, IaaS es un modelo de servicio con recursos virtualizados (computación, almacenamiento y redes) que permite el aprovisionamiento de recursos de infraestructura bajo pedido, junto con varias opciones de configuración de los sistemas operativos y las características de hardware personalizables (CPU, RAM y almacenamiento). Si bien la computación en la nube surgió como un servicio público (Buyya *et al.*, 2011), se adaptó a diferentes modelos de implementación, tales como nube pública, nube privada y nube híbrida. La nube pública está disponible para el público en general, con la forma de pago por el uso, mientras que la nube privada es propiedad de una sola organización. En cambio, la nube híbrida es un entorno compuesto por dos o más modelos de implementación diferentes, por ejemplo, los datos se almacenan en una nube privada y se utiliza la potencia de cómputo de una nube pública.

La Infraestructura como Código (*Infrastructure as Code*, IaC) proporciona un enfoque para la automatización de la infraestructura, independientemente de si el tipo de implementación es pública, privada o híbrida. El enfoque IaC ha sido implementado con éxito en las empresas más exigentes en asuntos tecnológicos, tales como Amazon, Netflix, Google, Facebook y Etsy, en donde los sistemas de TI (*Tecnología de la Información*) no solo son críticos para el negocio; los sistemas de TI son el negocio (Morris, 2016). En la práctica, el enfoque IaC utiliza scripts para definir y ejecutar los recursos de infraestructura en el modelo de servicio IaaS. Por ejemplo, la herramienta Ansible (DeHaan, 2012) utiliza scripts para aprovisionar la infraestructura, mientras que la herramienta Puppet (Puppet Labs, 2005) utiliza scripts para desplegar aplicaciones de software. En este escenario, las herramientas que proveen soporte al enfoque IaC tienen su propio lenguaje de scripting y además diferentes propósitos, tales como configuración y administración de sistemas operativos, aprovisionamiento de infraestructura, creación y configuración de entornos virtualizados, entre otros.

En un estudio empírico del sistema de control de versiones de 265 proyectos OpenStack, se descubrió que los scripts de infraestructura son de gran tamaño y cambian con frecuencia, lo que podría indicar la posibilidad de introducir errores (Jiang *et al.*, 2015). OpenStack es un ecosistema de proyectos que implementan una plataforma en la nube, y que requiere una gran cantidad de scripts para el aprovisionamiento y las pruebas en máquinas virtuales. En general, el tamaño de

los scripts, en número de líneas de código (*Lines of Code*, LOC), tienen un valor mediano de 2 486 LOC para proyectos con varios scripts, mientras que tiene un valor mediado de 1 398 LOC para proyectos con un solo script.

A pesar de que el enfoque IaC aporta múltiples beneficios, el uso de scripts ha generado problemas como cualquier código fuente de un lenguaje de programación. Al aumentar el tamaño de un script, aumenta también su complejidad y, por lo tanto, se requiere un mayor esfuerzo para su mantenimiento y depuración (Johann, 2017). En particular, el reto que aborta el presente trabajo de investigación está relacionado con el aprovisionamiento de la infraestructura en la nube:

El problema que afronta el aprovisionamiento de infraestructura en la nube es la diversidad de los lenguajes de scripting junto con la heterogeneidad de los servicios IaaS —cada proveedor de servicios en la nube ofrece un tipo diferente de infraestructura. Como resultado, el uso de scripts para definir el aprovisionamiento de infraestructura implica escribir varias líneas de código que dependen en gran medida de la plataforma de nube seleccionada y, por lo tanto, es necesario definir un script personalizado para cada proveedor. En consecuencia, utilizar scripts para definir y aprovisionar la infraestructura de la nube es una tarea que consume mucho tiempo y es propensa a errores.

1.3 Solución propuesta

Para abordar las limitaciones de la heterogeneidad de los recursos de infraestructura de cada proveedor de servicios en la nube y la diversidad de lenguajes de scripting para definir los recursos de infraestructura, se propone MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*) como una aproximación que apoya la Infraestructura como Código (*Infrastructure as Code*, IaC) utilizando la Ingeniería de Software Dirigida por Modelo (*Model-Driven Engineering*, MDE) para *abstraer* la complejidad de utilizar los recursos de infraestructura de diferentes proveedores, así como también para *abstraer* la complejidad de gestionar lenguajes de scripting de diferentes herramientas de aprovisionamiento en la nube. Además, MoCIP propone un proceso, basado en transformaciones de modelos, para *automatizar* la creación de modelos de infraestructura para diferentes proveedores de la nube y la generación de scripts que soporten el aprovisionamiento de infraestructura en diferentes proveedores de la nube.

El enfoque MoCIP está soportado por una herramienta de modelado de infraestructura para el aprovisionamiento en la nube (*An infrastruCTure modelinG*

tool for clOud provisioniNg, ARGON). Por un lado, ARGON provee un lenguaje específico de dominio denominado ArgonML (*ARGON Modeling Language*) para modelar los elementos de la infraestructura. Por otro lado, ARGON provee un motor de transformaciones de modelo a modelo (*Model-to-Model*, M2M) para crear modelos de infraestructura para diferentes proveedores de servicios en la nube y un motor de transformaciones de modelo a texto (*Model-to-Text*, M2T) para generar scripts que realicen el aprovisionamiento de infraestructura en la nube.

ArgonML es un lenguaje específico de dominio que *abstrae* las capacidades de la computación en la nube, tales como computación, almacenamiento y elasticidad. Este lenguaje tiene una sintaxis abstracta que utiliza las capacidades de la nube para definir un metamodelo de infraestructura y una sintaxis concreta que define la notación gráfica de los elementos de infraestructura y sus relaciones. A continuación, se explica brevemente los principales elementos de infraestructura de acuerdo con las capacidades de la nube. La *capacidad de computación* permite modelar máquinas virtuales con sus grupos de seguridad. Un grupo de seguridad trabaja como un cortafuegos, que permite las conexiones desde y hacia las máquinas virtuales a través de reglas. Además, permite modelar un balanceador de carga para distribuir las cargas de trabajo entre varias máquinas virtuales. La *capacidad de almacenamiento* permite modelar elementos, tales como bases de datos y servidores de archivos. La *capacidad de elasticidad* permite modelar una configuración de escalado horizontal, en donde se crea una red de máquinas virtuales con la finalidad de repartir la carga de trabajo entre todas las máquinas que conforman la red. Cuando el rendimiento de la red se ve afectado por el incremento de usuarios o la carga de trabajo, se agregan automáticamente nuevas máquinas virtuales a la red. En este escenario, a medida que se requiera, se creará nuevas máquinas virtuales para agregarlas a la red.

Los motores de transformación soportan la *automatización* del proceso de aprovisionamiento de la infraestructura en la nube. El motor de transformaciones M2M toma un modelo de origen, aplica un conjunto de reglas de transformación M2M y obtiene un modelo de destino. Por ejemplo, a partir de un modelo que representa la infraestructura en Amazon Web Services se generara un modelo que representa la infraestructura equivalente en Microsoft Azure. En este escenario, a partir de un modelo de infraestructura origen se pueden generar modelos para diferentes proveedores de servicios en la nube. Por otro lado, el motor de transformaciones M2T realiza la tarea de transformar un modelo de infraestructura en scripts con las instrucciones necesarias para aprovisionar la infraestructura en un proveedor de nube en particular.

Finalmente, es importante resaltar que en un entorno MDE los conceptos centrales son: *modelos* y *transformaciones* (Brambilla et al., 2017). Por esta razón, se propone un soporte al enfoque IaC utilizando *modelos* para definir los recursos de la infraestructura, así como *transformaciones* para generar modelos de infraestructura para diferentes proveedores de servicios IaaS y generar scripts que realicen la tarea de aprovisionamiento de infraestructura en la nube.

1.4 Contexto de la investigación

Esta tesis doctoral se ha desarrollado en el contexto del grupo de investigación de Ingeniería del Software y Sistemas de Información (ISSI) del Departamento de Sistemas Informáticos y Computación de la Universitat Politècnica de València (UPV).

Los trabajos que han hecho posible el desarrollo de esta tesis se engloban en proyectos de I+D financiados con fondos públicos. Los proyectos y ayudas son los siguientes:

- Proyecto Value@Cloud: “Desarrollo incremental de servicios en la nube dirigido por modelos y orientado al valor del cliente” de la convocatoria de ayudas a proyectos de I+D+i, orientada a los retos de la sociedad del año 2013 financiado por el Ministerio de Economía y Competitividad (España), Participantes: Universitat Politècnica de València (España) y Universidade Nova de Lisboa (Portugal). IP: Silvia Abrahão. De enero de 2013 a diciembre de 2017.
- Proyecto Adapt@Cloud: “Adaptación dinámica de servicios en la nube centrado en el usuario” de la convocatoria de ayudas a proyectos de I+D+i, orientada a los retos de la sociedad del año 2018 financiado por el Ministerio de Economía y Competitividad (España), Participantes: Universitat Politècnica de València (España), Universidade Nova de Lisboa (Portugal), Université Catholique de Louvain (Bélgica), Software Engineering Institute, Carnegie-Mellon University (USA). IP: Silvia Abrahão y Emilio Insfrán. De enero de 2018 a diciembre de 2020.
- Proyecto: “Model-Driven Incremental Development of Cloud Services”, Microsoft Azure Research Award, Microsoft Research. Valor estimado de la ayuda: USD \$ 40 000 por año. IP: Emilio Insfrán. De julio de 2014 a julio de 2016.
- El trabajo de Julio Sandobalín fue financiado por la Escuela Politécnica Nacional, Ecuador.

1.5 Contribuciones

Las contribuciones de esta tesis son el resultado del trabajo de investigación realizado para resolver el problema planteado.

1.5.1 Soporte a la infraestructura como código utilizando MDE

Se propone un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube, denominado MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*). MoCIP provee soporte a la Infraestructura como Código (*Infrastructure as Code*, IaC) en el proceso de aprovisionamiento de infraestructura en la nube.

Con fin de abstraer la complejidad de usar diferentes herramientas de aprovisionamiento y definir los recursos de infraestructura de distintos proveedores de servicios IaaS (*Infrastructure as a Service*) se utiliza la Ingeniería de Software Dirigida por Modelo (*Model-Driven Engineering*, MDE). En este caso, MoCIP tiene la herramienta ARGON (*An infrastruCTure modelinG tool for clOud provisioniNG*) para soportar el proceso de aprovisionamiento en la nube (disponible en <http://bit.ly/2QUSEA2>). Por un lado, ARGON proporciona el lenguaje específico ArgonML (*ARGON Modeling Language*) para modelar los recursos de infraestructura de la nube. Por otro lado, ARGON provee transformaciones de modelos para generar diversos modelos de infraestructura y scripts para el aprovisionamiento de infraestructura en la nube.

1.5.2 Metamodelos de infraestructura y sus mapeos

Se propone la abstracción de las capacidades de la computación en la nube, tales como cómputo, almacenamiento, redes y elasticidad para la definición de metamodelos de infraestructura. En primer lugar, se define un metamodelo de infraestructura *abstracto* que es independiente del proveedor de la nube. En segundo lugar, se definen los metamodelos que representan la infraestructura de Amazon Web Services y Microsoft Azure.

Para automatizar las transformaciones entre modelos de infraestructura se debe realizar los mapeos que definen las correspondencias entre elementos de cada modelo. Por ejemplo, se realiza el mapeo entre los conceptos de los elementos del metamodelo de Amazon Web Services y del metamodelo de Microsoft Azure. Los metamodelos de infraestructura y sus mapeos se utilizan en la herramienta ARGON. Es importante mencionar que ARGON no está limitada a los proveedores de servicios IaaS mencionados anteriormente. Para agregar soporte a nuevos proveedores de la nube, se debe definir el metamodelo que represente la infraestructura del nuevo proveedor de servicios IaaS y los correspondientes mapeos.

1.5.3 Un pipeline para el aprovisionamiento de infraestructura

Se propone un encadenamiento de herramientas DevOps para automatizar las tareas de control de versiones, generación de scripts, pruebas y el aprovisionamiento de infraestructura en la nube. En particular, se presenta un *pipeline* para apoyar al enfoque MoCIP mediante las prácticas de integración y despliegue continuos en el aprovisionamiento de infraestructura en la nube.

El pipeline utiliza la herramienta ARGON junto con su lenguaje ArgonML y un conjunto de herramientas de la comunidad DevOps, tales como GitHub (Preston-Werner *et al.*, 2008), Maven (Sonatype, 2002), Nexus (Sonatype, 2008), Jenkins (Kawaguchi, 2011), Ansible (DeHaan, 2012) y Ansible_spec (Volanja, 2013). En este caso, el encadenamiento de herramientas provee la abstracción necesaria para mitigar la complejidad del uso de lenguajes de scripting en la configuración de las prácticas de integración y despliegue continuos.

1.5.4 Evidencia empírica sobre MoCIP y ARGON

Se propone un conjunto de experimentos controlados para obtener evidencia empírica sobre el rendimiento y la percepción de los usuarios al utilizar MoCIP y la herramienta ARGON. Por un lado, se realizó dos experimentos controlados para evaluar el enfoque MoCIP respecto a su facilidad de uso percibida, utilidad percibida e intensidad de uso (disponible en <http://bit.ly/34wNBd8>). Por otro lado, se realizó una familia de tres experimentos controlados para evaluar la herramienta ARGON versus las herramientas Ansible respecto a la definición de los recursos de infraestructura de la nube (disponible en <http://bit.ly/2r2iPdw>). En este caso, se evaluó la efectividad y eficiencia de los participantes al definir la infraestructura, así como la facilidad de uso percibida, la utilidad percibida y la intensidad de uso luego de realizar las tareas experimentales con las dos herramientas.

1.6 Publicaciones

El trabajo de investigación de esta tesis ha dado lugar a dos publicaciones en revistas científicas, seis publicaciones en conferencias internacionales, tres participaciones en conferencias nacionales en España y una invitación a un blog divulgativo de carácter científico.

1.6.1 En revistas científicas

1. J. Sandobalín, E. Insfrán, y S. Abrahão, (**En Prensa**), “On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven versus Code-Centric,” *IEEE Access*, 2019. Factor de Impacto 2019: 4.098 (JCR Q1, Information Systems).

2. J. Sandobalín, E. Insfrán, y S. Abrahão, “A Smart Provisioning Approach to Cloud Infrastructure,” *Journal of Computers*, vol. 29, 2018.

1.6.2 *En conferencias internacionales*

1. J. Sandobalín, E. Insfrán, y S. Abrahão, “ARGON: A Model-Driven Infrastructure Provisioning Tool,” in *ACM/IEEE 22th International Conference on Model Driven Engineering Languages and Systems, MODELS, Tool Demonstration*, 2019. CORE A (2018).
2. J. Sandobalín, E. Insfrán, y S. Abrahão, “An Infrastructure Modeling Approach for Multi-Cloud Provisioning,” in *27th International Conference on Information Systems Development, ISD*, 2018. CORE A (2018).
3. J. Sandobalín, E. Insfrán, y S. Abrahão, “End-to-End Automation in Cloud Infrastructure Provisioning,” in *26th International Conference on Information Systems Development, ISD*, 2017. CORE A (2017).
4. J. Sandobalín, “A Model-Driven Approach to Continuous Delivery of Cloud Resources,” in *15th International Conference on Service-Oriented Computing, ICSOC, PhD Symposium*, 2017, CORE A (2017). **Premio al Mejor Artículo.**
5. J. Sandobalín, E. Insfrán, y S. Abrahão, “ARGON: A Tool for Modeling Cloud Resources,” in *15th International Conference on Service-Oriented Computing, ICSOC, Tool Demonstration*, 2017, CORE A (2017).
6. J. Sandobalín, E. Insfrán, y S. Abrahão, “An Infrastructure Modelling Tool for Cloud Provisioning,” in *IEEE 14th International Conference on Services Computing, SCC*, 2017. CORE A (2017).

1.6.3 *En conferencias nacionales de España*

1. J. Sandobalín, E. Insfrán, y S. Abrahão, “A Model-driven Migration Approach among Cloud Providers,” en *XIV Jornadas de Ciencia e Ingeniería de Servicios, JCIS*, 2018.
2. J. Sandobalín, E. Insfrán, y S. Abrahão, “Automatización del Aprovechamiento de Infraestructura en la Nube,” en *XIII Jornadas de Ciencia e Ingeniería de Servicios, JCIS*, 2017. **Premio al Mejor Artículo.**
3. J. Sandobalín, M. Zúñiga-Prieto, E. Insfrán, S. Abrahão y C. Cano, “Una aproximación DevOps para el Desarrollo Dirigido por Modelos de Servicios Cloud,” en *XII Jornadas de Ciencia e Ingeniería de Servicios, JCIS*, 2016.

1.6.4 *En un Blog científico divulgativo*

1. Julio Sandobalín, A Model-driven Migration Approach among Cloud Providers: Modeling Languages (Blog mantenido por el Prof. Dr. Jordi

Cabot, UOC, España), 2018, [Online] Disponible:
<http://bit.ly/36APLd0>

1.7 Estructura de la tesis

A continuación, se presenta el contenido de cada capítulo de la tesis:

- **Capítulo 2:** Metodología de Investigación

En este capítulo se presenta la metodología que guía el trabajo de investigación de esta tesis. La metodología investiga la interacción de un artefacto en un contexto de aplicación. En consecuencia, se explica el diseño de un artefacto y cómo se valida el artefacto en un problema del contexto de aplicación.

- **Capítulo 3:** Marco Conceptual

En este capítulo se presenta la definición de los conceptos o *constructos* utilizados para el planteamiento del problema, la solución propuesta, el soporte de herramientas y la validación. Este capítulo contribuye a la comprensión de los temas de esta tesis.

- **Capítulo 4:** Estado del Arte

En este capítulo se presenta una revisión de la literatura para resaltar los trabajos relacionados con el aprovisionamiento de infraestructura en la nube que utilizan la Ingeniería de Software Dirigida por Modelo (*Model-Driven Engineering*, MDE) y la Infraestructura como Código (*Infrastructure as Code*, IaC). El análisis de los trabajos relacionados permite comprender los problemas que afronta el trabajo de investigación en esta tesis.

- **Capítulo 5:** El Enfoque MoCIP

En este capítulo se presenta el diseño del enfoque MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*). Se describen los requisitos del diseño, así como los metamodelos que describen la estructura de los diferentes proveedores de servicios IaaS (*Infrastructure as a Service*) que apoyan a MoCIP —utilizando MDE— en el aprovisionamiento de infraestructura en la nube. Además, se explica cómo MoCIP soporta el enfoque IaC.

- **Capítulo 6:** Soporte de Herramientas

En este capítulo se presenta la herramienta ARGON (*An infrastruCTuRe modelinG tool for clOud provisioniNg*) como un soporte de MoCIP en el aprovisionamiento de infraestructura en la nube. Por un lado, ARGON

provee el lenguaje ArgonML (*ARGON Modeling Language*) para modelar los recursos de infraestructura de la nube. Por otro lado, ARGON realiza transformaciones de modelos para generar modelos de infraestructura para diferentes proveedores de servicios IaaS y scripts para el aprovisionamiento de infraestructura en la nube. Además, un pipeline configura a la herramienta ARGON en un encadenamiento de herramientas DevOps para proveer una entrega continua de recursos de infraestructura en la nube.

- **Capítulo 7:** Validación de MoCIP

En este capítulo se presenta la validación del enfoque MoCIP a través de la ejecución de dos experimentos controlados. Se delimita el alcance de cada experimento mediante un objetivo. Se describe la planificación de los experimentos y su ejecución. Finalmente, se realiza el análisis de los resultados de los experimentos utilizando estadística descriptiva y la prueba de hipótesis aplicando estadística inferencial.

- **Capítulo 8:** Validación de ARGON

En este capítulo se presenta la validación de la herramienta ARGON y, en particular, de su lenguaje de modelado ArgonML a través de una familia de experimentos controlados. Se delimita el alcance de la familia de experimentos a través de un objetivo. Se describe la planificación de los experimentos y su ejecución. Se realiza el análisis de los datos utilizando estadística descriptiva y la prueba de hipótesis mediante estadística inferencial. Finalmente, se realiza un metaanálisis para resumir y sintetizar el resultado de todos los experimentos.

- **Capítulo 9:** Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones basadas en las contribuciones realizadas a lo largo de este trabajo de investigación. Se contestan las preguntas de investigación que guiaron esta tesis. Finalmente, se presentan los proyectos previstos a realizar como próximos pasos derivados de esta tesis.

Capítulo 2

Metodología de Investigación

En este capítulo se presenta la metodología de investigación como un soporte conceptual al procedimiento que se utilizó para realizar esta tesis. El objetivo de la metodología *Design Science* es el estudio de un artefacto en contexto y sus dos actividades principales son el diseño y la investigación de ese artefacto en contexto (Wieringa, 2014). En primer lugar, se presenta el diseño del artefacto en un contexto de aplicación. A continuación, se exponen los objetivos de la investigación y las preguntas de investigación. Finalmente, se plantea el ciclo de diseño del artefacto y dos ciclos empíricos para contestar a las preguntas de investigación.

En la Sección 2.1 se presenta el proyecto *Design Science* donde se especifica el artefacto, su contexto y el problema del contexto.

En la Sección 2.2 se presenta la estructura de los objetivos de investigación.

En la Sección 2.3 se presentan las preguntas de investigación que se derivan de los objetivos de investigación.

En la Sección 2.4 se presenta el ciclo de diseño del artefacto y los dos ciclos empíricos para validar el artefacto y su instrumento.

En la Sección 2.5 se presenta un resumen de la metodología de investigación.

2.1 Proyecto *Design Science*

Para el desarrollo del proyecto de doctorado se llevó a cabo un proyecto *Design Science*. En un proyecto *Design Science* el objeto de estudio es un artefacto en contexto, y sus dos actividades principales son el diseño y la investigación del artefacto en contexto (Wieringa, 2014). El artefacto debe estar diseñado para interactuar en un problema del contexto, a fin de mejorar algo en dicho contexto. Es importante mencionar que el artefacto por sí solo no resuelve ningún problema. Es la interacción entre el artefacto y el contexto lo que contribuye a resolver los problemas. El proyecto *Design Science* de esta tesis doctoral ha sido presentado en (Sandobalín, 2017).

Para el proyecto de doctorado se ha diseñado MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*) para proveer soporte al proceso de aprovisionamiento de infraestructura en la nube. El enfoque MoCIP es un *artefacto*, y el *contexto* consiste en los desarrolladores y el personal de operaciones que necesitan definir, actualizar y ejecutar la infraestructura en diferentes proveedores de servicios IaaS (*Infrastructure as a Service*). El *problema del contexto* es la diversidad de lenguajes de scripting para definir, actualizar y ejecutar la infraestructura en la nube y que los proveedores de servicios IaaS ofrecen diferentes tipos de infraestructura.

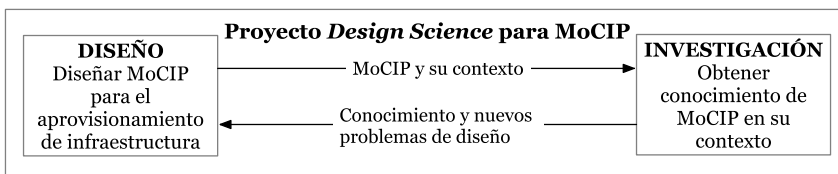


Figura 2-1 Proyecto *Design Science* para el enfoque MoCIP

La Figura 2-1 presenta las dos actividades principales de un proyecto *Design Science*: diseño e investigación. Por un lado, la actividad de diseño permite definir problemas de diseño para (re)diseñar un artefacto que contribuye de alguna manera a alcanzar un objetivo y que está relacionado con el diseño del enfoque MoCIP. Por otro lado, en la actividad de investigación se plantean preguntas de investigación relacionadas con la obtención de conocimiento acerca de la interacción entre MoCIP y su contexto de aplicación.

La Figura 2-2 presenta la estructura del proyecto *Design Science* para el enfoque MoCIP. El contexto social contiene a los stakeholders que son afectados o pueden ser afectados por el proyecto:

- Los desarrolladores y el personal de operaciones son los principales actores en DevOps. Debido a que el proyecto de doctorado está enfocado en el contexto de DevOps, los desarrolladores y el personal de operaciones serían los principales afectados y/o beneficiarios.
- Los proyectos del Plan Nacional de Investigación Value@Cloud (TIN2013-46300-R) y Adapt@Cloud (TIN2017-84550-R) proveen el marco de trabajo al proyecto de doctorado y brindan soporte en el diseño e investigación del enfoque MoCIP.
- La Universitat Politècnica de València (UPV) y la Escuela Politécnica Nacional (EPN) apoyan la investigación y la divulgación de los resultados de la investigación. En particular, la EPN patrocina al investigador principal (Julio Sandobalín) del proyecto de doctorado.

Por otro lado, el contexto de conocimiento consiste en teorías existentes de ciencia e ingeniería para ayudar a las actividades de diseño e investigación:

- DevOps y la Infraestructura como Código proveen los conceptos, principios y prácticas para apoyar las tareas de aprovisionamiento de infraestructura en la nube.
- La computación en la nube proporciona al contexto de aplicación en donde se aprovisionará la infraestructura.
- La Ingeniería de Software Dirigida por Modelos provee sus conceptos centrales: modelos y transformaciones (Brambilla *et al.*, 2017). Por un lado, los *modelos* permiten abstraer las características comunes de la infraestructura de los proveedores de la nube con el objetivo de proveer un lenguaje de modelado específico del dominio. Por otro lado, las *transformaciones* son operaciones de manipulación de los modelos. En este caso, las transformaciones de modelo a modelo permiten crear modelos de infraestructura para cada proveedor de la nube y las transformaciones de modelo a texto permiten generar scripts para realizar el aprovisionamiento de la infraestructura en la nube.
- La Ingeniería de Software Empírica provee los lineamientos para llevar a cabo experimentos controlados del enfoque MoCIP en su contexto de aplicación. Además, con los datos recolectados de cada experimento se realizan análisis descriptivos e inferenciales para proveer evidencia empírica con los resultados de la actividad de investigación.

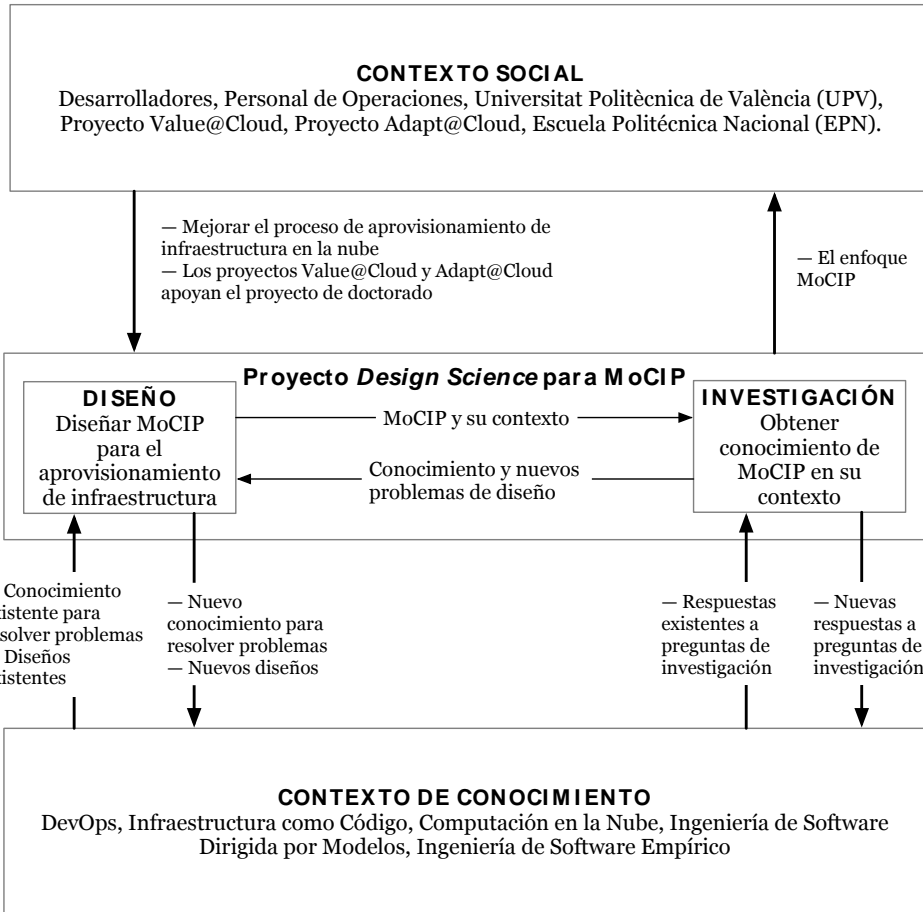


Figura 2-2 Marco para el proyecto *Design Science* para el enfoque MoCIP

2.2 Objetivos de investigación

En un proyecto *Design Science* se debe diferenciar entre los objetivos del investigador y los objetivos de los stakeholders. Los problemas de diseño requieren un análisis de los objetivos reales o hipotéticos de los stakeholders (Wieringa, 2014). En este caso, el proyecto de doctorado se basa en un objetivo hipotético de los stakeholders:

Mejorar el proceso de aprovisionamiento de infraestructura en la nube.

A pesar de que el objetivo propuesto para los stakeholders es muy general, también es un punto de partida para definir los objetivos de la investigación y proponer una solución para satisfacer dicho objetivo. La Figura 2-3 muestra la estructura de los objetivos (O) de investigación del proyecto de doctorado.

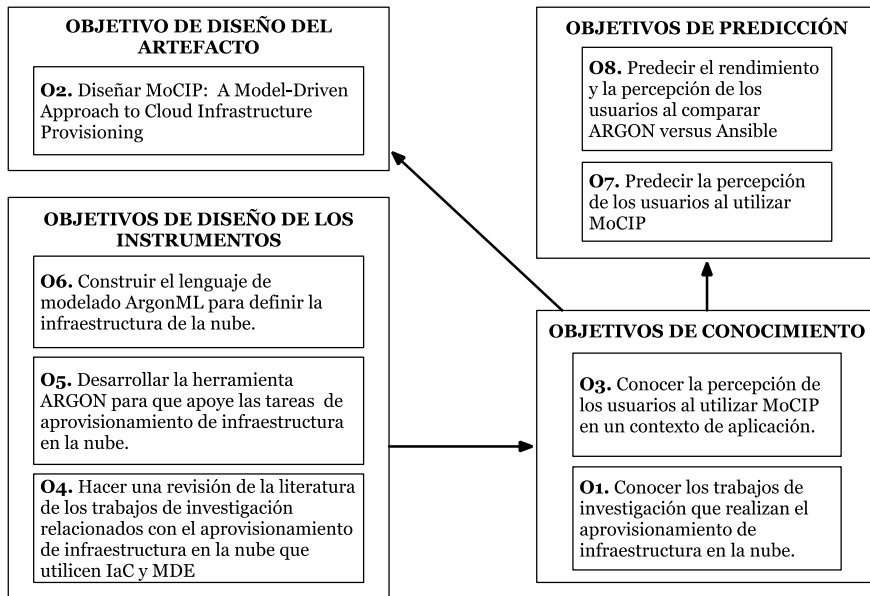


Figura 2-3 Estructura de los objetivos de investigación del proyecto de doctorado

2.2.1 *Objetivo de diseño del artefacto*

El objetivo de diseñar un artefacto es solucionar, mitigar o mejorar algún problema en el contexto social (Wieringa, 2014). El objetivo (O) de diseño del artefacto es:

- **O2.** Diseñar MoCIP —*A Model-Driven Approach to Cloud Infrastructure Provisioning.*

El O2 propone diseñar MoCIP, un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube. En este caso, el artefacto MoCIP plantea mejorar el proceso de aprovisionamiento de infraestructura en la nube utilizando IaC y MDE.

2.2.2 *Objetivos de conocimiento*

Los objetivos de conocimiento describen los fenómenos a estudiar (Wieringa, 2014). Los objetivos (O) de conocimiento son los siguientes:

- **O1.** Conocer los trabajos de investigación que realizan el aprovisionamiento de infraestructura en la nube.
- **O3.** Conocer la percepción de los usuarios al utilizar MoCIP en un contexto de aplicación.

El O1 busca conocer los trabajos de investigación y/o soluciones tecnológicas que realizan el aprovisionamiento de infraestructura en la nube. En cambio, el O3 busca conocer los resultados de la interacción entre el enfoque MoCIP y su contexto de aplicación.

2.2.3 *Objetivos de diseño de los instrumentos*

A partir de los objetivos de conocimiento se elaboran preguntas de investigación, y para contestar a estas preguntas se pueden desarrollar instrumentos. Los objetivos (O) de diseño de los instrumentos son los siguientes:

- **O4.** Hacer una revisión de la literatura de los trabajos de investigación relacionados con el aprovisionamiento de infraestructura en nube que utilicen la Infraestructura como Código (*Infrastructure as Code*, IaC) y la Ingeniería de Software Dirigida por Modelo (*Model-Driven Engineering*, MDE).
- **O5.** Desarrollar la herramienta ARGON (*An infrastruRcture modelinG tool for clOud provisioniNg*) para que apoye las tareas de aprovisionamiento de infraestructura en la nube.
- **O6.** Construir el lenguaje de modelado ArgonML (*ARGON Modeling Language*) para definir la infraestructura de la nube.

El O4 propone realizar una revisión de la literatura de los trabajos de investigación relacionados con el aprovisionamiento de infraestructura en la nube que utilicen IaC y MDE. El O4 es un instrumento que ayuda a contestar la pregunta de investigación relacionada con el O1. Por otro lado, el O5 propone desarrollar la herramienta ARGON para apoyar en las tareas de aprovisionamiento de infraestructura en la nube, mientras que el O6 propone construir el lenguaje de modelado ArgonML para definir los recursos de la infraestructura de la nube. En este caso, O5 y O6 son instrumentos que ayudan a contestar las preguntas de investigación relacionadas con el O2.

2.2.4 *Objetivos de predicción*

Una predicción es una creencia sobre lo que sucederá en el futuro (Wieringa, 2014). Un objetivo de predicción intenta predecir cómo será la interacción de un artefacto con el contexto de un problema o cómo evolucionaría un problema si no fuera tratado. Los objetivos (O) de predicción son los siguientes:

- **O7.** Predecir la percepción de los usuarios al utilizar MoCIP.
- **O8.** Predecir el rendimiento y la percepción de los usuarios al comparar ARGON versus Ansible.

El O7 propone predecir la percepción de los usuarios al utilizar MoCIP en un contexto de aplicación. En cambio, el O8 plantea predecir el rendimiento y la percepción de los usuarios al comprar ARGON versus Ansible respecto a la definición de la infraestructura de la nube.

2.3 Preguntas de investigación

Los objetivos de investigación del proyecto de doctorado plantean un conjunto de desafíos que deben ser superados. En un proyecto *Design Science*, estos desafíos se derivan en un conjunto de preguntas de investigación. En este caso, los objetivos de conocimiento se refinan en preguntas de conocimiento. Por otro lado, un problema de diseño es un problema para (re)diseñar un artefacto para que contribuya mejor al logro de algún objetivo.

A continuación, se presentan las preguntas de investigación (*Research Question*, RQ) mediante el uso de preguntas de conocimiento (*Knowledge Question*, KQ) y de los problemas de diseño o problemas de investigación técnicos (*Technical Research Problem*, TRP).

- **RQ1.** (KQ) ¿Cuáles son los trabajos de investigación relacionados con el aprovisionamiento de infraestructura en la nube que utilizan IaC y MDE?
- **RQ2.** (TRP) ¿Cómo diseñar el artefacto MoCIP utilizando IaC y MDE para realizar el aprovisionamiento de infraestructura en la nube?
- **RQ3.** (KQ) ¿Cuál es la percepción de los usuarios al utilizar MoCIP en su contexto de aplicación?

La pregunta RQ1 es una pregunta de conocimiento que está motivada por el objetivo O1. En este caso, para contestar a esta pregunta se debe realizar una revisión de la literatura científica (O4) y establecer el estado actual de los trabajos de investigación relacionados con el aprovisionamiento de infraestructura que utilizan IaC y MDE. Además, la respuesta a la pregunta RQ1 ayuda a los investigadores a identificar los problemas que afronta el aprovisionamiento de infraestructura en la nube y descubrir posibles áreas de investigación en ese contexto.

La pregunta RQ2 es un problema de diseño que está motivada por el objetivo O2. En este caso, la respuesta a la pregunta RQ2 define el ámbito de diseño del artefacto MoCIP. Sin embargo, la pregunta RQ2 debe ser refinada tomando en cuenta los objetivos O5 y O6 del diseño de los instrumentos. En particular, se presentan problemas de diseño para desarrollar instrumentos que soporten al artefacto MoCIP.

- RQ2.1. (TRP) ¿Cómo desarrollar la herramienta ARGON para que soporte a MoCIP en el aprovisionamiento de infraestructura en la nube?
- RQ2.2. (TRP) ¿Cómo construir el lenguaje ArgonML para modelar los elementos de infraestructura de la nube?

La pregunta RQ2.1 es un *problema de diseño* que está motivada por el objetivo O5. En este escenario, la respuesta a la pregunta RQ2.1 presenta un esfuerzo de ingeniería para desarrollar una herramienta MDE que realice la *abstracción* y *automatización* del enfoque IaC para soportar el aprovisionamiento de infraestructura en la nube. De manera similar, la pregunta RQ2.2 es un *problema de diseño* que está motivada por el objetivo O6. La respuesta a la pregunta RQ2.2 representa un esfuerzo de ingeniería para desarrollar un lenguaje específico de dominio para *abstraer* y *modelar* las capacidades de la computación en la nube de los proveedores de servicios IaaS.

La pregunta RQ3 es una pregunta de conocimiento que está motivada por el objetivo O3. En este caso, el objetivo O3 desea conocer la percepción de los usuarios al utilizar MoCIP en un contexto de aplicación. Sin embargo, debido a la herramienta ARGON soporta a MoCIP, el objetivo O3 es refinado en base a los objetivos de predicción O7 y O8.

En primer lugar, la pregunta RQ3 es refinada en base al objetivo O7 en las siguientes preguntas de conocimiento relacionadas con MoCIP:

- RQ3.1. (KQ) ¿MoCIP se percibe como fácil de usar?
- RQ3.2. (KQ) ¿MoCIP se percibe como útil?
- RQ3.3. (KQ) ¿Existe la intención de usar MoCIP en el futuro?

Por un lado, la respuesta a la pregunta RQ3.1 presenta una predicción de los usuarios de la *facilidad de uso percibida* de MoCIP. En cambio, la respuesta a la pregunta RQ3.2 presenta una predicción de los usuarios de la *utilidad percibida* de MoCIP. Por otro lado, la respuesta a la pregunta RQ3.3 presenta una predicción de los usuarios sobre la *intención de usar* MoCIP en el futuro.

En segundo lugar, la pregunta RQ3 es refinada tomando en cuenta el objetivo O8. En este caso, las preguntas de conocimiento están relacionadas con la comparación de dos herramientas IaC. Se desea comparar ARGON versus Ansible. Por un lado, ARGON es la herramienta IaC que soporta a MoCIP. Por otro lado, Ansible es una herramienta IaC (de control) ampliamente utilizada en la industria.

- RQ3.4. (KQ) ¿Cuál herramienta IaC es más efectiva al definir la infraestructura de la nube, ARGON o Ansible?

- RQ3.5. (KQ) ¿Cuál herramienta IaC es más eficiente al definir la infraestructura de la nube, ARGON o Ansible?
- RQ3.6. (KQ) ¿Cuál herramienta IaC se percibe como más fácil de usar, ARGON o Ansible?
- RQ3.7 (KQ) ¿Cuál herramienta IaC se percibe como más útil, ARGON o Ansible?
- RQ3.8 (KQ) ¿Cuál herramienta IaC tienen la mayor intención de uso en el futuro, ARGON o Ansible?

La respuesta a la pregunta RQ3.4 presenta qué herramienta IaC es más precisa (*efectiva*) en definir la infraestructura de la nube, mientras que la respuesta a la pregunta RQ3.5 muestra qué herramienta IaC es más exacta (*eficiente*) en la definición de la infraestructura de la nube en relación con el tiempo empleado. Por otro lado, la respuesta a la pregunta RQ3.6 expone qué herramienta IaC es percibida como más *fácil de usar* al definir la infraestructura de la nube, mientras que la respuesta a la pregunta RQ3.7 muestra que herramienta IaC es percibida como más *útil* al definir la infraestructura de la nube. Finalmente, la respuesta a la pregunta RQ3.8 expone qué herramienta IaC será *utilizada en el futuro* por parte de los usuarios.

2.4 Ciclos de ingeniería, diseño y empírico

En un proyecto Design Science un ciclo de ingeniería es un proceso racional de resolución de problemas (Wieringa, 2014). El resultado de un ciclo de ingeniería es un ciclo de diseño de un *tratamiento* validado —a través de un ciclo empírico— que se transfiere al mundo real, se utiliza y se evalúa. En este contexto, un tratamiento es la interacción entre el artefacto y el contexto del problema.

A continuación, se describen las tareas fundamentales del ciclo de diseño:

- **T1. Investigación del problema.** Se investiga los fenómenos que hay que mejorar en el problema del contexto.
- **T2. Diseño del tratamiento.** Se presenta el diseño del artefacto para tratar el problema del contexto.
- **T3. Validación del tratamiento.** Se realiza la validación empírica del tratamiento para evaluar si el diseño mejora el problema del contexto.

El ciclo de diseño es parte del ciclo de ingeniería, en el que se implementa un tratamiento diseñado y validado en el contexto del problema, y se evalúa la implementación. En este contexto, para transferir el tratamiento al mundo real, utilizarlo y validarlo debe utilizar las siguientes tareas:

- **Implantación del tratamiento.** Se trata el problema en el mundo real con el artefacto diseñado.
- **Evaluación de la implantación.** Se evalúa si el tratamiento ha sido exitoso en el mundo real. Esta tarea puede ser el comienzo de una nueva iteración a través del ciclo de ingeniería.

El objetivo de la implantación del tratamiento es implantar el artefacto diseñado en el contexto del problema en el mundo real, mientras que el objetivo de la evaluación de la implantación es evaluar un tratamiento después de implantarlo en el contexto del problema original. El proyecto de doctorado aborda las tareas T1, T2 y T3 del ciclo de diseño y plantea realizar como trabajo futuro la implantación y evaluación del tratamiento.

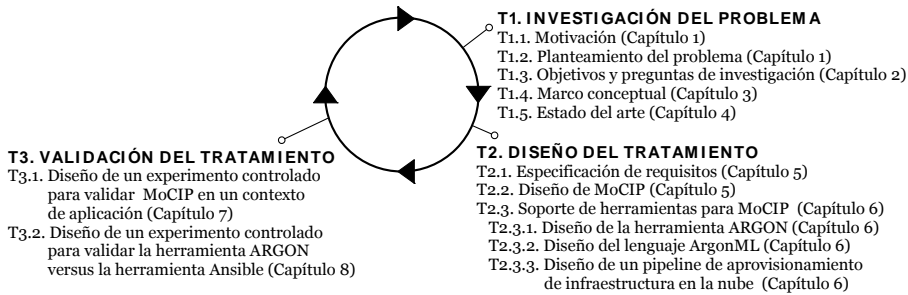


Figura 2-4 Ciclo de diseño del proyecto de doctorado

La Figura 2-4 presenta el ciclo de diseño del proyecto de doctorado y el mapeo entre las tareas del ciclo de diseño y los capítulos de esta tesis. A continuación, se describen las tareas (T) fundamentales del ciclo de diseño:

- **T1. Investigación del problema.** Se realiza una introducción al proyecto de doctorado (T1.1) así como el planteamiento del problema que hay que mejorar (T1.2). Se plantean los objetivos y las preguntas de investigación (T1.3). El marco conceptual (T1.4) define conceptos —llamados constructos— para definir las estructuras en el artefacto y su contexto. El estado del arte (T1.5) presenta el estado actual de los trabajos de investigación relacionados con el proyecto de doctorado.
- **T2. Diseño del tratamiento.** Se especifican los requisitos (T2.1) del artefacto y de la interacción entre el artefacto y su contexto. Se presenta el diseño de MoCIP (T2.2) para apoyar el aprovisionamiento de infraestructura en la nube. Además, se presenta el diseño de las herramientas que soportan MoCIP, tales como la herramienta ARGON (T2.3.1), el lenguaje de modelado ArgonML (T2.3.2) y el pipeline de aprovisionamiento de infraestructura (T2.3.3).

- **T3. Validación del tratamiento.** Se realiza la validación de MoCIP en el problema del contexto (T3.1), así como la validación de la herramienta ARGON versus la herramienta Ansible (T3.2).

Un ciclo empírico es una forma racional de contestar a las preguntas de conocimiento (Wieringa, 2014). En este caso, la tarea T3.1 de validación del tratamiento —del ciclo de diseño— produce un ciclo empírico para validar las preguntas de conocimiento (RQ3.1 - RQ3.3) sobre MoCIP en un contexto de aplicación. El método de investigación seleccionado para validar el tratamiento es un experimento controlado.

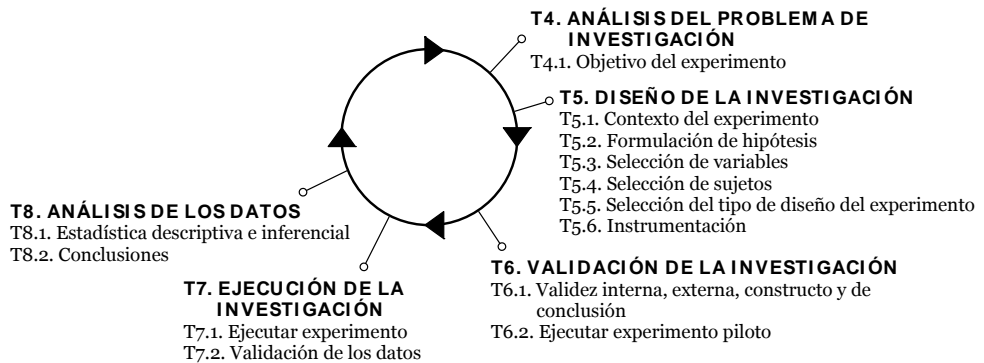


Figura 2-5. Ciclo empírico para validar MoCIP

La Figura 2-5 presenta el ciclo empírico para validar MoCIP en un contexto de aplicación. En este caso, el ciclo empírico consta de las siguientes tareas:

- **T4. Análisis del problema de investigación.** Se define el problema a investigar mediante el objetivo del experimento (T4.1): *Analizar* el aprovisionamiento de la infraestructura en Amazon Web Services utilizando MoCIP, *con el propósito de* evaluar la percepción de los usuarios, *con respecto a* la facilidad de uso percibida, utilidad percibida e intención de uso, *desde el punto de vista de* los ingenieros de software noveles, y *en el contexto de* los estudiantes del grado en Ingeniería Informática.
- **T5. Diseño de la investigación.** Los participantes del experimento deben realizar el aprovisionamiento de infraestructura en la nube (T5.1). Se define las hipótesis (T5.2) y las variables (T5.3) del experimento. Los sujetos (T5.4) son estudiantes del grado en Ingeniería Informática. El diseño del experimento (T5.5) tiene un factor (el aprovisionamiento de infraestructura en Amazon Web Services) con un tratamiento (MoCIP). La instrumentación (T5.6) son los objetos experimentales, las guías del experimento y los instrumentos de medición.

- **T6. Validación de la investigación.** Se realiza el análisis de las amenazas a la validez del experimento (T6.1). Además, se ejecuta un experimento piloto (T6.2) para validar y corregir la instrumentación.
- **T7. Ejecución de la investigación.** Se ejecuta el experimento controlado (T7.1) y luego se valida que los datos recolectados estén completos y en el formato adecuado (T7.2).
- **T8. Análisis de los datos.** Se realiza la estadística descriptiva para organizar, presentar y describir los datos, y la estadística inferencial realiza la prueba de hipótesis (T8.1). Con la evidencia empírica se exponen las conclusiones de los resultados en el contexto del experimento (T8.2).

Por otro lado, la tarea T3.2 de validación del tratamiento —del ciclo de diseño— produce un ciclo empírico para validar las preguntas de conocimiento RQ3.4 - RQ3.8 sobre la comparación de las herramientas IaC en un contexto de aplicación. En este caso, se realiza una familia de experimentos para evaluar la herramienta ARGON versus la herramienta Ansible respecto a la definición de la infraestructura de la nube. La Figura 2-6 presenta el ciclo empírico para validar la herramienta ARGON contra la herramienta Ansible.

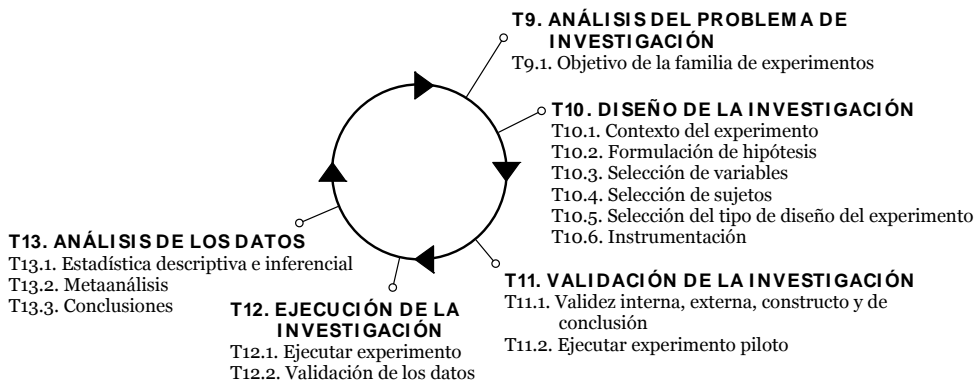


Figura 2-6 Ciclo empírico para validar la herramienta ARGON

En este caso, el ciclo empírico tiene las siguientes tareas:

- **T9. Análisis del problema de investigación.** Se define el problema a investigar mediante el objetivo del experimento (T9.1): *Analizar* la definición de la infraestructura de la nube especificada por ARGON y Ansible, *con el propósito* de evaluar las herramientas, *con respecto a* la efectividad, eficiencia, facilidad de uso percibida, utilidad percibida e intención de uso, *desde el punto de vista de* los ingenieros de software noveles, *y en el contexto de* los estudiantes del grado en Ingeniería

Informática y del Máster en Ingeniería y Tecnología de Sistemas Software.

- **T10. Diseño de la investigación.** Los participantes del experimento deben definir la infraestructura que será aprovisionada en Amazon Web Services (T10.1). Se define las hipótesis (T10.2) y las variables (T10.3) del experimento. Los sujetos (T10.4) son estudiantes del grado en Ingeniería Informática y del Máster en Ingeniería y Tecnología de Sistemas Software. El diseño del experimento (T10.5) es de un factor (definir la infraestructura de la nube) con dos tratamientos (ARGON y Ansible). La instrumentación (T10.6) son los objetos experimentales, las guías del experimento y los instrumentos de medición.
- **T11. Validación de la investigación.** Se realiza el análisis de las amenazas a la validez del experimento (T11.1). Además, se ejecuta un experimento piloto (T11.2) para validar y corregir la instrumentación de cada tratamiento (ARGON y Ansible).
- **T12. Ejecución de la investigación.** Se ejecuta cada experimento controlado (T12.1) y luego se valida que los datos recolectados estén completos y en el formato adecuado (T12.2).
- **T13. Análisis de los datos.** Se realiza la estadística descriptiva para organizar, presentar y describir los datos de cada experimento, y la estadística inferencial ejecuta la prueba de hipótesis (T13.1). Además, se realiza un metaanálisis (T13.2) para sintetizar los resultados de todos los experimentos. Con la evidencia empírica de la prueba de hipótesis y el metaanálisis se exponen las conclusiones de los resultados en el contexto del experimento (T13.3).

Para una mejor explicación de los ciclos aplicados a esta tesis, la Figura 2-7 presenta el ciclo de ingeniería con su ciclo de diseño y dos ciclos empíricos.

En este caso, el ciclo de diseño consiste en la investigación del problema (T1), diseño del tratamiento (T2) y la validación del tratamiento (T3). El ciclo de diseño es parte del ciclo de ingeniería, mientras que un ciclo empírico es una forma racional de contestar a las preguntas de conocimiento planteadas en la validación del tratamiento (T3). Debido a que un tratamiento es la interacción entre el artefacto MoCIP y su contexto de aplicación, se realiza un ciclo empírico para MoCIP y otro ciclo empírico para validar la herramienta ARGON que soporta el enfoque de aprovisionamiento de infraestructura propuesto por MoCIP.

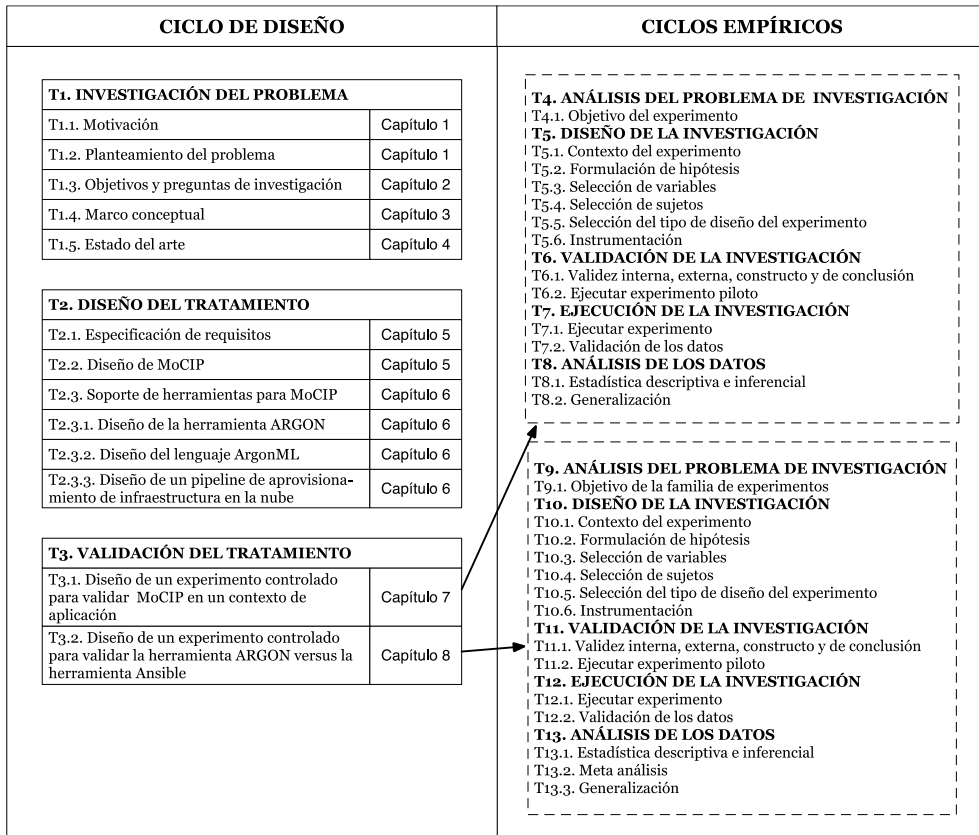


Figura 2-7 Ciclo de ingeniería, ciclo de diseño y ciclos empíricos aplicados a la tesis.

2.5 Resumen

En un proyecto *Design Science* el objeto de estudio es un artefacto en contexto, y sus dos actividades principales son el diseño y la investigación del artefacto en contexto. En consecuencia, el artefacto MoCIP provee soporte al proceso de aprovisionamiento de infraestructura en la nube, mientras que el contexto está definido por los desarrolladores y el personal de operaciones que necesitan definir, actualizar y ejecutar la infraestructura en diferentes proveedores de servicios IaaS. Además, el problema del contexto es la diversidad de lenguajes de scripting para definir, actualizar y ejecutar la infraestructura en la nube y que los proveedores de servicios IaaS ofrecen diferentes tipos de infraestructura. En este escenario, al contexto se lo refina en los aspectos social y de conocimiento. El contexto social contiene a los stakeholders que son afectados o pueden ser afectados por el proyecto, mientras que el contexto de conocimiento consiste en

teorías existentes de ciencia e ingeniería para ayudar a las actividades de diseño e investigación.

En un proyecto *Design Science* se debe diferenciar entre los objetivos del investigador y los objetivos de los stakeholders. Los problemas de diseño requieren un análisis de los objetivos reales o hipotéticos de los stakeholders. En este caso, el proyecto de doctorado se basa en un objetivo hipotético de los stakeholders: *Mejorar el proceso de aprovisionamiento de infraestructura en la nube*. A pesar de que el objetivo propuesto para los stakeholders es muy general, también es un punto de partida para definir la estructura de los objetivos de la investigación. En este escenario, los *objetivos de conocimiento* describen los fenómenos a estudiar, y a partir de estos objetivos se elaboran preguntas de investigación. Para contestar a las preguntas de investigación se desarrollan instrumentos y se establecen los *objetivos de diseño de los instrumentos*. En cambio, el *objetivo de diseño del artefacto* es solucionar, mitigar o mejorar algún problema en el contexto social, mientras que el *objetivo de predicción* intenta predecir cómo será la interacción de un artefacto con el contexto de un problema. Por otro lado, los objetivos de investigación plantean un conjunto de desafíos que deben ser superados y estos desafíos se derivan en preguntas de investigación.

Finalmente, un *ciclo de ingeniería* es un proceso racional de resolución de problemas y su resultado es un *ciclo de diseño* de un tratamiento validado (a través de un *ciclo empírico*) que se transfiere al mundo real, se utiliza y se evalúa. El *ciclo de diseño* tiene tres tareas fundamentales: La *investigación del problema* investiga los fenómenos que hay que mejorar en el problema del contexto, el *diseño del tratamiento* presenta el diseño del artefacto para tratar el problema del contexto y la *validación del tratamiento* realiza la validación empírica del tratamiento para evaluar si el diseño mejora el problema del contexto. En este contexto, es importante definir que un tratamiento es la interacción entre el artefacto y el contexto del problema. Asimismo, un *ciclo empírico* es una forma racional de contestar a las preguntas de conocimiento. Un *ciclo empírico* colabora con la *validación del tratamiento* del *ciclo de diseño*. En este caso, se realizan experimentos controlados para aplicar el ciclo empírico y validar el artefacto MoCIP y su instrumento ARGON en el problema del contexto.

Capítulo 3

Marco Conceptual

En un proyecto *Design Science*, cuando se diseña e investiga un artefacto en un contexto se debe establecer un marco conceptual para definir las estructuras en el artefacto y su contexto. Un marco conceptual es un conjunto de definiciones de conceptos, a menudo llamados *constructos* (Wieringa, 2014). En este capítulo se presentan los *constructos* o conceptos que utiliza el proyecto de doctorado como sustento teórico para el planteamiento del problema y en la solución propuesta.

En la Sección 3.1 se presentan los conceptos básicos que están relacionados con el artefacto y su contexto.

En la Sección 3.2 se presentan los conceptos fundamentales de la computación en la nube, modelos de servicio y modelos de implementación.

En la Sección 3.3 se presenta los conceptos de DevOps, Infraestructura como Código y las prácticas de DevOps.

En la Sección 3.4 se presentan los conceptos sobre la Ingeniería de Software Dirigida por Modelos que se utilizan para proponer un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube.

En la Sección 3.5 se presentan los conceptos sobre experimentos en Ingeniería de Software que se utilizan para evaluar la interacción entre el artefacto y su contexto de aplicación.

3.1 Conceptos básicos

Primero se definen los conceptos básicos que subyacen al artefacto y su contexto. Es importante definir los constructos relacionados con la metodología de investigación. A lo largo de este capítulo se utiliza la letra C seguida de un número para identificar la definición de un concepto (por ejemplo, C1), el nombre del concepto en negrita y subrayado (por ejemplo, **Software**) y una definición.

- C1. **Hardware** es un equipo físico utilizado para procesar, almacenar o transmitir programas o datos informáticos (ISO/IEC *et al.*, 2010).
- C2. **Software** es todo o parte de los programas, procedimiento, reglas y documentación asociada de un sistema de procesamiento de información (ISO/IEC *et al.*, 2010).
- C3. **Software de la aplicación** es un software diseñado para ayudar a los usuarios a realizar tareas particulares o manejar tipos particulares de problemas, a diferencia del software que controla la computadora en sí misma (ISO/IEC *et al.*, 2010)
- C4. **Software del sistema** es un software diseñado para facilitar la operación y el mantenimiento de un sistema informático y sus programas asociados (ISO/IEC *et al.*, 2010).
- C5. El término **tecnología de la información** o TI, significa cualquier equipo o sistema interconectado o subsistema de equipo, que se utiliza en la adquisición, almacenamiento, manipulación, gestión, movimiento, control, visualización, conmutación, intercambio, transmisión, o recepción de datos o información (Barker, 2003).
- C6. La **metodología Design Science** es el diseño y la investigación de un artefacto en contexto (Wieringa, 2014). En particular, la metodología *Design Science* estudia la interacción entre un artefacto y su contexto.
- C7. Un **artefacto** es algo creado por personas con algún propósito práctico (Wieringa, 2014).
- C8. El **stakeholder** de un problema es una persona, grupo de personas o institución afectada por tratar el problema (Wieringa, 2014).
- C9. El **contexto del problema** de un artefacto puede definirse con los stakeholders del artefacto y con el conocimiento utilizado para diseñar el artefacto (Wieringa, 2014).
- C10. El **contexto social** de un proyecto *Design Science* consiste en los stakeholders que pueden afectar el proyecto o que pueden verse afectadas por el proyecto (Wieringa, 2014).
- C11. El **contexto del conocimiento** consiste en teorías existentes de ciencia e ingeniería, especificaciones de diseños conocidos, datos útiles sobre

- productos disponibles, lecciones aprendidas y sentido común (Wieringa, 2014).
- C12. Un **tratamiento** es la interacción entre el artefacto y el contexto del problema (Wieringa, 2014).
- C13. Un **ciclo de ingeniería** es un proceso racional de resolución de problemas (Wieringa, 2014).
- C14. Un **ciclo de diseño** es parte de un ciclo de ingeniería, en el cual el resultado es un tratamiento validado, que se transfiere al mundo real, se usa y se evalúa (Wieringa, 2014).
- C15. Un **ciclo empírico** es una forma racional de responder a preguntas del conocimiento científico (Wieringa, 2014).
- C16. Una **pregunta de conocimiento** pide conocimiento sobre el mundo, sin pedir una mejora en el mundo (Wieringa, 2014).

3.2 Un marco conceptual para la Computación en la Nube

La computación en la nube es una forma especializada de computación distribuida que presenta modelos de utilización para el aprovisionamiento remoto de recursos escalables (Erl *et al.*, 2013). En particular, la computación en la nube se ha convertido en el modelo principal de pago por uso (*pay-as-you-go*) para obtener servicios en la nube en un corto tiempo. A continuación, se definen los *constructos* de la computación en la nube.

- C17. La **computación en la nube** es un modelo para permitir el acceso de red ubicuo, conveniente y bajo demanda a un grupo compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden aprovisionar y liberar rápidamente con un mínimo esfuerzo de gestión o interacción del proveedor de servicios (Mell *et al.*, 2011).
- C18. La **infraestructura** en la nube es la colección de hardware y software que permite las características esenciales de la computación en la nube, tales como autoservicio, acceso a la red, recursos compartidos, elasticidad y medición del servicio (Mell *et al.*, 2011).
- C19. La **elasticidad** es la capacidad automatizada de una nube para escalar de manera transparente los recursos de TI, según se requiera en respuesta a las condiciones de tiempo de ejecución o según lo predeterminado por el consumidor o el proveedor de la nube (Erl *et al.*, 2013).
- C20. Un **proveedor de la nube** es una organización que proporciona recursos de TI basados en la nube (Erl *et al.*, 2013).
- C21. Un **consumidor de la nube** es una organización (o un ser humano) que tiene un contrato formal o acuerdo con un proveedor de la nube para

usar los recursos de TI disponibles por el proveedor de la nube (Erl *et al.*, 2013).

- C22. Un **modelo de servicio** en la nube representa una combinación específica y pre-empaquetada de recursos de TI ofrecidos por un proveedor de la nube (Erl *et al.*, 2013).
- C23. La **Infraestructura como Servicio** (*Infrastructure-as-a-Service*, IaaS) representa un entorno de TI autónomo, compuesto por recursos de TI centrados en la infraestructura a los que se puede acceder y administrar a través de interfaces y herramientas basadas en servicios en la nube (Erl *et al.*, 2013). El modelo de servicio IaaS puede incluir hardware, red, conectividad, sistemas operativos y otros recursos de TI.
- C24. La **Plataforma como un Servicios** (*Platform-as-a-Service*, PaaS) representa un entorno predefinido y listo para usar que generalmente comprende recursos de TI ya implementados y configurados (Erl *et al.*, 2013). El modelo de servicio PaaS se define por el uso de un entorno listo para usar, que establece un conjunto de productos y herramientas pre-empaquetados y que se utilizan para soportar todo el ciclo de vida de la entrega de aplicaciones personalizadas.
- C25. El **Software como Servicio** (*Software-as-a-Service*, SaaS) es un programa de software posicionado como un servicio compartido en la nube o de utilidad genérica (Erl *et al.*, 2013). El modelo de servicio SaaS se utiliza para crear un servicio en la nube reutilizable y que esté ampliamente disponible para una extensa gama de clientes de la nube.
- C26. Un **modelo de implementación** en la nube representa un tipo específico de entorno en la nube, que se distingue principalmente por la propiedad, el tamaño y el acceso (Erl *et al.*, 2013).
- C27. Una **nube pública** es un entorno de nube de acceso público, que es propiedad de un proveedor de nube externo (Erl *et al.*, 2013). La nube pública es un modelo de implementación que ofrece por un costo o importe los recursos de TI a los consumidores de la nube.
- C28. Una **nube privada** es propiedad de una sola organización (Erl *et al.*, 2013). La nube privada es un modelo de implementación que permite que una organización use la tecnología de la computación en la nube como un medio de centralizar el acceso a los recursos de TI por diferentes partes, ubicaciones o departamentos de la organización.
- C29. Una **nube híbrida** es un entorno de nube compuesto por dos o más modelos de implementación en nubes diferentes (Erl *et al.*, 2013). La nube híbrida es un modelo de despliegue que permite, por ejemplo, a un consumidor de la nube optar por implementar servicios en la nube que

procesen datos confidenciales en una nube privada y otros servicios menos sensibles en una nube pública.

- C30. **Amazon Web Services** es una subsidiaria de la compañía Amazon que proporciona plataformas de computación en la nube bajo demanda para individuos, empresas y gobiernos, en una base de pago por el uso (Amazon, 2006).
- C31. **Microsoft Azure** es un servicio de computación en la nube creado por Microsoft para crear, probar, implementar y administrar aplicaciones y servicios a través de centros de datos administrados por Microsoft (Microsoft, 2010).
- C32. **Google Compute Engine** (GCE) es el componente de Infraestructura como Servicio (IaaS) de Google Cloud Platform que se basa en la infraestructura global que ejecuta el motor de búsqueda de Google, Gmail, YouTube y otros servicios (Google Inc., 2012).

3.3 Un marco conceptual para DevOps y la Infraestructura como Código

DevOps (*Development & Operations*) es un conjunto de prácticas destinadas a reducir el tiempo entre la confirmación de un cambio en un sistema y el cambio que se coloca en un entorno de producción, al tiempo que garantiza una alta calidad. (Humble *et al.*, 2010). La Infraestructura como Código es uno de los pilares de DevOps. A continuación, se definen los *constructos* para DevOps y de la Infraestructura como Código:

- C33. **DevOps** promueve la continua colaboración entre los desarrolladores y el personal de operaciones a través de un conjunto de principios, prácticas y herramientas para optimizar el tiempo de entrega de software (Humble *et al.*, 2010).
- C34. La **Infraestructura como Código** (*Infrastructure as Code, IaC*) es un enfoque para la automatización de la infraestructura que utiliza prácticas de desarrollo de software y enfatiza rutinas consistentes y repetibles para aprovisionar y cambiar sistemas software y sus configuraciones (Morris, 2016).
- C35. La **integración continua** es una práctica que consiste en integrar y realizar pruebas con frecuencia en todos los cambios de un sistema a medida que se desarrollan los artefactos de software (Morris, 2016).
- C36. La **entrega continua** es una práctica que garantiza que todo el software de aplicaciones, sistemas e infraestructura se validen continuamente para certificar que estén listos antes de pasar a la etapa de producción (Morris, 2016).

- C37. Un **despliegue continuo** es una práctica que implica realizar automáticamente las pruebas correspondientes sobre los cambios de software incrementales y con frecuencia implementarlos en entornos de producción (Parnin *et al.*, 2017).
- C38. El **aprovisionamiento de infraestructura** se utiliza para hacer que un elemento de la infraestructura como un servidor o dispositivo de red esté listo para su uso (Morris, 2016). En particular, el aprovisionamiento de infraestructura es un proceso para crear, actualizar, configurar o eliminar uno o varios elementos de la infraestructura.
- C39. Un **proceso de aprovisionamiento** efectivo basado en infraestructura como código (IaC) tiene las siguientes características (Morris, 2016): Cualquier elemento de infraestructura existente puede reconstruirse sin esfuerzo bajo demanda. Se puede definir un nuevo elemento una vez, y luego desplegarlo y replicarlo en múltiples instancias y entornos. La definición de cualquier elemento, y el proceso para aprovisionarlo, es transparente y fácil de modificar.

3.4 Un marco conceptual para la Ingeniería de Software Dirigida por Modelos

La ingeniería de software dirigida por modelos ofrece un enfoque prometedor para abordar la incapacidad de los lenguajes de tercera generación en aliviar la complejidad de las plataformas tecnológicas y expresar los conceptos del dominio de una manera más efectiva (Schmidt, 2006). A continuación, se definen los *constructos* para la ingeniería de software dirigida por modelos:

- C40. La **abstracción** consiste en la capacidad de encontrar la similitud en muchas observaciones diferentes y generar así una representación mental de la realidad (Brambilla *et al.*, 2017).
- C41. Un **modelo** es una abstracción de la realidad o simplifica la realidad de algunos fenómenos del mundo real. En particular, un modelo es una representación de un sistema y/o de su entorno (Brambilla *et al.*, 2017).
- C42. El **desarrollo dirigido por modelos** (*Model-Driven Development*, MDD) es un paradigma de desarrollo que utiliza modelos como el artefacto principal del proceso de desarrollo (Brambilla *et al.*, 2017).
- C43. La **ingeniería dirigida por modelos** (*Model-Driven Engineering*, MDE) se puede definir como una metodología para aplicar las ventajas del modelado a las actividades de la ingeniería de software (Brambilla *et al.*, 2017).
- C44. La **arquitectura dirigida por modelos** (*Model-Driven Architecture*, MDA) es la visión particular de MDD propuesta por el Object Management

- Group (OMG) y, por lo tanto, se basa en el uso de los estándares OMG (Brambilla *et al.*, 2017).
- C45. Un **modelo independiente de la computación** (*Computation-Independent Model*, CIM) representa el contexto, los requisitos y el propósito de la solución sin ninguna vinculación con las implicaciones computacionales (Brambilla *et al.*, 2017).
- C46. Un **modelo independiente de la plataforma** (*Platform-Independent Model*, PIM) describe el comportamiento y la estructura de la aplicación, independientemente de la plataforma de implementación (Brambilla *et al.*, 2017).
- C47. Un **modelo específico de plataforma** (*Platform-Specific Model*, PSM) debe contener toda la información requerida sobre el comportamiento y la estructura de una aplicación en una plataforma específica (Brambilla *et al.*, 2017).
- C48. Un **lenguaje de modelado** es una herramienta que permite a los diseñadores especificar los modelos para sus sistemas, en términos de representaciones gráficas o textuales (Brambilla *et al.*, 2017).
- C49. Un **metamodelo** constituye la definición de un lenguaje de modelado, ya que proporciona una forma de describir toda la clase de modelos que ese lenguaje puede representar (Brambilla *et al.*, 2017).
- C50. Un **mapeo** consiste en la definición de las correspondencias entre elementos de dos modelos diferentes (Brambilla *et al.*, 2017). Los mapeos entre los metamodelos permiten construir asignaciones entre modelos.
- C51. En una **transformación de modelos** los modelos se fusionan para homogeneizar diferentes versiones de un sistema, se alinean para crear una representación global del sistema, se refactorizan para mejorar su estructura interna sin cambiar su comportamiento, se refinan para detallar modelos de alto nivel y se traducen a otros lenguajes o representaciones (Brambilla *et al.*, 2017).
- C52. Una **transformación de modelo a modelo** (*Model-to-Model*, M2M) es un programa que toma uno o más modelos como entrada para producir uno o más modelos como salida (Brambilla *et al.*, 2017).
- C53. Una **transformación de modelo a texto** (*Model-to-Text*, M2T) se refieren principalmente a la generación de código para lograr la transición del nivel de modelo al nivel de código (Brambilla *et al.*, 2017).
- C54. Un **lenguaje específico de dominio** (*Domain-Specific Language*, DSL) es un lenguaje diseñado específicamente para un determinado dominio, contexto o empresa para facilitar la tarea de las personas que necesitan describir cosas en ese dominio (Brambilla *et al.*, 2017).

- C55. La **sintaxis abstracta** describe la estructura del lenguaje y la forma en que se pueden combinar diferentes primitivas, independientemente de cualquier representación o codificación particular (Brambilla *et al.*, 2017).
- C56. La **sintaxis concreta** es la descripción de representaciones específicas del lenguaje de modelado, cubriendo problemas de codificación y apariencia visual (Brambilla *et al.*, 2017).

3.5 Un marco conceptual para experimentos en Ingeniería de Software

Los estudios empíricos son cruciales para la evaluación de procesos y actividades humanas. En el caso de la ingeniería de software es beneficioso utilizar estudios empíricos cuando es necesario evaluar el uso de productos o herramientas de software. En este contexto, la experimentación proporciona una forma sistemática, disciplinada, cuantificable y controlada de evaluar actividades basadas en humanos. A continuación, se definen los *constructos* para definir experimentos en ingeniería de software:

- C57. Una **hipótesis** es una teoría provisional o una suposición que se cree que explica un comportamiento que se pretende explorar (Genero *et al.*, 2014).
- C58. Una **variable** es una característica o un valor numérico que varía para cada individuo (Kuehl, 2000).
- C59. Una **variable independiente** es una variable cuyos valores cambian para estudiar que efecto producen esos cambios (Genero *et al.*, 2014).
- C60. Un **tratamiento** es cada uno de los posibles valores que toma una variable independiente (Genero *et al.*, 2014).
- C61. Una **variable dependiente** es una variable que se estudia para comprobar el efecto de los cambios en las variables independientes (Genero *et al.*, 2014).
- C62. Un **experimento** en ingeniería de software es una investigación empírica que manipula una variable (denominada independiente o factor) del entorno o fenómeno estudiado, midiendo el efecto que tiene sobre otra variable denominada dependiente (Genero *et al.*, 2014).
- C63. Un **sujeto** es aquella persona que aplica los tratamientos del experimento (Genero *et al.*, 2014).
- C64. La **aleatorización** es la asignación de manera aleatoria de los tratamientos a los sujetos, y también a la selección de los sujetos de manera aleatoria dentro de un muestra representativa (Genero *et al.*, 2014).

- C65. Un **objeto experimental** es una herramienta usada para verificar la relación causa-efecto en una teoría (Genero *et al.*, 2014).
- C66. Una **población** es un grupo de elementos o unidades que se desea estudiar para responder a una pregunta de investigación (Kuehl, 2000).
- C67. Una **muestra** es un subconjunto de una población que cumple ciertos criterios (Kuehl, 2000).
- C68. El **balanceo** es el principio que se cumple cuando a cada tratamiento se le asigna el mismo número de sujetos (Genero *et al.*, 2014).
- C69. La **validez interna** de un experimento define el grado de confianza en una relación causa-efecto entre los factores de interés y los resultados observados, es decir, el grado en el que se pueden extraerse conclusiones en la relación variables independientes-dependientes (Genero *et al.*, 2014).
- C70. La **validez externa** en un experimento representa el grado hasta el que los resultados alcanzados pueden generalizarse teniendo en cuenta la población utilizada y otros parámetros de la investigación (Genero *et al.*, 2014).
- C71. La **validez del constructo** en un experimento define hasta donde las variables miden correctamente los constructos teóricos de las hipótesis (Genero *et al.*, 2014).
- C72. La **validez de la conclusión** en un experimento define hasta dónde las conclusiones son estadísticamente válidas, es decir, cuán correcta es la conclusión de la relación entre el tratamiento y la variable dependiente (Genero *et al.*, 2014).
- C73. Una **encuesta** es un sistema para recopilar información de o sobre personas para describir, comparar o explicar sus conocimientos, actitudes y comportamiento (Wohlin *et al.*, 2012).
- C74. La **replicación de un experimento** implica repetir la investigación en condiciones similares, mientras que, por ejemplo, varía la población de sujetos (Wohlin *et al.*, 2012).
- C75. Un **diseño cruzado** es un tipo particular de diseño donde cada sujeto experimental aplica todos los tratamientos, pero diferentes sujetos aplican tratamientos en un orden diferente (Vegas *et al.*, 2016).
- C76. Un **período** se define por la aplicación de un tratamiento por un sujeto a un objeto experimental (Vegas *et al.*, 2016).
- C77. Una **sesión** es una parte del tiempo que un sujeto dedica a completar (una o más) tareas experimentales (Vegas *et al.*, 2016).
- C78. La **secuencia** es el orden en que los sujetos aplican los tratamientos (Vegas *et al.*, 2016).

- C79. La **transferencia** o *carryover* es una amenaza de validez interna de los diseños cruzados (Vegas *et al.*, 2016). Ocurre cuando se administra un tratamiento antes de que el efecto de otro tratamiento administrado previamente haya desaparecido por completo.

Capítulo 4

Estado del Arte

En este capítulo se analizan los marcos, técnicas, y herramientas existentes para el aprovisionamiento de infraestructura y el despliegue de aplicaciones en la nube. Se consideran los trabajos de investigación sobre la Infraestructura como Código (*Infrastructure as Code*, IaC). Además, se toma en cuenta los trabajos relacionados con IaC que utilicen la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering*, MDE) para abstraer y automatizar las tareas de aprovisionamiento de infraestructura y despliegue de aplicaciones en la nube.

En la Sección 4.1 se explican las librerías y las herramientas IaC de la comunidad DevOps que realizan el aprovisionamiento de infraestructura y el despliegue de aplicaciones en la nube.

En la Sección 4.2 se revisan los trabajos de investigación que utilizan IaC y MDE para aprovisionar la infraestructura y desplegar aplicaciones en la nube.

En la Sección 4.3 se presenta el estudio de un mapeo sistemático sobre los trabajos de investigación que utilizan IaC en diferentes áreas.

En la Sección 4.4 se presenta un resumen sobre estado del arte.

4.1 Librerías y herramientas IaC

La Infraestructura como Código (IaC) es un enfoque para la automatización de la infraestructura que utiliza prácticas de desarrollo de software y enfatiza rutinas consistentes y repetibles para aprovisionar y cambiar sistemas software y sus configuraciones (Morris, 2016). En este escenario, es importante diferenciar entre aprovisionamiento y despliegue. Por un lado, el aprovisionamiento se refiere a escribir y ejecutar código para definir, crear y actualizar la infraestructura. Por otro lado, el despliegue se refiere a escribir y ejecutar código para definir e implementar las aplicaciones de software en la infraestructura aprovisionada. La comunidad DevOps provee una variedad de herramientas para diferentes tareas. El propósito de esta sección es ofrecer una visión general de las herramientas IaC en lugar de nombrar todas las herramientas existentes. Además, es importante mencionar que el enfoque IaC se puede aplicar a la infraestructura en varios escenarios, tales como computación en la nube, sistemas virtualizados y en un hardware físico.

En los últimos años, se han desarrollado conceptos y tecnologías para admitir el aprovisionamiento, el despliegue y la adaptación de aplicaciones en la nube (Nitto *et al.*, 2017). En este caso, se aprovecha la interfaz de programación de aplicaciones (*Application Programming Interface*, API) que proveen las librerías de software para el despliegue y control de aplicaciones, tales como:

- jcloud (Apache Software Foundation, 2011) es una librería de código abierto para la plataforma Java que permite la creación de aplicaciones portables a través de múltiples nubes.
- δ -cloud (Apache Software Foundation, 2009a) proporciona un API y los controladores necesarios para conectarse a proveedores de servicios en la nube. La librería mantiene la estabilidad a largo plazo de los scripts, herramientas, aplicaciones y la compatibilidad con diferentes versiones. Además, puede iniciar una instancia o máquina virtual.
- libcloud (Apache Software Foundation, 2009b) es un librería de Python que utiliza una API para crear aplicaciones que funcionen en diferentes proveedores de servicios en la nube.
- fog (Beary, 2012) es una librería de servicios en la nube desarrollada en Ruby que facilita el uso y la migración de los servicios en la nube entre diferentes proveedores. Además, evalúa los servicios para encontrar la mejor configuración y facilitar su funcionamiento.

Las librerías mencionadas anteriormente proporcionan un acceso común a varios proveedores de servicios en la nube, pero dependen de un lenguaje de programación. Por otro lado, el principal objetivo de la Infraestructura como

Código es administrar casi todo utilizando código, incluidos servidores, bases de datos, redes, archivos de registro, configuración de aplicaciones, documentación, pruebas automatizadas, procesos de implementación, etc. En este contexto, de acuerdo con (Brikman, 2017) existe cuatro grandes categorías de herramientas IaC:

- *Scripts Ad Hoc*. Es el enfoque más directo para automatizar cualquier tarea de aprovisionamiento de infraestructura. En este caso, se utiliza lenguajes de programación, tales como Bash, Ruby o Python para definir en código cada tarea de aprovisionamiento y ejecutar el script en un servidor. La ventaja de utilizar scripts Ad Hoc es utilizar lenguajes de programación populares de propósito general. La desventaja es que mientras las herramientas IaC proporcionan APIs concisas para realizar tareas complicadas, con los lenguajes de programación se debe escribir un código completamente personalizado para cada tarea.
- Herramientas de gestión de la configuración (*Configuration Management Tools*, CMT). Las CMT están diseñadas para instalar y administrar software en servidores existentes, tales como máquinas virtuales aprovisionadas en un proveedor de servicios en la nube. Por ejemplo, entre las CMT más populares están Chef (Chef, 2009), Puppet (Puppet Labs, 2005) y Ansible (DeHaan, 2012). El código es similar a un script Bash, pero al usar una CMT como Ansible existe un número de ventajas, tales como (1) las convenciones de codificación facilitan la navegación del código, (2) la idempotencia permite que el código funcione correctamente sin importar cuantas veces se ejecute, y (3) la distribución permite administrar servidores remotos.
- Herramientas de plantillas de servidor (*Server Templating Tools*, STT). El objetivo de las STT es crear una imagen de un servidor que capture completamente una “instantánea” autónoma del sistema operativo, el software, los archivos y todos los detalles relevantes. En este caso, se utilizan herramientas como Docker (Hykes, 2013), Packer (HashiCorp Inc, 2012) o Vagrant ((Hashimoto et al., 2010)). Para instalar la imagen de un sistema operativo se utilizan otras herramientas IaC, por ejemplo, la herramienta Ansible.
- Herramientas de aprovisionamiento de servidores (*Servers Provisioning Tools*, SPT). Las SPT son responsables de crear los servidores —es decir, máquinas virtuales— y toda la infraestructura subyacente. De hecho, se puede usar herramientas de aprovisionamiento no únicamente para crear servidores, sino también para crear bases de datos, cachés, balanceadores de carga, colas, monitoreo, configuraciones de subred, configuraciones de firewall, reglas de enrutamiento, certificados SSL y casi cualquier otro

aspecto de su infraestructura. En este caso, se utilizan herramientas como Ansible (DeHaan, 2012), Terraform (HashiCorp, 2017), CloudFormation (Amazon Web Services, 2011) y OpenStack Heat (OpenStack Foundation, 2013).

4.2 Trabajos de investigación que utilizan IaC y MDE

El propósito del proyecto doctoral es mejorar el aprovisionamiento de la infraestructura en la nube utilizando técnicas dirigidas por modelos. En consecuencia, en esta sección se presenta un análisis de los trabajos de investigación que abordan la Infraestructura como Código (IaC) mediante la Ingeniería de Software Dirigida por Modelos (MDE). La Tabla 4-1 presenta las publicaciones que utilizan IaC y MDE. La primera columna representa la indexación de cada publicación como “E#”, por ejemplo, el índice “E8” se refiere a la publicación “*Model-Driven Continuous Deployment for Quality DevOps*”. La segunda columna muestra los autores de los trabajos relacionado junto con el año de publicación. La tercera columna presenta el título de la publicación.

Las publicaciones E2, E3, E9 y E12 pertenecen a proyectos europeos, tales como, PaaSage European Project, ENVISAGE FP7 European Project y DICE H2020 European Project. En consecuencia, los proyectos de investigación relacionados con IaC y MDE tienen relevancia e importancia en la comunidad académica.

Las publicaciones E5, E6 y E15 estudian el aprovisionamiento de infraestructura, mientras que todas las publicaciones (E1-E15) abordan el despliegue de aplicaciones. Adicionalmente, la publicación E9 realiza el monitoreo de aplicaciones desplegadas en la nube. Es importante mencionar que E5 únicamente realiza el aprovisionamiento de máquinas virtuales, mientras que E6 y E15 centran sus esfuerzos en el aprovisionamiento de contenedores Docker (Hykes, 2013).

Las publicaciones E1, E2, E3, E5, E7, E8, E11, E12 y E15 afrontan el problema de la heterogeneidad de los lenguajes de scripting para definir, configurar y ejecutar el aprovisionamiento de infraestructura y el despliegue de aplicaciones. Los autores resaltan que utilizar lenguajes de scripting para definir y ejecutar la infraestructura y aplicaciones es una tarea lenta y propensa a errores. En este caso, el uso de lenguajes de scripting requiere experiencia técnica y conocimientos en el dominio y en muchos casos un tiempo considerable de aprendizaje.

Tabla 4-1 Lista de publicaciones que utilizan IaC y MDE

Índice	Autor(es)	Publicación
E1	(Wettinger <i>et al.</i> , 2013)	Integrating Configuration Management with Model-Driven Cloud Management Based on TOSCA
E2	(Rossini, 2015)	Cloud Application Modelling and Execution Language (CAMEL) and the PaaSage Workflow
E3	(Gouw <i>et al.</i> , 2015)	On the Integration of Automatic Deployment into the ABS Modeling Language
E4	(Marrone <i>et al.</i> , 2015)	Automatic Resource Allocation for High Availability Cloud Services
E5	(Glaser, 2016)	Domain Model Optimized Deployment and Execution of Cloud Applications with TOSCA
E6	(Weerasiri <i>et al.</i> , 2016)	A Model-Driven Framework for Interoperable Cloud Resources Management
E7	(Chen <i>et al.</i> , 2016)	MORE: A Model-driven Operation Service for Cloud-based IT Systems
E8	(Artač <i>et al.</i> , 2016)	Model-Driven Continuous Deployment for Quality DevOps
E9	(Nitto <i>et al.</i> , 2017)	Model-Driven Development and Operation of Multi-Cloud Applications
E10	(Alipour <i>et al.</i> , 2018)	Model Driven Deployment of Auto-scaling Services on Multiple Clouds
E11	(Bhattacharjee <i>et al.</i> , 2018)	CloudCAMP: Automating the Deployment and Management of Cloud Services
E12	(Artac <i>et al.</i> , 2018)	Infrastructure-as-Code for Data-Intensive Architectures: A Model-Driven Development Approach
E13	(Ferry <i>et al.</i> , 2018)	CloudMF: Model-Driven Management of Multi-Cloud Applications
E14	(Casale <i>et al.</i> , 2019)	RADON: rational decomposition and orchestration for serverless computing
E15	(Brabra <i>et al.</i> , 2019)	Model-Driven Orchestration for Cloud Resources

Por otro lado, las publicaciones E6, E9 y E13 abordan el problema de la diversidad de las tecnologías entre los proveedores de servicios en la nube y la interoperabilidad entre las soluciones de nube existentes. En cambio, los autores en E4 afrontan la integración de las diferentes fases del ciclo de vida de los servicios en la nube, mientras que los autores en E10 abordan el escalado automático de servicios entre múltiples plataformas en la nube. En E14, los autores enfrentan la falta de metodologías para obtener ventajas de las funciones como servicio (*Function as a Service*, FaaS) en la nube.

Con respecto a las soluciones propuestas, las publicaciones E2, E3, E6, E7, E8, E9, E10, E11, E13 y E14 proponen un lenguaje de dominio específico (*Domain-Specific Language*, DSL) para modelar el despliegue de aplicaciones en la nube. Los autores en E4 utilizan un perfil de UML para modelar y analizar sistemas en tiempo real, mientras que los autores en E12 utilizan un perfil de UML para modelar la implementación de arquitecturas intensivas de datos. Por otro lado, en E1 se plantea la integración de la gestión de la configuración y la implementación de los servicios en la nube, mientras que en E5 propone un marco (*framework*) para escalar las aplicaciones en la nube de acuerdo con las demandas del modelo de dominio de los usuarios. Los autores en E8 y E15 proponen un mapeo y traducción desde los recursos de la nube definidos en TOSCA (OASIS, 2013) hacia herramientas de la comunidad DevOps.

En cuanto a las técnicas MDE, las publicaciones E2, E3, E4, E6, E7, E8, E9, E11, E12, E13 y E14 utilizan modelos para representar en un alto nivel de abstracción la infraestructura, aplicaciones y configuraciones. En E2 y E13 los autores utilizan modelos en tiempo de ejecución (*model@run-time*) para representar en un modelo el sistema en ejecución. Además, mediante *model@run-time* una modificación en el modelo se promulga hacia el sistema y un cambio en el sistema se refleja automáticamente en el modelo. Por otro lado, las publicaciones E1, E5, E8 y E15 utilizan transformaciones de modelo a texto (*Model-to-Text*, M2T) para convertir modelos TOSCA en scripts para herramientas IaC, mientras que las publicaciones E7, E12, y E14 utilizan transformaciones M2T para convertir los modelos de infraestructura, aplicaciones y configuraciones en scripts para herramientas IaC. En este contexto, en E9, E10, E13 y E15 los autores utilizan y adaptan la arquitectura dirigida por modelos (*Model-Driven Architecture*, MDA) para definir en un modelo independiente de la nube (*Cloud Independent Model*, CIM) los requisitos de la aplicación, luego en un modelo independiente del proveedor de la nube (*Cloud Provider Independent Model*, CPIM) se definen aspectos generales de la nube y finalmente en un modelo específico del proveedor de la nube (*Cloud Provider Specific Model*, CPSM) se definen aspectos específicos de un proveedor de la nube. Por un lado, se utiliza transformaciones de modelo a modelo (*Model-to-Model*,

M2M) para pasar desde un modelo CIM hacia un modelo CPIM y luego hacia un modelo CPSM. Por otro lado, se utilizan transformaciones M2T para generar scripts para herramientas IaC a partir de un modelo CPSM.

En cuanto a las herramientas IaC, las publicaciones E1, E2, E6, E8, E11, E12, E14 y E15 utilizan TOSCA (OASIS, 2013) para describir la interoperabilidad de los servicios y aplicaciones alojadas en la nube, así como sus componentes, relaciones, dependencias, requisitos y capacidades, lo que permite la portabilidad y la gestión automatizada entre los proveedores de servicios en la nube, independientemente de la plataforma o infraestructura subyacente. En E1, E8 y E12 los autores utilizan la herramienta Chef (Chef, 2009) para instalar y configurar las aplicaciones de software, mientras que en E7, E9 y E13 los autores emplean la herramienta Puppet (Puppet Labs, 2005) para instalar y gestionar las aplicaciones de software.

Por otro lado, las publicaciones E5, E8 y E12 emplean Cloudify (Cloudify, 2012) como un marco de orquestación para el despliegue de aplicaciones en la nube y para la automatización de su ciclo de vida. En E6 y E15 los autores utilizan Juju (Canonical, 2011) para operar las aplicaciones de software en servidores, máquinas virtuales y en contenedores. En E5 se utiliza Ansible (DeHaan, 2012) para instalar y configurar aplicaciones de software y para aprovisionar máquinas virtuales en la nube, mientras que en E15 se emplea Terraform (HashiCorp, 2017) para aprovisionar contenedores. Finalmente, en E5 los autores utilizan OpenStack (OpenStack Foundation, 2010) como un proveedor de infraestructura como un servicio (*Infrastructure as a Services*, IaaS), mientras que en E6 y E15 los autores emplean contenedores Docker (Hykes, 2013) para instalar las aplicaciones de software.

4.3 Mapeo sistemático sobre IaC

Un mapeo sistemático sobre los trabajos relacionados con la Infraestructura como Código (IaC) ayuda a los investigadores a identificar posibles áreas de investigación. En consecuencia, a continuación se presenta el estudio de un mapeo sistemático realizado por (Rahman *et al.*, 2019).

La Tabla 4-1 presenta las 32 publicaciones del mapeo sistemático. La primera columna representa la indexación de cada publicación como “S#”, por ejemplo, el índice “S5” se refiere a la publicación “*Testing Idempotence for Infrastructure as Code*”. La segunda columna muestra los autores de los trabajos relacionado junto con el año de publicación. La tercera columna presenta el título de la publicación.

Tabla 4-2 Lista de publicaciones del mapeo sistemático (Rahman *et al.*, 2019)

Índice	Autor(es)	Publicación
S1	(Scheuner <i>et al.</i> , 2015)	Cloud WorkBench: Benchmarking IaaS Providers based on Infrastructure-as-Code
S2	(Jiang <i>et al.</i> , 2015)	Co-evolution of infrastructure and source code: An empirical study
S3	(Artac <i>et al.</i> , 2017)	DevOps: Introducing Infrastructure-as-Code
S4	(Ikeshita <i>et al.</i> , 2017)	Test Suite Reduction in Idempotence Testing of Infrastructure as Code
S5	(Hummer <i>et al.</i> , 2013)	Testing Idempotence for Infrastructure as Code
S6	(Hanappi <i>et al.</i> , 2016)	Asserting reliable convergence for configuration management scripts
S7	(Singh <i>et al.</i> , 2015)	Automated provisioning of application in IAAS cloud using Ansible configuration management
S8	(Sharma <i>et al.</i> , 2016)	Does your configuration code smell?
S9	(Hintsch <i>et al.</i> , 2016)	Modularization of Software as a Service Products: A Case Study of the Configuration Management Tool Puppet
S10	(Weiss <i>et al.</i> , 2017)	Tortoise: interactive system configuration repair
S11	(Hummer <i>et al.</i> , 2013)	Automated testing of chef automation scripts
S12	(Wettinger, Breitenbucher, <i>et al.</i> , 2014)	Standards-Based DevOps Automation and Integration Using TOSCA
S13	(Adams <i>et al.</i> , 2016)	Modern Release Engineering in a Nutshell –Why Researchers Should Care
S14	(Spinellis, 2012)	Don't Install Software by Hand
S15	(Miglierina, 2015)	Application Deployment and Management in the Cloud
S16	(Kecskemeti <i>et al.</i> , 2016)	Towards a Methodology to Form Microservices from Monolithic Ones
S17	(Wettinger, Breitenbücher, <i>et al.</i> , 2014)	DevOpSlang - Bridging the Gap between Development and Operations
S18	(Gouw <i>et al.</i> , 2015)	On the Integration of Automatic Deployment into the ABS Modeling Language

Continuación de la Tabla 4-2.

Índice	Autor(es)	Publicación
S19	(Wettinger, Breitenbücher, <i>et al.</i> , 2015)	Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel
S20	(Dolstra <i>et al.</i> , 2013)	Charon: declarative provisioning and deployment
S21	(Kathiravelu <i>et al.</i> , 2016)	SENDIM for Incremental Development of Cloud Networks: Simulation, Emulation and Deployment Integration Middleware
S22	(Karvinen <i>et al.</i> , 2017)	Investigating survivability of configuration management tools in unreliable and hostile networks
S23	(Parnin <i>et al.</i> , 2017)	The Top 10 Adages in Continuous Deployment
S24	(Baset <i>et al.</i> , 2017)	Usable declarative configuration specification and validation for applications, systems, and cloud
S25	(Lwakatare <i>et al.</i> , 2015)	Dimensions of DevOps
S26	(Wettinger, Andrikopoulos, <i>et al.</i> , 2015)	Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications
S27	(Miglierina <i>et al.</i> , 2017)	Towards Omnia: A Monitoring Factory for Quality-Aware DevOps
S28	(Virmani, 2015)	Understanding DevOps & bridging the gap from continuous integration to continuous delivery
S29	(Aiftimiei <i>et al.</i> , 2017)	Cloud Environment Automation: from infrastructure deployment to application monitoring
S30	(Sandobalín <i>et al.</i> , 2017a)	An Infrastructure Modeling Tool for Cloud Provisioning
S31	(Sandobalín <i>et al.</i> , 2017b)	End-to-End Automation in Cloud Infrastructure Provisioning
S32	(Scheuner <i>et al.</i> , 2014)	Cloud Work Bench - Infrastructure-as-Code Based Cloud Benchmarking

4.3.1 Categorización de las publicaciones relacionadas con IaC

Los autores del mapeo sistemático (Rahman *et al.*, 2019) identificaron y categorizaron los temas que se han investigado en el área de IaC mediante la aplicación de un análisis cualitativo. En este caso, una publicación puede pertenecer a múltiples categorías, lo que implica que los temas identificados no son ortogonales entre sí. La Tabla 4-3 presenta cada categoría junto con el recuento de sus publicaciones.

Tabla 4-3 Mapeo entre tema y publicación (Rahman *et al.*, 2019)

Categoría	Publicación
Herramienta/Marco para IaC	S6, S10, S12, S13, S15, S18, S19, S20, S21, S22, S24, S25, S26, S29, S30, S31
Adopción de IaC	S1, S3, S7, S9, S14, S16, S17, S18, S23, S27, S28, S32
Estudios empíricos sobre IaC	S2, S4, S5, S8, S10, S11, S23
Pruebas en IaC	S4, S5, S6, S11

4.3.1.1 Herramientas o marcos para IaC

El tema más investigado en el estudio del mapeo sistemático —con un total de 16 publicaciones— está relacionado con las herramientas o marcos para IaC.

En el caso de las publicaciones relacionadas con herramientas IaC, las publicaciones S12, S18 y S19 presentan una clasificación de las herramientas DevOps y cómo estas pueden transformarse hacia el estándar TOSCA (OASIS, 2013). En cambio, los autores de S15, S20 y S29 proponen herramientas para mejorar el aprovisionamiento de la infraestructura y el despliegue de aplicaciones en la nube. S24 propone un lenguaje de validación para detectar configuraciones incorrectas que pueden afectar la seguridad, el rendimiento y la funcionalidad de la infraestructura, mientras que S31 propone un lenguaje específico de dominio para modelar la infraestructura de la nube.

Con respecto a los marcos para IaC, S13 propone una ingeniería de liberación de pipelines (*release engineering pipeline*), mientras que S21 presenta un middleware de integración, simulación, emulación e implementación para redes (*networking*) en la nube. Los autores en S25 presentan un marco conceptual basado en cuatro dimensiones (tales como colaboración, automatización, medición y monitoreo) para facilitar la adopción de DevOps, mientras que los autores en S26 proponen un enfoque colaborativo y holístico para capturar el conocimiento de DevOps en una base de conocimiento.

En cuanto a la reparación de scripts IaC, S10 propone una técnica para corregir los errores en los scripts IaC utilizando un Shell. Este enfoque clasifica y presenta múltiples reparaciones en scripts IaC y admite todos los Shell que el administrador desee utilizar.

Respecto a la confiabilidad de scripts IaC, S6 presenta un marco conceptual para afirmar una convergencia confiable en la gestión de la configuración. El marco conceptual utiliza una definición formal de los scripts de configuración, sus recursos y gráficos de transición de estado para probar si un script hace que el sistema converja al estado deseado en diferentes condiciones. Por otro lado, los autores en S22 centran su investigación en redes de programas malignos (*malware*) que sobreviven en redes hostiles y heterogéneas.

4.3.1.2 *Adopción de IaC*

Las publicaciones que se relacionan con esta categoría discuten cómo se puede usar IaC en diferentes dominios de ingeniería de software, tales como el monitoreo de sistemas y la implementación de aplicaciones empresariales.

La publicación S1 propone una herramienta para evaluar el rendimiento de las aplicaciones en la nube. En cambio, los autores en S3 y S14 discuten cómo se puede usar IaC para implementar DevOps. S7 se centra en cómo se puede usar Ansible (DeHaan, 2012) para aprovisionar automáticamente una aplicación empresarial, mientras que S9 investiga la viabilidad de utilizar módulos Puppet (Puppet Labs, 2005) para implementar una aplicación de software como un servicio (SaaS). Además, en S9 se observó que los módulos Puppet son adecuados para el aprovisionamiento de aplicaciones SaaS, pero tienen una capa adicional de complejidad. Los autores en S17 proponen “DevOpsLang” que utiliza Chef (Chef, 2009) para implementar automáticamente una aplicación de chat, mientras que los autores en S18 proponen el lenguaje de modelado ABS (*Abstract Behavioral Specification*) que usa IaC para implementar una aplicación de comercio electrónico. Por otro lado, los autores en S23 entrevistaron a profesionales de 10 compañías sobre el uso de IaC en la práctica del despliegue continuo. Finalmente, los autores en S27 proponen “Omnia” que usa IaC para crear un marco para monitorear las operaciones de DevOps.

4.3.1.3 *Estudios empíricos relacionados con IaC*

Las publicaciones que pertenecen a esta categoría emplean análisis empíricos para investigar sobre scripts IaC.

Con respecto a las publicaciones relacionadas con pruebas de scripts IaC, en S4 se presenta un método para verificar de manera eficiente la idempotencia de los scripts IaC combinando los enfoques de prueba y verificación estática. En S5,

los autores plantean un marco de pruebas basado en modelos para scripts IaC y proponen casos de prueba en máquinas virtuales utilizando la herramienta Chef (Chef, 2009). En cambio, S11 proponen una herramienta que trabaja con Chef para inspeccionar los cambios de estado de un sistema durante su ejecución.

En el caso de la coevolución de scripts IaC, en S2 se realizó un estudio empírico de un sistema de control de versiones de 265 proyectos OpenStack (OpenStack Foundation, 2010). Los autores descubrieron que los archivos de infraestructura son grandes y cambian con frecuencia, lo que podría indicar la posibilidad de introducir errores. Además, los archivos IaC están estrechamente vinculados con los otros archivos en un proyecto, especialmente los archivos de prueba, lo que implica que los ingenieros de pruebas a menudo necesitan cambiar las especificaciones de infraestructura al realizar cambios en el marco de las pruebas.

En cuanto a las publicaciones sobre la calidad del código de los scripts IaC, los autores en S8 proponen un catálogo de 13 problemas de configuración de implementación y 11 de diseño, donde cada problema viola las mejores prácticas recomendadas para los scripts IaC. En cambio, los autores en S10 presentan una técnica basada en síntesis que permite corregir los errores en scripts IaC utilizando un Shell para reparar automáticamente la especificación de IaC escrita en un lenguaje de scripting.

Los autores en S23 entrevistaron a profesionales de 10 compañías sobre el uso de scripts IaC en el despliegue continuo de aplicaciones software. Los autores informaron que los scripts IaC han cambiado la forma en que las organizaciones de TI administran su infraestructura utilizando IaC.

4.3.1.4 Pruebas con IaC

Las publicaciones realizadas con esta categoría abordan el tema de las pruebas de scripts IaC.

En el caso de las publicaciones relacionadas con la idempotencia de script IaC, es importante mencionar que la idempotencia es una propiedad que permite ejecutar varias veces un script IaC y aún así conseguir el resultado que se obtendría si se ejecuta el script IaC una sola vez (Hummer *et al.*, 2013). En S5, se presenta un método para verificar de manera eficiente la idempotencia combinando los enfoques de prueba y verificación estática.

En el caso de la reducción de casos de prueba de los scripts IaC, en S4 los autores informaron que el método sugerido en S5 genera demasiados casos de prueba y propusieron un enfoque para reducir la cantidad de casos de prueba para probar la idempotencia en scripts IaC.

En cuanto a las publicaciones relacionadas con marcos de pruebas IaC, en S6 los autores proponen un marco conceptual para validar una convergencia confiable de los scripts IaC de la herramienta Puppet (Puppet Labs, 2005) en la gestión de la configuración. En cambio, en S11 los autores proponen un marco para garantizar que un sistema converja de manera confiable con los estados inicial, intermedio y final deseados.

4.4 Resumen

La comunidad DevOps han desarrollado conceptos, tecnologías y herramientas para apoyar el aprovisionamiento, el despliegue y la adaptación de aplicaciones en la nube. En este escenario, la Infraestructura como Código (IaC) cambió la forma de definir, actualizar y ejecutar la infraestructura en diferentes entornos, tales como máquinas virtuales, servidores físicos y contenedores. Por otro lado, la Ingeniería de Software Dirigida por Modelos (MDE) ayuda a abstraer y automatizar las prácticas de IaC. Por tal motivo, en este capítulo se analizó quince trabajos de investigación que utilizan IaC y MDE.

El principal problema que abordan los trabajos de investigación relacionados con IaC y MDE es la heterogeneidad de los lenguajes de scripting para definir, configurar y ejecutar el aprovisionamiento de infraestructura y el despliegue de aplicaciones. Todos los trabajos centran sus esfuerzos en mejorar el despliegue de aplicaciones en la nube, mientras que solo tres investigan el aprovisionamiento de máquinas virtuales y contenedores. Existen nueve trabajos relacionados que emplean modelos como una solución para definir la infraestructura, aplicaciones y configuraciones. En cambio, cinco trabajos de investigación utilizan y adaptan la Arquitectura Dirigida por Modelos (MDA) para generar artefactos IaC mediante transformaciones de modelos y brindar soporte al despliegue de aplicaciones en la nube. Como resultado, los investigadores han centrado sus esfuerzos en mejorar la entrega continua de aplicaciones en la nube, pero existe una brecha que cubrir en el aprovisionamiento de infraestructura en nube. No se ha encontrado una solución para *abstraer* la complejidad de especificar los recursos de infraestructura de diversos proveedores de servicios IaaS, así como no se ha encontrado una solución para *automatizar* la generación de scripts para el aprovisionamiento y configuración de la infraestructura en la nube.

Con respecto al estudio del mapeo sistemático realizado por (Rahman *et al.*, 2019), se analizaron 32 trabajos de investigación relacionados con IaC. Los autores categorizaron las temáticas analizadas en el estudio. Los trabajos relacionados con IaC pueden pertenecer a múltiples categorías, lo que implica que los temas identificados no son ortogonales entre sí. La primera categoría

tiene trabajos relacionados con herramientas y marcos (*framework*) para implementar las prácticas IaC, ampliar una funcionalidad de IaC, reparar los scripts IaC y verificar la confiabilidad de los scripts IaC. La segunda categoría investiga cómo se puede usar IaC en diferentes dominios de la ingeniería de software, tales como el aprovisionamiento de infraestructura, despliegue de aplicaciones, benchmarking, microservicios y el uso de IaC para implementar DevOps. La tercera categoría tiene estudios empíricos relacionados con pruebas en scripts IaC, coevolución de los scripts IaC con otros artefactos de software, calidad de los scripts IaC y encuestas realizadas a profesionales. La cuarta categoría realiza pruebas en los scripts IaC para estudiar la idempotencia de los scripts IaC, los marcos de creación de pruebas para IaC y la reducción de casos de pruebas para scripts IaC.

A pesar de que el mapeo sistemático aporta conocimientos importantes sobre el estado actual de la investigación, también es una importante herramienta para identificar posibles áreas de investigación. Los resultados muestran una brecha en los estudios empíricos realizados. No se han encontrado estudios empíricos que evalúen el rendimiento de una persona al definir un script IaC y tampoco se evalúa la percepción de los usuarios después de utilizar una herramienta IaC.

Capítulo 5

El Enfoque MoCIP

En este capítulo se presenta el enfoque MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*) para proveer soporte a la Infraestructura como Código (*Infrastructure as Code, IaC*) en el aprovisionamiento de infraestructura en la nube. MoCIP aprovecha la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering, MDE*) para abstraer la complejidad de utilizar los lenguajes de scripting de las herramientas IaC en el proceso de definir, configurar y ejecutar la infraestructura en los proveedores de servicios IaaS (*Infrastructure as a Service*).

En la Sección 5.1 se presentan las ventajas de utilizar técnicas dirigidas por modelos (MDE) en el aprovisionamiento de infraestructura en la nube.

En la Sección 5.2 se presentan los requisitos para el enfoque MoCIP y las suposiciones del contexto de aplicación.

En la Sección 5.3 se presenta MoCIP: Un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube.

En la Sección 5.4 se presentan los metamodelos de infraestructura que soportan el enfoque MoCIP.

En la Sección 5.5 se presenta un resumen del enfoque MoCIP.

5.1 ¿Por qué un enfoque dirigido en modelos?

Los artefactos de software se están volviendo cada vez más complejos, y por lo tanto, deben discutirse en diferentes niveles de abstracción según el perfil de los stakeholders, la fase del proceso de desarrollo y los objetivos del trabajo (Brambilla *et al.*, 2017). Además, cada empresa, equipo, departamento o artefacto de software tiene sus propios requisitos en términos de privacidad, seguridad, rendimiento y alcance geográfico. En este escenario, la computación en la nube permite el aprovisionamiento de los recursos de infraestructura con el fin de apoyar el despliegue de artefactos de software. Aprovechando la computación en la nube, podemos crear y administrar recursos de infraestructura en diferentes lugares del mundo y modificar sus características de hardware bajo pedido. Sin embargo, la computación en la nube ha creado escenarios complejos en los que crear artefactos de software requiere enfoques más abstractos que la simple codificación. Para enfrentar estos desafíos, los investigadores y profesionales han empezado a utilizar técnicas dirigidas por modelos para abstraer y automatizar el proceso de desarrollo de software y el aprovisionamiento de infraestructura en la nube. En particular, la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering*, MDE) permite la automatización de la Infraestructura como Código (*Infrastructure as Code*, IaC) en el proceso de aprovisionamiento de infraestructura. A continuación, se describe cómo las técnicas dirigidas por modelos soportan el aprovisionamiento de la infraestructura en la nube.

- Los modelos son la piedra angular en MDE para comprender y compartir conocimientos sobre artefactos complejos de software. En este caso, el propósito de los modelos es comunicar en *lengua franca* los recursos de infraestructura de la nube entre los desarrolladores y el personal de operaciones.
- Los modelos permiten abstraer las características comunes de la infraestructura de diferentes proveedores de servicios en la nube. Como resultado, los modelos son capaces de generalizar elementos de infraestructura y clasificarlos en grupos coherentes, por ejemplo, de acuerdo con las capacidades de la nube, tales como cómputo, almacenamiento, redes y elasticidad.
- Los modelos pueden ser ejecutables, ya que tienen una sintaxis y una semántica, lo que significa que, a partir de un modelo, es posible generar recursos de infraestructura que se ejecutan en diferentes proveedores de servicios en la nube.
- Los modelos permiten definir la estructura de los recursos de infraestructura en la nube. Los lenguajes de modelado específicos del dominio (*Domain-Specific Modeling Language*, DSL) son lenguajes diseñados

específicamente para un dominio particular con el fin de facilitar la tarea de describir los elementos en ese dominio. En este caso, usar un lenguaje de modelado es una actividad menos propensa a errores que escribir código para definir la infraestructura de la nube.

- Los modelos permiten especificar en un nivel de abstracción más alto que los lenguajes de scripting. Un modelo de infraestructura está en un nivel de abstracción más alto y es mucho más pequeño en comparación con el mismo modelo expresado en un lenguaje de scripting. En consecuencia, cada elemento en el modelo de infraestructura representa múltiples líneas de código utilizando un lenguaje de scripting.
- Los modelos son menos sensibles al cambio entre plataformas de la nube. La arquitectura dirigida por modelos (*Model-Driven Architecture*, MDA) permite obtener un modelo independiente de plataforma (*Platform-Independent Model*, PIM) que es genérico e independientemente de cualquier proveedor de servicios en la nube. A partir de un PIM se pueden obtener modelos específicos de plataforma (*Platform-Specific Model*, PSM) que representan la infraestructura de proveedores específicos de servicios en la nube. Adicionalmente, a partir de un PSM de un proveedor de nube en particular, se puede generar un nuevo PSM para un proveedor de nube diferente.
- Los modelos permiten validar la infraestructura de la nube. Cuando se utiliza un desarrollo dirigido por modelos, es posible realizar una validación específica de dominio en tiempo de diseño. Como resultado, los recursos de infraestructura de un modelo pueden ser validados en tiempo de diseño con el fin de obtener un mensaje de error adecuado para el dominio.
- Los modelos permiten que los desarrolladores y el personal de operaciones se concentren en las tareas relacionadas con los recursos de infraestructura de la nube. MDE permite a los expertos hacer un trabajo menos repetitivo para definir los recursos de infraestructura. En consecuencia, los desarrolladores y el personal de operaciones pueden centrarse en modelar los recursos de infraestructura en lugar de escribir código repetitivo.

Las prácticas de MDE demostraron aumentar la eficiencia y la efectividad en el desarrollo de software, como lo demuestran diversos estudios cuantitativos y cualitativos (Acerbis *et al.*, 2007). MDE se centra en el dominio del problema, la abstracción y la automatización. En este escenario, MDE permite que los expertos en el dominio se centren en los requisitos de la infraestructura, requisitos abstractos del modelo y la configuración del aprovisionamiento de infraestructura en la nube. En consecuencia, MDE permite dar soporte al

enfoque IaC y contribuye a la automatización del proceso de aprovisionamiento de la infraestructura en la nube.

5.2 Requisitos para el enfoque MoCIP

Un requisito es una propiedad del tratamiento deseado por los *stakeholders*, que han comprometido sus recursos, tales como tiempo y/o dinero (Wieringa, 2014). Para un proyecto *Design Science*, un tratamiento es la interacción entre el artefacto y el contexto del problema (Wieringa, 2014). Los requisitos del tratamiento se descomponen en requisitos de artefactos y suposiciones del contexto. En este caso, los requisitos están motivados por los objetivos de investigación (Sección 2.2) y las preguntas de investigación (Sección 2.3). A continuación, se describe los requisitos del artefacto MoCIP y las suposiciones del contexto de aplicación.

- **R1.** El enfoque MoCIP debe soportar el aprovisionamiento de infraestructura para diferentes proveedores de servicios IaaS (*Infrastructure as a Service*) utilizando la Infraestructura como Código (IaC) y la Ingeniería de Software Dirigida por Modelos (MDE).
- **R2.** La herramienta ARGON debe proveer soporte al enfoque MoCIP en el aprovisionamiento de la infraestructura en la nube.
- **R3.** La herramienta ARGON debe seguir la Arquitectura Dirigida por Modelos (MDA), de tal manera que permita crear modelos de infraestructura para diferentes proveedores de la nube y generar scripts para soportar el aprovisionamiento de la infraestructura en la nube.
- **R4.** El lenguaje ArgonML debe ser un lenguaje de modelado específico del dominio (DSL) que permita *abstraer* las capacidades de la computación en la nube, tales como cómputo, almacenamiento, elasticidad y redes. ArgonML debe modelar los elementos de la infraestructura, sus atributos y configuraciones.
- **R5.** El lenguaje ArgonML debe tener una sintaxis abstracta y una sintaxis concreta. Por un lado, la sintaxis abstracta tendrá metamodelos de los proveedores de servicios IaaS. Los metamodelos deben definir todos los modelos válidos del lenguaje de modelado. Por otro lado, la sintaxis concreta define la notación gráfica de ArgonML. Los usuarios deben utilizar la notación gráfica para definir los elementos de la infraestructura.
- **R6.** La herramienta ARGON y el lenguaje ArgonML deben ser empaquetados como plugins y funcionar en el entorno de desarrollo integrado Eclipse.
- **R7.** El enfoque MoCIP debe tener facilidad de uso y ser útil para el proceso de aprovisionamiento de la infraestructura en la nube.

- **R8.** El lenguaje ArgonML debe ser efectivo y eficiente para definir la infraestructura de la nube.
- **R9.** El lenguaje ArgonML debe tener facilidad de uso y ser útil para definir la infraestructura de la nube.

5.3 MoCIP: Un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube

MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*) es un enfoque para mitigar los problemas sobre la diversidad de los lenguajes de scripting utilizados en la Infraestructura como Código (*Infrastructure as Code*, IaC) y la heterogeneidad de la infraestructura de los proveedores de servicios en la nube. El enfoque IaC propone automatizar el aprovisionamiento de infraestructura con el fin de apoyar al despliegue del software y, por lo tanto, mejorar el tiempo de entrega de un producto software.

MoCIP soporta el enfoque IaC mediante la abstracción y automatización del proceso de aprovisionamiento de infraestructura utilizando la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering*, MDE). MDE permite mitigar la heterogeneidad de los servicios IaaS (*Infrastructure as a Service*) mediante la *abstracción* de los recursos de infraestructura de los diferentes proveedores de servicios en la nube. En este caso, MDE permite *generalizar* características comunes de la infraestructura de los proveedores IaaS en un modelo. Además, MDE permite *automatizar* la generación de modelos de infraestructura y scripts de aprovisionamiento.

La Figura 5-1 presenta una visión general del enfoque MoCIP para el aprovisionamiento de los recursos de infraestructura en la nube. La primera actividad es la *Elicitación de los Requisitos* de infraestructura. El objetivo de la actividad de elicitación es descubrir y extraer las necesidades de los stakeholders y/o de la aplicación software. En este caso, se reúne información sobre la infraestructura que debe ser aprovisionada en un proveedor de servicios IaaS. El resultado de la actividad de elicitación son los requisitos de infraestructura a ser modelados y aprovisionados. Por otro lado, la actividad de *Modelado de Infraestructura* consiste en modelar los recursos de infraestructura con una Herramienta de Modelado. Para la actividad de modelado se utiliza los requisitos obtenidos en la primera actividad. A partir de un modelo de infraestructura se pueden generar scripts con las instrucciones para el aprovisionamiento de infraestructura en la nube. Los scripts son el insumo principal para la última actividad, el *Aprovisionamiento de Infraestructura*. Para la actividad de aprovisionamiento se utiliza una Herramienta de Aprovisionamiento. Esta

herramienta utiliza los scripts para realizar el aprovisionamiento de infraestructura en un proveedor de servicios en la nube.

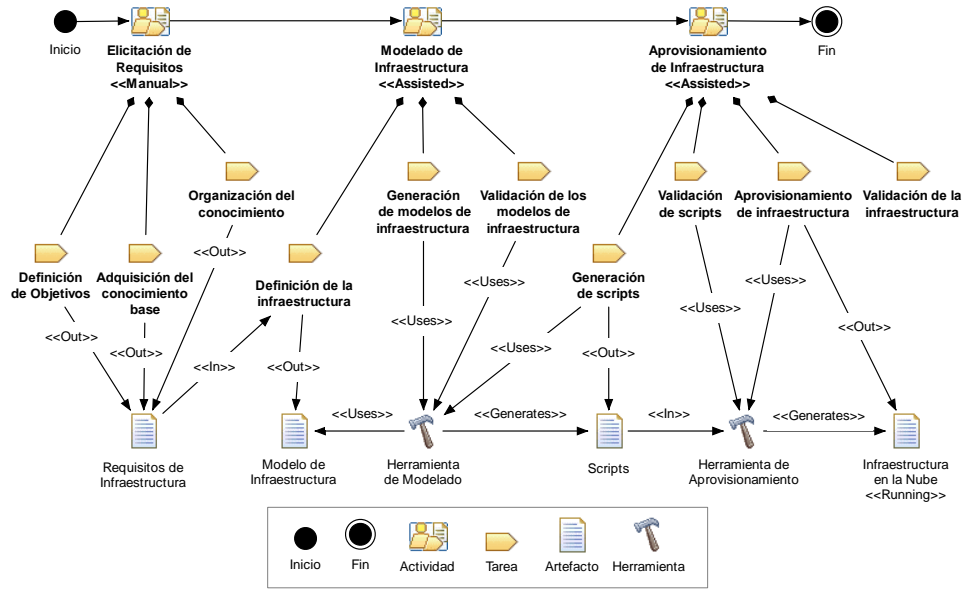


Figura 5-1 El enfoque MoCIP para el aprovisionamiento de infraestructura en la nube

5.3.1 Elicitación de requisitos

La primera actividad del enfoque MoCIP es una actividad manual, que consiste en descubrir y extraer los requisitos de infraestructura de los stakeholders y/o de una aplicación de software. Para este fin, se deben realizar las siguientes tareas que han sido adaptadas de (Kotonya *et al.*, 1998):

- **Definición de objetivos.** Se debe obtener el propósito de la infraestructura, es decir, identificar qué tipo de aplicación de software será implementada en los servidores virtuales. Por ejemplo, la aplicación de software debe estar distribuida en diferentes lugares geográficos, la aplicación de software debe distribuir la carga de trabajo entre varios servidores, la aplicación de software tiene una arquitectura multicapa, etc. Además, se debe investigar y analizar las restricciones de la aplicación de software que será implementada en la infraestructura, tales como seguridad y privacidad de los datos, localización geográfica para almacenar los datos, latencia, etc.
- **Adquisición del conocimiento base.** Se debe obtener los elementos de la infraestructura, así como sus atributos. Por ejemplo: máquinas virtuales especificando su sistema operativo, procesador virtual, memoria RAM y capacidad de almacenamiento; grupos de seguridad

detallando sus reglas para habilitar puertos desde y hacia las máquinas virtuales; etc. Además, se debe analizar si la aplicación de software — que utilizará la infraestructura— debe funcionar con otros sistemas de software existentes.

- **Organización del conocimiento.** Una vez definidos los elementos y sus atributos, el siguiente paso es definir la arquitectura de la infraestructura, es decir, determinar si será una arquitectura auto escalable, una arquitectura con múltiples capas, una arquitectura con balanceadores de carga, etc. Por otra parte, se debe establecer el proveedor de servicios en la nube donde se aprovisionará la infraestructura.

Como resultado se obtiene un documento con una lista de requisitos funcionales y no funcionales de la infraestructura. Los requisitos funcionales (RF) describen la funcionalidad que la infraestructura debe proveer. En cambio, los requisitos no funcionales (RNF) se refieren a los atributos de la infraestructura, tal como seguridad, rendimiento, escalabilidad, etc. A continuación, se describen requisitos funcionales y no funcionales:

- **RF.** La infraestructura debe ser aprovisionada en la región Norte de Virginia en Amazon Web Services.
- **RF.** Se debe utilizar una arquitectura con un balanceador de carga que distribuya la carga de trabajo entre diez máquinas virtuales.
- **RNF.** El balanceador de carga debe comprobar el estado de las máquinas virtuales a través del protocolo TCP y el puerto 80 mediante intervalos de comprobación cada 35 segundos. Para notificar una comprobación de error debe esperar al menos 7 segundos.

5.3.2 Modelado de infraestructura

El modelado de infraestructura es una actividad asistida con una herramienta de modelado para definir los recursos de la infraestructura en un modelo. Esta actividad debe apoyar las tareas de definición y configuración de los elementos de la infraestructura del enfoque IaC. A continuación, se detallan las tareas de la actividad de modelado de infraestructura.

- **Definición de la infraestructura.** Para esta tarea se debe utilizar una herramienta de modelado. En este caso, la herramienta debe soportar la definición de los elementos de la infraestructura, sus relaciones y la especificación de los atributos de cada elemento.
- **Generación de modelos de infraestructura.** Una vez definido un modelo de infraestructura para un proveedor de servicios IaaS, mediante transformaciones de modelo a modelo (*Model-to-Model*, M2M) se puede

agregar (al modelo inicial) características específicas de la infraestructura de un nuevo proveedor de servicios IaaS. En este caso, las transformaciones M2M permiten generar modelos de infraestructura para diferentes proveedores de servicios IaaS. Por un lado, la herramienta de modelado permite definir la infraestructura para un proveedor de servicios IaaS. Por otro lado, la herramienta de modelado utiliza transformaciones M2M para mover (*migrar*) la infraestructura de un proveedor de servicios IaaS a un nuevo proveedor de servicios IaaS.

- **Validación de los modelos de infraestructura.** Una vez terminada la tarea de definición de la infraestructura y, de ser el caso, la generación de modelos de infraestructura para diferentes proveedores de servicios IaaS, el siguiente paso es la validación de la sintaxis de cada modelo. En este escenario, la herramienta debe soportar la tarea de validación, y en el caso de existir un error, debe notificar un mensaje de error adecuado para el dominio.

Como resultado se obtiene un modelo con las características de la infraestructura para un proveedor de servicios IaaS. En este caso, el modelo debe presentar los elementos de infraestructura, sus relaciones y los atributos que pertenecen a cada elemento. Finalmente, para comprobar que el modelo tiene sus elementos y relaciones debidamente definidos, se realiza la validación de la sintaxis del modelo.

5.3.3 *Aprovisionamiento de infraestructura*

El aprovisionamiento de infraestructura es una actividad asistida por una herramienta de aprovisionamiento de la comunidad DevOps. El objetivo de la actividad de aprovisionamiento es orquestar la creación y/o actualización de la infraestructura junto con su configuración en un proveedor de servicios en la nube. Esta actividad brinda apoyo al enfoque IaC y está soportada por MDE. A continuación, se detallan las tareas de la actividad de aprovisionamiento de infraestructura.

- **Generación de scripts.** Esta tarea se realiza a partir de un modelo de infraestructura y con la herramienta de modelado. Se utiliza transformaciones de modelo a texto (*Model-to-Text*, M2T) para generar scripts con las instrucciones necesarias para la creación y/o actualización de la infraestructura. En este caso, se generan scripts que siguen las directrices del enfoque IaC a través de transformaciones M2T. Como resultado se obtiene un script listo para realizar el aprovisionamiento de infraestructura en la nube.

- **Validación de scripts.** A pesar de que los scripts fueron generados automáticamente a partir de un modelo de infraestructura, es necesario validar su sintaxis con el fin de comprobar que no existió ningún problema en la generación y que no existirá un error en el aprovisionamiento de infraestructura. Esta tarea es asistida por una herramienta de aprovisionamiento que será utilizada para la creación y/o actualización de infraestructura en la nube. En el caso que exista un error en la sintaxis del código generado, la herramienta debe notificar un mensaje de error adecuado para el dominio.
- **Aprovisionamiento de infraestructura.** Es una tarea asistida por una herramienta de aprovisionamiento de la comunidad DevOps. En este caso, la herramienta de aprovisionamiento debe acceder a los recursos de infraestructura del proveedor de servicios IaaS. Esta tarea ejecuta la creación o actualización de la infraestructura en la nube. En el caso de que exista algún error en la creación y/o actualización, la herramienta debe notificar un mensaje de error adecuado para el dominio.
- **Validación de la infraestructura.** Esta tarea es asistida por una herramienta con el objetivo de validar que la infraestructura y los paquetes de software se han creado e instalado correctamente. En este caso, la herramienta realizará pruebas automáticamente sobre la infraestructura aprovisionada en la nube. La automatización de la tarea de validación es muy importante en un pipeline de aprovisionamiento de infraestructura y en consecuencia en un despliegue continuo de productos software. Sin embargo, cabe mencionar se puede realizar una validación manual accediendo al panel de control del proveedor de servicios en la nube.

Como resultado se obtiene la infraestructura ejecutándose en un proveedor de servicios IaaS. Primero, a partir de un modelo de infraestructura se generan los scripts con las instrucciones necesarias para crear la infraestructura. Luego, una herramienta de aprovisionamiento utiliza los scripts generados para crear la infraestructura en el proveedor de servicios IaaS. Finalmente, para validar que la infraestructura y los paquetes de software se han creado e instalado correctamente, se puede realizar utilizar el panel de control del proveedor de servicios en la nube o utilizar una herramienta para automatizar el proceso.

5.4 Soporte de metamodelos para MoCIP

El enfoque MoCIP tiene el propósito de apoyar el enfoque IaC mediante MDE. En el contexto MDE los conceptos centrales son: modelos y transformaciones (Brambilla *et al.*, 2017). La definición de los conceptos de modelado y sus

propiedades mediante la definición de metamodelos desempeñan un papel similar a la gramática para los lenguajes textuales. Mientras que la gramática define todas las oraciones válidas de un lenguaje, un metamodelo define todos los modelos válidos de un lenguaje de modelado (Brambilla *et al.*, 2017). Las transformaciones son operaciones de manipulación de los modelos. Además, cada modelo debe estar conforme a su metamodelo. Por lo tanto, los metamodelos son una parte fundamental del enfoque MoCIP para brindar soporte al aprovisionamiento de infraestructura en la nube. A continuación, se explican los metamodelos utilizados en MoCIP. Es importante mencionar que MoCIP no está limitado a estos metamodelos, de ser el caso se puede agregar nuevos metamodelos con sus respectivas transformaciones.

5.4.1 Metamodelo abstracto de infraestructura

MoCIP utiliza MDE para abstraer y automatizar las tareas de aprovisionamiento de la infraestructura. En este caso, la Figura 5-2 muestra el metamodelo de infraestructura (Sandobalín *et al.*, 2017a) que es *abstracto* e independiente del proveedor de servicios IaaS. Por un lado, el metamodelo *abstrae* las capacidades de la computación en la nube, tales como cómputo, almacenamiento y elasticidad. Por otro lado, el metamodelo es la base (*Core*) para la definición de los metamodelos de los proveedores de servicios IaaS.

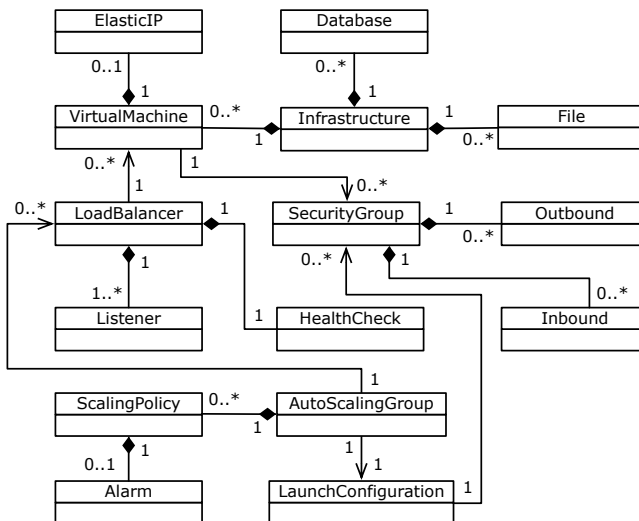


Figura 5-2 Metamodelo abstracto de infraestructura

Cada metaclassa del metamodelo de infraestructura se explica de acuerdo con las capacidades de la computación en la nube.

- La capacidad de cómputo permite modelar máquinas virtuales (*Virtual Machine*) con sus respectivos grupos de seguridad (*Security Group*). Un grupo de seguridad actúa como un corta fuegos (*Firewall*). Cada grupo de seguridad habilita conexiones desde y hacia máquinas virtuales a través de reglas de entrada (*Inbound*) y salida (*Outbound*). Una dirección IP (*Elastic IP*) puede ser asignada a una máquina virtual. Un balanceador de carga (*Load Balancer*) permite distribuir la carga de trabajo entre varias máquinas virtuales. Un oyente (*Listener*) verifica las solicitudes de conexión al balanceador de carga y la comprobación de estado (*Health Check*) valida que todas las máquinas virtuales conectadas al balanceador de carga estén disponibles.
- La capacidad de almacenamiento permite modelar elementos como bases de datos (*Database*) y servidores de archivos (*File*).
- La capacidad de elasticidad permite modelar una configuración de lanzamiento (*Launch Configuration*) en donde se especifican las características de una máquina virtual. Un grupo de auto escalado (*Auto Scaling Group*) determina el número mínimo y máximo de máquinas virtuales que se crearán. Las máquinas virtuales se crean o se eliminan en base a políticas de escalado (*Scaling Policy*) en donde se ejecuta una alarma (*Alarm*) que monitorea una métrica en un período de tiempo.

En este caso, el metamodelo *abstracto* de infraestructura representa los conceptos fundamentales de cada elemento de infraestructura de los proveedores de servicios IaaS. En consecuencia, este metamodelo *abstracto* es la base para crear (especializar) los metamodelos que representan la infraestructura para Amazon Web Services, Microsoft Azure, entre otros proveedores de servicios IaaS.

5.4.2 Metamodelo de Amazon Web Services

MoCIP soporta el aprovisionamiento de infraestructura en diferentes proveedores de servicios IaaS. En este escenario, se ha definido (a partir del metamodelo *abstracto*) el metamodelo que representa la infraestructura para Amazon Web Services (Sandobalín *et al.*, 2018).

La Figura 5-3 presenta el metamodelo de Amazon Web Services. En este caso, el metamodelo *abstracto* las capacidades de los servicios IaaS, tales como cómputo, almacenamiento y elasticidad. En consecuencia, el metamodelo presenta la metaclassa central *Service* donde se especifica la región en la que se implementará la infraestructura en Amazon Web Services. La metaclassa *Element* es abstracta y, por lo tanto, tiene atributos que serán heredados por el resto de metaclasses. La metaclassa *Tag* agrega atributos extra a las metaclasses en el estilo clave-valor. A

continuación, se explica las metACLases restantes del metamodelo de Amazon de acuerdo con las capacidades de la computación en la nube.

- En términos de capacidad de almacenamiento, se puede modelar bases de datos (*Database*) y el servicio de almacenamiento Amazon Simple Storage Service (*Bucket*).

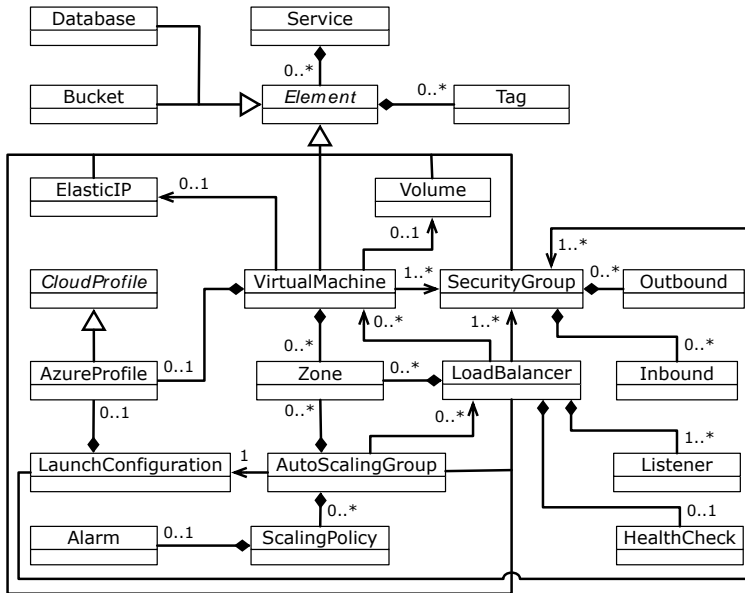


Figura 5-3 Metamodelo de Amazon Web Services.

- En términos de la capacidad de cómputo, se puede modelar máquinas virtuales (*Virtual Machine*) con sus grupos de seguridad. Un grupo de seguridad actúa como un corta fuegos (*Firewall*). Cada grupo de seguridad habilita conexiones desde y hacia máquinas virtuales a través de reglas de entrada (*Inbound*) y salida (*Outbound*). Una dirección IP (*Elastic IP*) puede ser asignada a una máquina virtual. Un volumen (*Volume*) es como un disco externo que puede ser conectado a una máquina virtual. Un balanceador de carga (*Load Balancer*) permite distribuir la carga de trabajo entre varias máquinas virtuales. Un oyente (*Listener*) verifica las solicitudes de conexión al balanceador de carga, mientras que la comprobación de estado (*Health Check*) valida que todas las máquinas virtuales conectadas al balanceador de carga estén disponibles. Una zona (*Zone*) permite la distribución de máquinas virtuales en múltiples zonas de disponibilidad de una región de Amazon Web Services. *Cloud Profile* es una metACLase abstracta que proporciona atributos correspondientes a máquinas virtuales de otros proveedores de servicios en la nube. Sin

embargo, la metaclassa de *Azure Profile* proporciona atributos específicos correspondientes a las máquinas virtuales de Microsoft Azure. En consecuencia, para proporcionar soporte a otro proveedor de la nube, se debe agregar una nueva metaclassa que herede de la metaclassa abstracta *Cloud Profile*.

- En términos de la capacidad de elasticidad, se puede modelar una configuración de lanzamiento (*Launch Configuration*) en donde se especifiquen las características de una máquina virtual. Un grupo de auto escalado (*Auto Scaling Group*) determina el número mínimo y máximo de máquinas virtuales que se crearán. Las máquinas virtuales se crean o eliminan en base a políticas de escalado (*Scaling Policy*), y una política de escalado ejecuta una alarma (*Alarm*) para supervisar una métrica en un período de tiempo.

5.4.3 Metamodelo de Microsoft Azure

MoCIP soporta el aprovisionamiento de infraestructura en diferentes proveedores de servicios IaaS. Por lo tanto, se ha definido (a partir del metamodelo *abstracto*) el metamodelo que representa la infraestructura para Microsoft Azure (Sandobalín *et al.*, 2018).

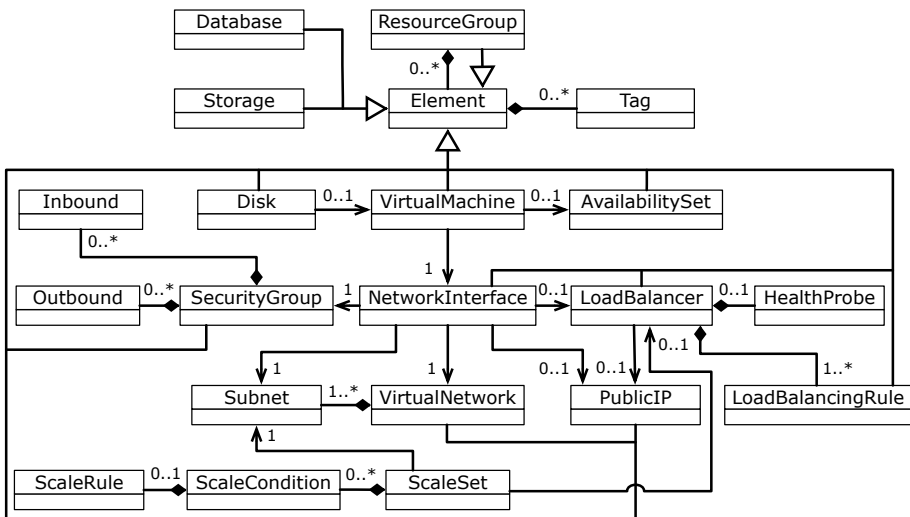


Figura 5-4 Metamodelo de Microsoft Azure

La Figura 5-4 presenta el metamodelo de Microsoft Azure. En este caso, el metamodelo *abstrae* las capacidades de los servicios IaaS, tales como cómputo, almacenamiento, elasticidad y redes. Por lo tanto, la metaclassa central *Resource Group* permite establecer la ubicación en dónde se implementará la

infraestructura en Microsoft Azure. Además, *Resource Group* es un elemento más en la infraestructura. La metaclass *Element* es una metaclass abstracta que tiene atributos que serán heredados por el resto de metaclasses. La metaclass *Tag* permite agregar atributos extra a las metaclasses en el estilo clave-valor. A continuación, se explican las metaclasses restantes del metamodelo de Azure de acuerdo con las capacidades de la computación en la nube.

- Con respecto a la capacidad de cómputo, se puede modelar máquinas virtuales (*Virtual Machine*) con sus grupos de seguridad (*Security Group*). Un grupo de seguridad actúa como un corta fuegos (*Firewall*). Cada grupo de seguridad permite conexiones desde y hacia máquinas virtuales mediante el uso de reglas de entrada (*Inbound*) y salida (*Outbound*). Se puede asignar una dirección IP pública (*Public IP*) a una máquina virtual. Un disco (*Disk*) es como un disco externo que puede ser conectado a una máquina virtual. Un balanceador de carga (*Load Balancer*) permite distribuir la carga de trabajo entre varias máquinas virtuales. Una regla de equilibrio de carga (*Load Balancing Rule*) verifica las solicitudes de conexión al balanceador de carga, mientras que una sonda de salud (*Health Probe*) valida que todas las máquinas virtuales conectadas al balanceador de carga estén disponibles. Un conjunto de disponibilidad (*Availability Set*) garantiza que todas las máquinas virtuales se distribuyan en varios nodos de hardware aislados en un clúster.
- En el caso de la capacidad de almacenamiento, se puede modelar bases de datos (*Database*) y almacenes de datos (*Storage*).
- Con respecto a la capacidad de elasticidad, se puede modelar un conjunto de escalado (*Scale Set*) en donde se especifica las características de una máquina virtual y el número mínimo y máximo de máquinas virtuales que se crearán. Las máquinas virtuales se crean o eliminan en función de una condición de escalado (*Scale Condition*) en la que se ejecuta una regla de escalado (*Scale Rule*) que supervisa una métrica en un período de tiempo.
- En el caso de la capacidad de red, se puede modelar una red virtual (*Virtual Network*) que permite que los recursos de Azure (es decir, a los elementos de infraestructura) se comuniquen entre sí en una red segura. Una red virtual puede ser segmentada en múltiples subredes (*Subnet*). Finalmente, una interfaz de red (*Network Interface*) permite que una máquina virtual interactúe con otros recursos en una red virtual.

5.5 Resumen

En este capítulo se ha presentado el enfoque MoCIP para apoyar al proceso de aprovisionamiento de infraestructura en diferentes proveedores de servicios en la nube. MoCIP promueve la automatización de la infraestructura haciendo uso del enfoque IaC. En este caso, el enfoque IaC permite la gestión de la infraestructura en diferentes proveedores de la nube haciendo uso de scripts. Sin embargo, debido a la diversidad de lenguajes de scripting (de las herramientas IaC) para definir los recursos de infraestructura junto con la heterogeneidad de servicios IaaS, la definición, actualización y ejecución de la infraestructura se ha convertido en un proceso lento y propenso a errores. Con el fin de abstraer y automatizar el proceso de aprovisionamiento, el enfoque MoCIP utiliza MDE para abstraer las capacidades de la computación en la nube y para ello utiliza (meta)modelos que permiten definir los recursos de infraestructura. Además, mediante transformaciones de modelos es posible obtener modelos de infraestructura para diferentes proveedores de servicio en la nube.

MoCIP empieza con la *Elicitación de los Requisitos* de infraestructura con el objetivo de descubrir y extraer las necesidades de los stakeholders y/o de la aplicación software. En esta actividad, se reúne información sobre la infraestructura que debe ser aprovisionada en un proveedor de servicios IaaS y se obtiene una lista de requisitos de infraestructura. Luego, se realiza la actividad de *Modelado de Infraestructura* para modelar la infraestructura con una herramienta de modelado. A partir de un modelo de infraestructura se pueden generar scripts con las instrucciones para el aprovisionamiento de infraestructura en la nube. Los scripts son el insumo principal para la última actividad, el *Aprovisionamiento de Infraestructura*. Para la actividad de aprovisionamiento se utiliza una herramienta de la comunidad DevOps para realizar el aprovisionamiento de infraestructura en la nube.

Con el fin de obtener modelos válidos para representar la infraestructura, se han definido los metamodelos correspondientes a los proveedores de servicios IaaS. En primer lugar, se propone un metamodelo *abstracto* de infraestructura que es independiente del proveedor de servicios IaaS y es la base para la definición de los metamodelos de los proveedores de servicios IaaS. En segundo lugar, a partir del metamodelo abstracto, se definen (*especializan*) los metamodelos que representan la infraestructura de Amazon Web Services y Microsoft Azure. Es importante mencionar que MoCIP no está limitado a los metamodelos mencionados, por ejemplo, a partir del metamodelo *abstracto* de infraestructura se pueden definir (*especializar*) metamodelos que representen la infraestructura de Google Computing Engine, Rackspace, entre otros proveedores de servicios IaaS.

Capítulo 6

Soporte de Herramientas

En este capítulo se presenta el soporte de herramientas para MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*). MoCIP es un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube. MoCIP soporta la Infraestructura como Código (*Infrastructure as Code*, IaC) utilizando la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering*, MDE). En este escenario, se ha desarrollado una herramienta que utiliza los principios MDE para soportar el aprovisionamiento de infraestructura en la nube.

En la Sección 6.1 se presenta ArgonML, un lenguaje específico del dominio para modelar la infraestructura de los proveedores de servicios en la nube.

En la Sección 6.2 se presenta ARGON, una herramienta de modelado de infraestructura para el aprovisionamiento en la nube.

En la Sección 6.3 se presenta un *pipeline* que utiliza un encadenamiento de herramientas DevOps junto con ARGON para automatizar el proceso de aprovisionamiento de infraestructura en la nube.

En la Sección 6.4 se presenta un resumen del soporte de herramientas para MoCIP.

6.1 El lenguaje de modelado ArgonML

Los lenguajes de modelado específicos de dominio (*Domain-Specific Modeling Language*, DSL) son lenguajes diseñados específicamente para un dominio, contexto o empresa, con el fin de ayudar a los *stakeholders* que necesitan describir cosas en ese dominio (Brambilla *et al.*, 2017). Un DSL es particularmente útil porque se adapta a los requisitos del dominio, tanto en términos de expresividad como de notación. En este caso, ArgonML (Sandobalín *et al.*, 2017a) (*ARGON Modeling Language*) es un DSL que tiene como objetivo modelar los recursos de infraestructura que serán aprovisionados en un proveedor de servicios IaaS (*Infrastructure as a Service*).

ArgonML abstrae las capacidades de la computación en la nube, tales como cómputo, almacenamiento, redes y elasticidad. En este escenario, ArgonML abstrae las características comunes de los recursos de infraestructura de los proveedores de servicios IaaS con el fin de generalizarlos en grupos coherentes, por ejemplo, agruparlos de acuerdo con las capacidades de la nube. Además, el propósito de ArgonML es mitigar la complejidad de utilizar lenguajes de scripting —utilizados en la Infraestructura como Código (*Infrastructure as Code*, IaC)— para especificar los recursos de infraestructura. El desarrollo de ArgonML está de acuerdo con los criterios MDE definidos en (Brambilla *et al.*, 2017). A continuación, se explica los ingredientes sintácticos más importantes de ArgonML, es decir, su sintaxis abstracta y su sintaxis concreta.

6.1.1 Sintaxis abstracta

Los lenguajes de modelado se desarrollan utilizando un metamodelo como técnica central (Brambilla *et al.*, 2017). El metamodelo representa la *sintaxis abstracta* del lenguaje de modelado. Debido a que ArgonML debe trabajar con Eclipse Modeling Framework (Steinberg *et al.*, 2008) se utiliza el lenguaje de metamodelado Ecore. En este contexto, el metamodelo define todos los modelos válidos del lenguaje ArgonML. Esto significa que un metamodelo de infraestructura es un conjunto de reglas para construir modelos de infraestructura. Además, el metamodelo provee un conjunto de restricciones que un modelo de infraestructura debe cumplir para estar *conforme* con su metamodelo. Con el fin de elaborar un metamodelo, se debe identificar los conceptos que ArgonML debe admitir, y entonces especificar estos conceptos de manera concisa utilizando el lenguaje de metamodelado Ecore. La Tabla 6-1 presenta los *conceptos* de modelado utilizados para definir el metamodelo de infraestructura respecto a su capacidad de cómputo.

Tabla 6-1 Conceptos de modelado de ArgonML respecto a la capacidad de cómputo

Concepto	Propiedades intrínsecas	Propiedades extrínsecas
Virtual Machine	<ul style="list-style-type: none"> ▪ Name: String ▪ Count: Integer ▪ Image: String ▪ Instance type: String 	<ul style="list-style-type: none"> ▪ Many Security Groups ▪ One Elastic IP
Security Group	<ul style="list-style-type: none"> ▪ Name: String 	<ul style="list-style-type: none"> ▪ Many Inbound Rules ▪ Manu Outbound Rules
Inbound	<ul style="list-style-type: none"> ▪ Name: String ▪ Protocol: String ▪ From port: Integer ▪ To port: Integer 	
Outbound	<ul style="list-style-type: none"> ▪ Name: String ▪ Protocol: String ▪ From port: Integer ▪ To port: Integer 	
Elastic IP	<ul style="list-style-type: none"> ▪ Name: String 	
Load Balancer	<ul style="list-style-type: none"> ▪ Name: String 	<ul style="list-style-type: none"> ▪ One Health Check ▪ Many Listeners ▪ Many Virtual Machines
Health Check	<ul style="list-style-type: none"> ▪ Name: String ▪ Ping protocol: String ▪ Ping port: Integer ▪ Interval: Integer ▪ Timeout: Integer ▪ Healthy: Integer ▪ Unhealthy: Integer 	
Listener	<ul style="list-style-type: none"> ▪ Name: String ▪ Protocol: String ▪ Instance port: Integer 	

A continuación, se resume las principales tareas del proceso de metamodelado:

- **Análisis del dominio de modelado.** Debido a que el dominio específico es la computación en la nube, el lenguaje ArgonML debe modelar la infraestructura de los servicios IaaS. En este escenario, se debe identificar los conceptos de modelado y sus propiedades. Por lo tanto, se debe identificar los elementos de infraestructura y sus atributos.
- **Diseño del lenguaje de modelado.** Se utiliza el lenguaje de metamodelado Ecore para definir el metamodelo de infraestructura. Los conceptos son transformados en metaclases, las propiedades intrínsecas en atributos y las propiedades extrínsecas son definidas como asociaciones entre las metaclases.
- **Validación del lenguaje de modelado:** La validación temprana de la sintaxis abstracta se logra mediante la instanciación manual del metamodelo de infraestructura. En este caso, se crea instancias del metamodelo en la herramienta Eclipse Modeling Framework para validar que sus metaclases, atributos y asociaciones satisfacen adecuadamente el dominio de la computación en la nube.

Como resultado del proceso de metamodelado, la Figura 6-1 presenta el metamodelo de infraestructura que abstrae las capacidades de la computación en la nube. En este caso, el metamodelo de infraestructura se corresponde con la sintaxis abstracta del lenguaje ArgonML

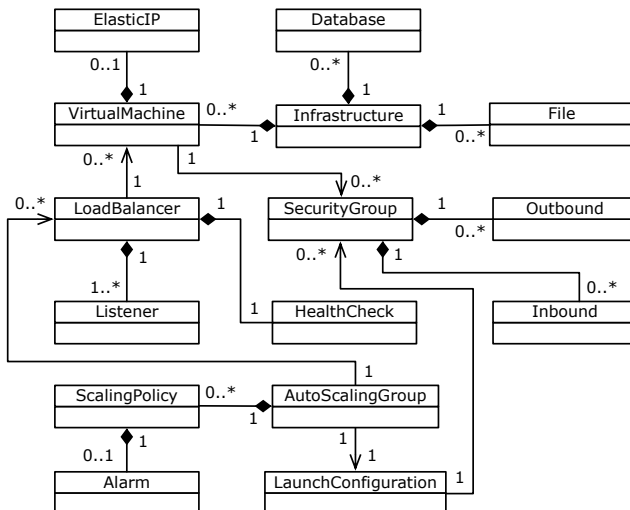










Figura 6-1 Sintaxis abstracta del lenguaje ArgonML

6.1.2 Sintaxis concreta

El lenguaje ArgonML define una notación visual para modelar los elementos de infraestructura. Con el fin de utilizar elementos gráficos para representar los recursos y propiedades que describan la infraestructura se hace uso de un editor de modelos. En este caso, el editor de modelos es Graphical Modeling Framework (Eclipse Foundation, 2006b) (GMF). El lenguaje ArgonML utiliza una sintaxis concreta gráfica para definir los recursos de infraestructura de la nube.

Existen varias herramientas que proporcionan lenguajes específicos para describir formalmente la sintaxis concreta de un lenguaje de modelado. En este caso, para definir la sintaxis concreta del lenguaje ArgonML se utiliza la herramienta Eugenia (Kolovos *et al.*, 2015) con su lenguaje específico Emfatic (Eclipse Foundation, 2012). Eugenia permite realizar un mapeo entre los conceptos de modelado descritos en el metamodelo de infraestructura (sintaxis abstracta) de la Figura 6-1 y sus representaciones visuales o elementos gráficos.

Tabla 6-2 Conceptos de modelado respecto a la capacidad de cómputo y sus elementos gráficos



Elemento Gráfico	Concepto de Modelado
	Virtual Machine
	Security Group
	Inbound
	Outbound
	Elastic IP
	Load Balancer
	Listener
	Health Check

La Tabla 6-2 presenta un mapeo entre los conceptos de modelado de la sintaxis abstracta respecto a la capacidad de cómputo y sus elementos gráficos. En este caso, los conceptos de modelado representan a los elementos del metamodelo de infraestructura (sintaxis abstracta) que se presenta en la Figura 6-1. Por

ejemplo, se muestra la representación visual de elementos, tales como máquina virtual (*Virtual Machine*), grupo de seguridad (*Security Group*), entre otros.





La Tabla 6-3 presenta un mapeo entre los conceptos de modelado de la sintaxis abstracta respecto a la capacidad de almacenamiento y sus elementos gráficos. Por ejemplo, se muestra la representación visual de los elementos base de datos (*Database*) y servidor de archivos (*File*).

Tabla 6-3 Conceptos de modelado respecto a la capacidad de almacenamiento y sus elementos gráficos

Elemento Gráfico	Concepto de Modelado
	File
	Database

La Tabla 6-4 presenta un mapeo entre los conceptos de modelado de la sintaxis abstracta respecto a la capacidad de elasticidad y sus elementos gráficos. Por ejemplo, se muestra la representación visual de elementos, tales como alarma (*Alarm*), configuración de lanzamiento (*Launch Configuration*), entre otros.

Tabla 6-4 Conceptos de modelado respecto a la capacidad de elasticidad y sus elementos gráficos

Elemento Gráfico	Concepto de Modelado
	Launch Configuration
	Auto Scaling Group
	Scaling Policy
	Alarm

Por otro lado, el lenguaje Emfatic permite representar el metamodelo de infraestructura (sintaxis abstracta) de una manera textual, con el fin de agregar anotaciones para crear un editor de modelos completamente funcional que trabaje en GMF. Es importante mencionar que GMF proporciona un enfoque dirigido por modelos para generar editores gráficos en el entorno de desarrollo integrado Eclipse. La Figura 6-2 presenta un extracto del archivo Emfatic donde se especifica la sintaxis concreta del lenguaje de modelado ArgonML.


```

Infrastructure.emf
1 @gmf
2 @namespace(uri="platform:/resource/edu.cloud.dsl/model/Infrastructure.ecore",
3           prefix="Infrastructure")
4 package Infrastructure;
5
6 @gmf.diagram
7 class Infrastructure { }
8
9 @gmf.node(label="name", tool.name="Virtual Machine")
10 class VirtualMachine {
11   @gmf.link
12   ref SecurityGroup group;
13   @gmf.link
14   val ElasticIP elastic_id;
15   attr String image;
16   attr String instance_type;
17   attr int count;
18 }
19
20 @gmf.node(label="name", tool.name="Security Group")
21 class SecurityGroup {
22   @gmf.link
23   val Outbound rules_egress;
24   @gmf.link
25   val Inbound rules;
26   attr String description;
27 }

```

Figura 6-2 Extracto de la definición de la sintaxis concreta de lenguaje ArgonML

A continuación, se explican los elementos de la sintaxis concreta definida con el lenguaje Emfatic y sus anotaciones.

- **Símbolos gráficos.** La anotación `@gmf.note` representa un elemento gráfico de un modelo de infraestructura. La línea 9 representa una máquina virtual (*Virtual Machine*), mientras que la línea 20 representa un grupo de seguridad (*Security Group*).
- **Reglas de composición.** La anotación `@gmf.link` representa una asociación entre dos elementos de un modelo de infraestructura. La línea 11 representa que un elemento máquina virtual (*Virtual Machine*) tiene una asociación con un elemento grupo de seguridad (*Security Group*). En cambio, la línea 24 representa que un elemento grupo de seguridad (*Security Group*) tiene una asociación con un elemento regla de entrada (*Inbound*).
- **Mapeo.** La herramienta Eugenia permite hacer un mapeo entre los conceptos de modelado de la sintaxis abstracta (Figura 6-1) y su representación gráfica en la sintaxis concreta (Figura 6-3). Por ejemplo, la línea 9 representa el mapeo entre el elemento gráfico de una máquina virtual (*Virtual Machine*) y su concepto de modelado (metaclase) en la sintaxis abstracta (metamodelo de infraestructura).

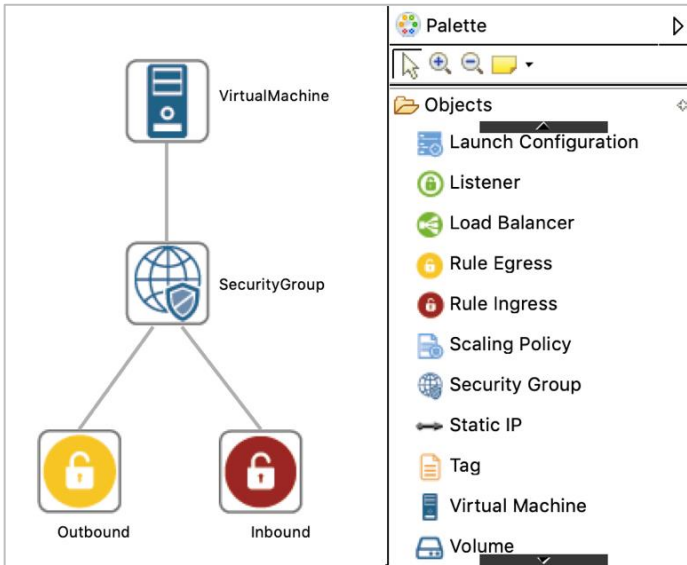


Figura 6-3 Sintaxis concreta del lenguaje ArgonML

Por otro lado, la anotación `@gmf.diagram` representa el diagrama que contendrá a todos los elementos de la infraestructura. Además, la herramienta Eugenia permite generar los modelos necesarios para visualizar la notación gráfica del lenguaje ArgonML con GMF. Por lo tanto, se genera el modelo `gmfgraph` para visualizar cada elemento gráfico, el modelo `gmftool` para generar una paleta con los elementos de la infraestructura, y el modelo `gmfmap` define el mapeo entre los elementos del metamodelo (sintaxis abstracta) y los elementos gráficos (sintaxis concreta) definidos en el modelo `gmfgraph`.

La Figura 6-3 muestra la sintaxis concreta del lenguaje de modelado ArgonML. La sintaxis concreta permite definir mediante un diagrama los elementos de la infraestructura. El diagrama presenta una máquina virtual (*Virtual Machine*) conectada con un grupo de seguridad (*Security Group*). El grupo de seguridad permite las conexiones entrantes hacia la máquina virtual a través de una regla de entrada (*Inbound*). Además, el grupo de seguridad habilita las conexiones salientes desde la máquina virtual con una regla de salida (*Outbound*).

6.2 La herramienta ARGON

Los modelos no son entidades aisladas ni estáticas. Como parte de un proceso MDE, los modelos se traducen a otros lenguajes o representaciones, por ejemplo, a partir de un modelo se puede generar código (Brambilla *et al.*, 2017). En este contexto, el lenguaje ArgonML debe combinarse con herramientas de

soporte para ayudar a los expertos del dominio a maximizar su productividad al trabajar en la definición de la infraestructura de la nube. En consecuencia, el lenguaje ArgonML apoya a la herramienta ARGON (Sandobalín *et al.*, 2017a) (*An infrastruRcture modelinG tool for clOud provisioniNg*) en el aprovisionamiento de infraestructura en la nube.

ARGON realiza operaciones con modelos implementando transformaciones de modelo a modelo (*Model to Model*, M2M) y de modelo a texto (*Model to Text*, M2T). En el caso de las transformaciones M2M, los parámetros de entrada y salida de la transformación son modelos. ARGON utiliza transformaciones M2M para generar los modelos de infraestructura para diferentes proveedores de servicios IaaS. En cambio, en las transformaciones M2T, la entrada es un modelo y la salida es una cadena de texto. ARGON utiliza transformaciones M2T para generar los scripts con las instrucciones necesarias para realizar el aprovisionamiento de infraestructura en la nube.

6.2.1 La arquitectura de ARGON

Los modelos deben construirse de conformidad con un conjunto de metamodelos, lo que facilita la transformación entre modelos y su automatización a través de herramientas (Brambilla *et al.*, 2017). En este escenario, ARGON utiliza la Arquitectura Dirigida por Modelos (*Model-Driven Architecture*, MDA) para definir metamodelos en diferentes niveles de abstracción y facilitar las transformaciones entre modelos de infraestructura con el fin de automatizar las actividades del enfoque MoCIP.

A parte de la definición de los niveles de abstracción, otro de los componentes principales de MDA son los lenguajes de modelado utilizados en cada nivel (Brambilla *et al.*, 2017). En este escenario, la herramienta ARGON utiliza el lenguaje ArgonML para realizar un modelado de alto nivel de las capacidades de la nube, tales como cómputo, almacenamiento, redes y la elasticidad. En este escenario, primero se *abstrae* las capacidades de la nube en un metamodelo *abstracto* de infraestructura, que es independiente de cualquier proveedor de servicios en la nube. El metamodelo abstracto sirve como base para definir (*especializar*) los metamodelos que representen la infraestructura para cada proveedor de servicios IaaS, tales como Amazon Web Services, Microsoft Azure, entre otros. La Figura 6-4 presenta la arquitectura de la herramienta ARGON de acuerdo con los niveles de abstracción de MDA.

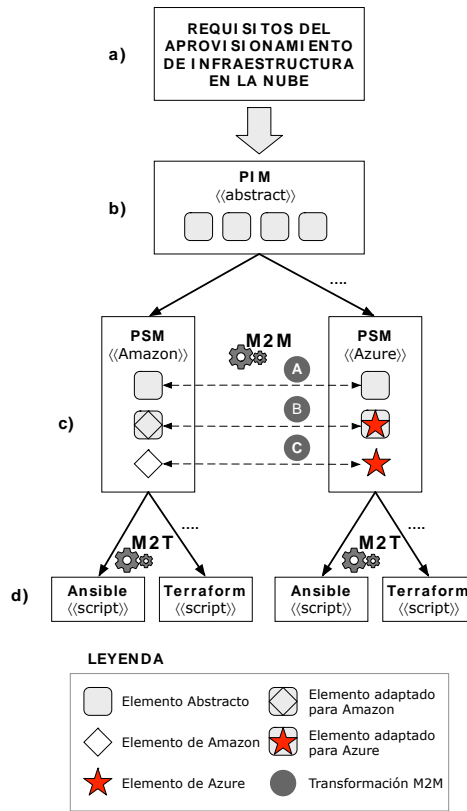


Figura 6-4 Arquitectura de la herramienta ARGON

6.2.1.1 Requerimientos

Es la capa más abstracta que representa el contexto, los requisitos y el propósito de la infraestructura de la nube, sin ningún vínculo con los proveedores de servicio IaaS. La Figura 6-4a muestra la capa en la que se capturan los criterios para obtener un aprovisionamiento de infraestructura en la nube.

6.2.1.2 Modelo independiente de la plataforma (PIM)

En esta capa, se define un metamodelo *abstracto* de infraestructura, que abstrae las capacidades de la nube, tales como cómputo, almacenamiento y elasticidad. El metamodelo *abstracto* de infraestructura se explica en la Sección 5.4.1. La Figura 6-4b representa un modelo de infraestructura *abstracto* e independiente de la plataforma (*Platform-Independent Model*, PIM). Sin embargo, en la práctica, no se modela un PIM de infraestructura abstracto, se modela un modelo de infraestructura específico de la plataforma (*Platform-Specific Model*, PSM). Existen dos razones pragmáticas para modelar directamente los modelos PSM.

1. Los proveedores de servicios IaaS tienen conceptos similares respecto a las capacidades de cómputo, almacenamiento, elasticidad y redes.
2. Los desarrolladores y personal de operaciones conocen por adelantado cual es el proveedor de servicios IaaS a utilizar.

En este caso, por motivos pragmáticos, al no haber mucha diferencia semántica entre los distintos proveedores de servicios IaaS, la construcción de un PIM que posteriormente debe ser transformado a un PSM sería un esfuerzo mayor e innecesario. Por lo tanto, se considera más apropiado definir la infraestructura específica y luego realizar transformaciones M2M para obtener la infraestructura para diferentes proveedores de servicios IaaS. En consecuencia, el metamodelo definido en esta capa es la base para definir —en la siguiente capa— los metamodelos para los diferentes proveedores de servicios IaaS, tales como Amazon Web Services, Microsoft Azure, Google Computing Engine, etc.

La Figura 6-4b muestra cuadrados que representan elementos abstractos de infraestructura que son independientes de cualquier proveedor de servicios IaaS. Por ejemplo, elementos de infraestructura, tales como grupo de seguridad (*Security Group*), máquina virtual (*Virtual Machine*), etiqueta (*Tag*), etc., que luego se (re)definen como parte de los diferentes modelos PSM de infraestructura.

6.2.1.3 Modelo específico de la plataforma (PSM)

En esta capa, se define los recursos de infraestructura para cada proveedor de servicios IaaS. Se utiliza el metamodelo *abstracto* de infraestructura de la capa PIM para definir (*especializar*) los metamodelos para proveedores de servicios IaaS. En este caso, la Figura 6-4c presenta los modelos PSM que representan la infraestructura de Amazon Web Services y Microsoft Azure. El metamodelo de Amazon Web Services se explica en la sección 5.4.2, mientras que el metamodelo de Microsoft Azure se explica en la Sección 5.4.3.

Los metamodelos PSM tienen algunos elementos abstractos (cuadrados) que se utilizan sin ninguna modificación (desde el metamodelo PIM), por ejemplo, grupo de seguridad (*Security Group*) y etiqueta (*Tag*). Sin embargo, en la capa PSM existen elementos con atributos que dependen de la plataforma de nube de destino, tales como máquinas virtuales (*Virtual Machine*) y equilibradores de carga (*Load Balancer*). Por ejemplo, la Figura 6-4c muestra un elemento adaptado para Amazon Web Services (diamante inscrito en el cuadrado) y un elemento adaptado para Microsoft Azure (estrella inscrita en el cuadrado).

En la capa PSM, se realizan transformaciones M2M para obtener modelos de infraestructura para proveedores específicos de servicios IaaS. A continuación, se explica los tres casos de transformaciones M2M:

- **Caso de transformación A.** Es un caso particular, un elemento del modelo PSM origen se copia en el modelo PSM destino sin ninguna modificación. Por ejemplo, un elemento grupo de seguridad (*Security Group*) de Amazon se copia con todos sus atributos en el modelo de destino de Azure.
- **Caso de transformación B.** En este caso, un elemento del modelo PSM de origen se copia con algunos de sus atributos en el modelo PSM de destino. Dado que algunos atributos son diferentes en cada plataforma en la nube, estos atributos diferentes del modelo PSM origen se descartan y se agregan nuevos atributos al elemento del modelo PSM destino. Por ejemplo, un elemento máquina virtual (*Virtual Machine*) del modelo de Amazon se copia con la mayoría de sus atributos en el modelo de Azure. Sin embargo, el sistema operativo para Amazon Web Services se especifica con el atributo imagen (*Image*), mientras que para Microsoft Azure se utilizan los atributos oferta (*Offer*), publicador (*Publisher*) y SKU. Se considera estos cambios en la transformación M2M de un modelo PSM de Amazon hacia un modelo PSM de Azure.
- **Caso de transformación C.** En este caso, un elemento del modelo PSM de origen coincide con un elemento similar en el modelo PSM de destino. Sin embargo, esta coincidencia es solo conceptual debido a que están representados de manera diferente. Por ejemplo, un elemento zona de disponibilidad (*Availability Zone*) de Amazon Web Services es similar al elemento conjunto de disponibilidad (*Availability Set*) de Microsoft Azure. Una zona de disponibilidad es una ubicación aislada en una región particular de Amazon Web Services, mientras que un conjunto de disponibilidad es un nodo de hardware aislado en un clúster de Microsoft Azure.

El lenguaje para realizar las transformaciones M2M es ATL (Jouault *et al.*, 2008). Para realizar las transformaciones M2M, se definen módulos de transformación M2M, por ejemplo, *Amazon2Azure.atl*. Estos módulos contienen reglas de mapeo que representan la equivalencia entre los conceptos de los (meta)modelos. En este caso, el mapeo entre el modelo PSM de Amazon y el modelo PSM de Azure. Es importante mencionar que la Figura 6-4c representa el modelo PSM de Amazon y el modelo PSM de Azure. Sin embargo, ARGON no se limita a estos modelos PSM. ARGON permite agregar nuevos (meta)modelos de infraestructura para otros proveedores de servicios IaaS. En este caso, se debe definir el metamodelo de infraestructura del nuevo proveedor de servicios IaaS junto con los mapeos entre (meta)modelos para realizar las transformaciones M2M. Por ejemplo, definir el metamodelo que represente la infraestructura de Google Computing Engine y los mapeos para realizar las

transformaciones M2M desde un modelo de Amazon hacia un modelo de Google.

6.2.1.4 *Instancias*

En esta capa, se generan los scripts para herramientas de aprovisionamiento, tales como Ansible (DeHaan, 2012), CloudFormation (Amazon Web Services, 2011), Terraform (HashiCorp, 2017), entre otras. En este caso, para generar scripts se utilizan transformaciones M2T. La entrada para una transformación M2T es un modelo PSM de infraestructura (por ejemplo, modelo de Amazon, modelo de Azure, modelo de Google, etc.) y la salida es un script para realizar el aprovisionamiento de infraestructura.

La Figura 6-4d representa el proceso de transformación M2T. En este caso, se realizó la abstracción de las características de cada lenguaje de scripting (de las herramientas de aprovisionamiento) para definir reglas de transformación en plantillas. Estas plantillas se definen para la herramienta Acceleo (Eclipse Foundation, 2006a). Finalmente, los scripts se utilizan en el aprovisionamiento de infraestructura en un proveedor de servicios IaaS.

6.2.2 *Generación de modelos de infraestructura*

Las transformaciones M2M toman uno o más modelos de entrada para producir uno o más modelos de salida. En el caso de la herramienta ARGON, las transformaciones M2M son uno a uno, es decir, existe un modelo de entrada y un modelo de salida. ARGON utiliza el lenguaje de transformación ATLAS (*ATLAS Transformation Language*, ATL) para realizar transformaciones M2M. ATL es uno de los lenguajes de transformación M2M más utilizados, tanto en la academia como en la industria, y existe un soporte maduro de herramientas disponible (Brambilla *et al.*, 2017). De acuerdo con la nomenclatura de ATL, para un modelo de entrada se usa el término *modelo de origen*, mientras que para un modelo de salida se utiliza el término *modelo de destino*.

La herramienta ARGON utiliza transformaciones M2M para generar modelos de infraestructura para diferentes proveedores de servicios IaaS. En este caso, a partir de un *modelo origen* de infraestructura se puede generar diferentes *modelos destino* de infraestructura para proveedores de servicios IaaS.

Los conceptos de cada elemento de infraestructura deben organizarse en torno a los metamodelos y a las transformaciones M2M asociadas que implementan los mapeos (*Mappings*) entre los metamodelos. En este caso, un *mapeo* consiste en la definición de las correspondencias entre los conceptos (de los elementos) de los metamodelos, con el fin de automatizar las transformaciones M2M entre los respectivos modelos de infraestructura.

A continuación, se explica las transformaciones M2M para obtener un modelo de Azure a partir de un modelo de Amazon. Es importante mencionar que, con el fin de demostrar la viabilidad de la propuesta, se utiliza los metamodelos de Amazon Web Services y Microsoft Azure. Sin embargo, ARGON puede agregar modelos de infraestructura para diferentes proveedores de la nube.

6.2.2.1 Transformaciones M2M para un Máquina Virtual

La Figura 6-5 presenta las transformaciones M2M para obtener una máquina virtual para Microsoft Azure a partir de una máquina virtual de Amazon Web Services.

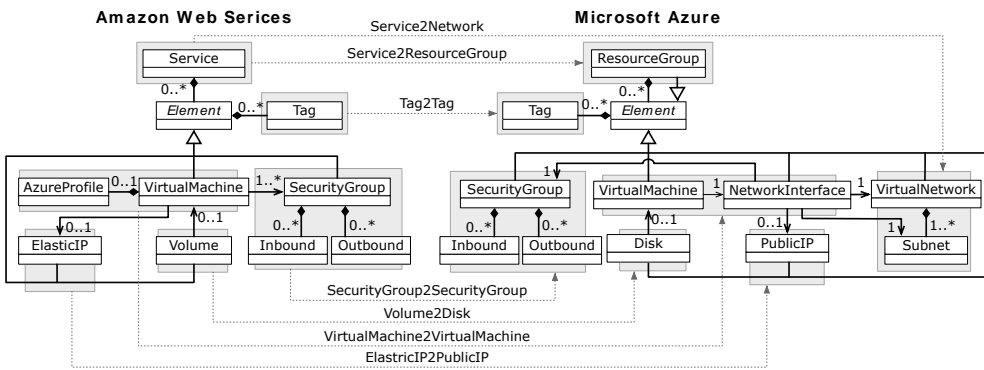


Figura 6-5 Transformaciones M2M para una máquina virtual

En este caso, una máquina virtual es elemento central de las transformaciones M2M. A continuación, se explican las transformaciones M2M:

- **Service2Network.** La metaclass *Service* representa un modelo de Amazon. En este caso, por cada modelo de Amazon se crea una red virtual (*Virtual Network*) con su respectiva subred (*Subnet*) en el modelo de Azure.
- **Service2ResourceGroup.** La metaclass *Service* representa un modelo de Amazon. En cambio, la metaclass *Resource Group* representa un modelo de Azure. Por lo tanto, por cada modelo de Amazon se crea un modelo de Azure.
- **Tag2Tag.** La metaclass *Tag* agrega atributos extra a los elementos de infraestructura en el estilo clave-valor. Por lo tanto, por cada *Tag* encontrado en un elemento del modelo de Amazon, se crea el mismo *Tag* en el elemento correspondiente del modelo de Azure.
- **SecurityGroup2SecurityGroup.** Esta transformación especifica que por cada grupo de seguridad (*Security Group*) con sus reglas de entrada (*Inbound*) y sus reglas de salida (*Outbound*) en el modelo de Amazon, se

crea el grupo de seguridad (*Security Group*) correspondiente con sus reglas de entrada (*Inbound*) y reglas de salida (*Outbound*) en el modelo de Azure.

- **Volume2Disk.** Esta transformación específica que por cada elemento volumen (*Volume*) en el modelo de Amazon, se debe crear un elemento disco (*Disk*) en el modelo de Azure. Los elementos *Volume* y *Disk* representan discos externos que pueden ser conectados a una máquina virtual (*Virtual Machine*).
- **VirtualMachine2VirtualMachine.** Esta transformación específica que por cada máquina virtual (*Virtual Machine*) junto con un perfil de Azure (*Azure Profile*) en el modelo de Amazon, se crea la máquina virtual (*Virtual Machine*) correspondiente con una interfaz de red (*Network Interface*) en el modelo de Azure. En este caso, el perfil de Azure permite agregar atributos que necesita una máquina virtual para funcionar en Microsoft Azure
- **ElasticIP2PublicIP.** Esta transformación específica que por cada dirección elástica IP (*Elastic IP*) asignada a una máquina virtual en el modelo de Amazon, se crea la dirección pública IP (*Public IP*) correspondiente a la máquina virtual en el modelo de Azure.

6.2.2.2 Transformaciones M2M para un Balanceador de Carga

La Figura 6-6 presenta las transformaciones M2M para obtener un balanceador de carga para Microsoft Azure a partir de un balanceador de carga de Amazon Web Services. Un balanceador de carga es un elemento que distribuye la carga de trabajo entre varias máquinas virtuales. En este caso, un balanceador de carga es el elemento central de las transformaciones M2M.

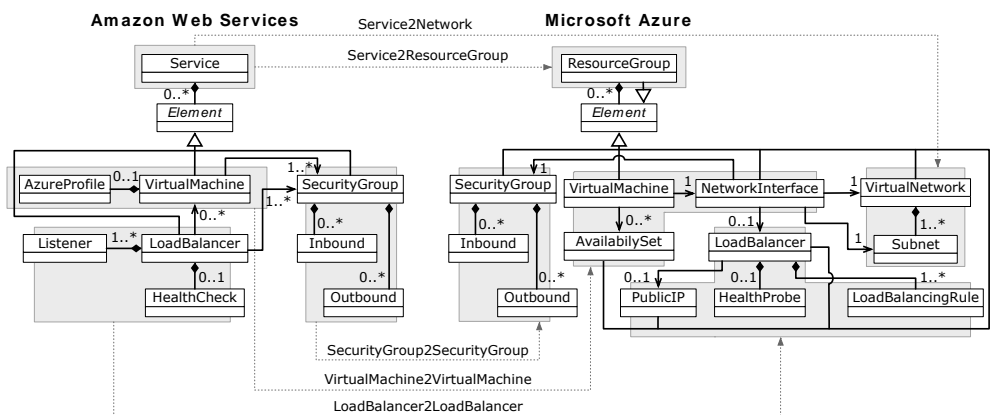


Figura 6-6 Transformaciones M2M para un balanceador de carga

A continuación, se explican las principales transformaciones M2M para un balanceador de carga. Es importante mencionar que se utilizan las transformaciones M2M para una máquina virtual que se explicaron previamente en la Sección 6.2.2.1, tales como *Service2Network*, *Service2ResourceGroup*, *SecurityGroup2SecurityGroup* y *Tag2Tag*.

- **VirtualMachine2VirtualMachine.** El metamodelo de Amazon tiene una máquina virtual (*Virtual Machine*) junto con un perfil de Azure (*Azure Profile*). El perfil de Azure permite agregar atributos que necesita una máquina virtual para funcionar en Microsoft Azure. En cambio, el metamodelo de Azure tiene una máquina virtual (*Virtual Machine*) junto su interfaz de red (*Network Interface*) en un conjunto de disponibilidad (*Availability Set*). Un conjunto de disponibilidad garantiza que todas las máquinas virtuales se distribuyan en varios nodos de hardware aislados en un clúster. Por lo tanto, por cada máquina virtual (*Virtual Machine*) junto con su perfil de Azure (*Azure Profile*) en el modelo de Amazon, se crea la máquina virtual (*Virtual Machine*) correspondiente junto con su interfaz de red (*Network Interface*) y un conjunto de disponibilidad (*Availability Set*) en el modelo de Azure.
- **LoadBalance2LoadBalancer.** El metamodelo de Amazon tiene un balanceador de carga (*Load Balancer*) con un oyente (*Listener*) que verifica todas las solicitudes de conexión, mientras que la comprobación de estado (*Health Check*) valida que todas las máquinas virtuales conectadas estén disponibles. En cambio, el metamodelo de Azure tiene un balanceador de carga (*Load Balancer*) con una regla de equilibrio de carga (*Load Balancing Rule*) que verifica todas las solicitudes de conexión, mientras que una sonda de salud (*Health Probe*) valida que todas las máquinas virtuales conectadas estén disponibles. Además, el balanceador de carga (en Microsoft Azure) tiene una dirección pública IP (*Public IP*). Por lo tanto, por cada balanceador de carga (*Load Balancer*) junto con su oyente (*Listener*) y su comprobación de estado (*Health Check*) en el modelo de Amazon, se crea el balanceador de carga (*Load Balancer*) correspondiente junto con su regla de equilibrio de carga (*Load Balancing Rule*), sonda de salud (*Health Probe*) y dirección pública IP (*Public IP*) en el modelo de Azure.

6.2.2.3 Transformaciones M2M para un escalado automático

La Figura 6-7 presenta las transformaciones M2M para una configuración de escalado automático. En un escalado automático, las máquinas virtuales se crean o destruyen de acuerdo con políticas de escalado. En este caso, se debe agregar

un balanceador de carga que distribuya la carga de trabajo entre todas máquinas virtuales.

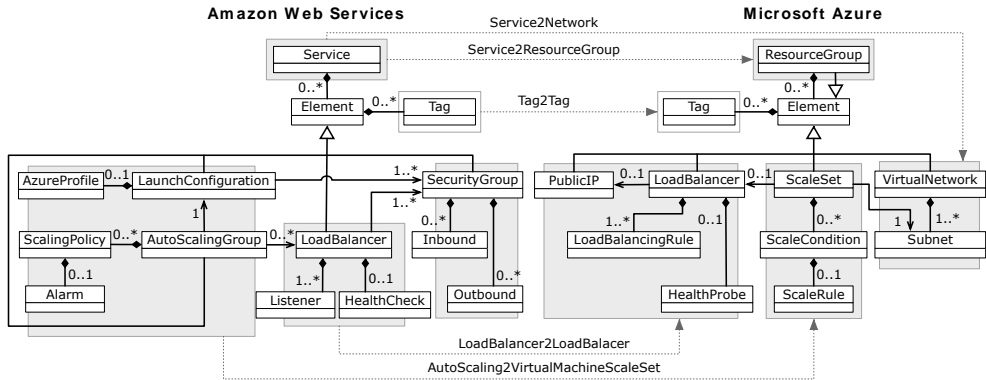


Figura 6-7 Transformaciones M2M para un escalado automático

A continuación, se explican las transformaciones M2M para una configuración de escalado automático. Se utilizan las transformaciones M2M para una máquina virtual que se explicaron en la Sección 6.2.2.1, tales como *Service2Network*, *Service2ResourceGroup*, *Tag2Tag* y *SecurityGroup2SecurityGroup*. Además, se utiliza la transformación M2M para un balanceador de carga (*LoadBalance2LoadBalancer*) que se explicó en la Sección 6.2.2.2.

- AutoScaling2VirtualMachineScaleSet.** El metamodelo de Amazon tiene una configuración de lanzamiento (*Launch Configuration*) en donde se definen las características de máquinas virtuales y un perfil de Azure (*Azure Profile*). El perfil de Azure permite agregar atributos que necesita una máquina virtual para funcionar en Microsoft. Además, un grupo de escalado automático (*Auto Scaling Group*) define el número mínimo y máximo de máquinas virtuales que formarán parte de la configuración del escalado automático. Las máquinas virtuales se crean o se destruyen utilizando políticas de escalado (*Scaling Policy*) con una alarma (*Alarm*) que monitoriza una métrica en un período de tiempo. En cambio, el metamodelo de Azure tiene un conjunto de escalado (*Scale Set*) en donde se especifican tanto las características de las máquinas virtuales como el número mínimo y máximo de máquinas virtuales que se crearán. Las máquinas virtuales (en Microsoft Azure) se crean o eliminan en función de una condición de escalado (*Scale Condition*) en donde se ejecuta una regla de escalado (*Scale Rule*) que supervisa una métrica en un período de tiempo. Por lo tanto, por cada configuración de lanzamiento (*Launch Configuration*) junto con su perfil de Azure (*Azure Profile*), grupo de escalado automático (*Auto Scaling Group*), política de escalado (*Scaling*

Policy) y alarma (*Alarm*) en el modelo de Amazon, se crea el conjunto de escalado (*Scale Set*) correspondiente junto con su condición de escalado (*Scale Condition*) y regla de escalado (*Scale Rule*) en el modelo de Azure.

6.2.2.4 Módulo de transformaciones M2M

ARGON utiliza el lenguaje de transformación ATL para realizar las transformaciones M2M. En este caso, la Figura 6-8 presenta un extracto del módulo *amazon2azure.atl* donde se definen las transformaciones M2M.

El modelo origen es un *modelo de Amazon*, mientras que el modelo destino es un *modelo de Azure*. El módulo *amazon2azure.atl* está dividido por un encabezado y la sección del cuerpo. En la sección del encabezado se establece el nombre del módulo de transformación (línea 5) y se declara el metamodelo origen (línea 1) y el metamodelo destino (líneas 2). En cambio, la sección del cuerpo (para transformaciones ATL) está compuesto por un conjunto de reglas (*Rules*) y ayudantes (*Helpers*) que se establecen en un orden arbitrario después del encabezado. Por ejemplo, la regla *ResourceGroup* (líneas 11-25) describe como el modelo de Azure se genera con elementos del modelo de Amazon. En cambio, el ayudante *rootResource* (líneas 8-9) es una función auxiliar que permite inicializar el elemento *ResourceGroup* del metamodelo de Azure desde diferentes puntos de la transformación M2M.

```
amazon2azure.atl
1  -- @path MMAAmazon=/edu.cloud.m2m/metamodels/Amazon.ecore
2  -- @path MMAzure=/edu.cloud.m2m/metamodels/Azure.ecore
3  -- @atlcompiler atl2010
4
5  module amazon2azure;
6  create OUT: MMAzure from IN: MMAAmazon;
7
8  helper def: rootResource: MMAzure!ResourceGroup =
9      OclUndefined;
10
11 rule ResourceGroup {
12     from
13         a: MMAAmazon!Infrastructure
14     to
15         zrgroup: MMAzure!ResourceGroup (
16             file_name <- a.file_name,
17             location <- a.region,
18             name <- a.file_name,
19             elements <- a.elements
20         )
21     do{
22         thisModule.rootResource<-zrgroup;
23         thisModule.rootResource.elements<-thisModule.VirtualNetwork(OclUndefined);
24     }
25 }
```

Figura 6-8 Extracto del módulo de transformaciones M2M *amazon2azure.atl*

Es importante mencionar que la Figura 6-8 muestra solo un extracto del módulo *amazon2azure.atl* con el fin de explicar las partes principales de una transformación ATL. Adicionalmente, se puede definir diferentes módulos de transformación M2M. Por ejemplo, el módulo *azure2google.atl* tiene el *modelo de Azure* como el modelo origen y el *modelo de Google* como el modelo destino.

Para definir un módulo de transformación M2M para la herramienta ARGON se deben cumplir dos requisitos fundamentales: (1) definir un metamodelo origen y un metamodelo destino, y (2) definir los mapeos entre los elementos de los metamodelos.

6.2.3 Generación de scripts de aprovisionamiento

Cuando se emplea el lenguaje de modelado ArgonML para construir los modelos de infraestructura, no todos los detalles de las tecnologías subyacentes pueden ser expresados, debido a que el objetivo de MDE es abstraer los detalles técnicos. Por lo tanto, ARGON utiliza transformaciones M2T para generar scripts —a partir de modelos de infraestructura— con la información técnica tanto de la herramienta de aprovisionamiento como del proveedor de servicios IaaS. En este contexto, la generación de scripts de aprovisionamiento puede describirse como la transición vertical desde modelos de infraestructura en un nivel superior de abstracción hacia artefactos de nivel inferior.

Las transformaciones M2T leen los metamodelos de infraestructura que están basados en Ecore y generan los extractos o partes del script por cada elemento del modelo de infraestructura. Los lenguajes de transformación M2T separan el código estático y dinámico mediante el uso de un enfoque basado en *plantillas* para transformaciones M2T. Una plantilla puede verse como un tipo de plano que define elementos de texto estáticos compartidos por todos los artefactos, así como partes dinámicas que deben llenarse con información específica de cada caso en particular (Brambilla *et al.*, 2017).

ARGON utiliza Acceleo (Eclipse Foundation, 2006a) para implementar las transformaciones M2T. El objetivo de Acceleo es proporcionar una versión pragmática del estándar de transformación M2T del OMG (*Object Management Group*) para modelos basados en Eclipse Modeling Framework (Steinberg *et al.*, 2008). Acceleo es un lenguaje para definir plantillas de generación de código.

La Figura 6-9 presenta la plantilla de transformación M2T del elemento *Elastic IP* del metamodelo de Amazon Web Services (ver Sección 5.4.2). En este caso, a partir de un modelo de Amazon se realizan transformaciones M2T para obtener un script para la herramienta Ansible (DeHaan, 2012). El módulo (línea 2) define el nombre de la plantilla y realiza la importación del metamodelo *Amazon.ecore*. El módulo está directamente relacionado con la metaclass *Elastic*

IP. Las etiquetas *template* (líneas 4-19) contienen las instrucciones para crear un elemento *Elastic IP* en Amazon Web Services. Por un lado, el código estático define las instrucciones que comparten todos los scripts de aprovisionamiento. Por ejemplo, el módulo *ec2_eip* (línea 6) de Ansible especifica que se debe crear un elemento *Elastic IP*. Por otro lado, el código dinámico completa la información específica para cada script de aprovisionamiento. Por ejemplo, se debe completar el nombre (línea 5) de cada elemento *Elastic IP*.

```

StaticIP.mtl
1 [comment encoding = UTF-8 /]
2 [module StaticIP('platform:/resource/edu.cloud.amazon.dsl/model/Amazon.ecore')]
3
4 [template public getEIP(IPname: String, VM:String) post(trim())]
5 - name: associate new elastic IPs [IPname/]
6   ec2_eip:
7     in_vpc: yes
8     reuse_existing_ip_allowed: yes
9     region: "{{ region }}"
10    device_id: "{{ item }}"
11    with_items: "{{ [VM/].instance_ids }}"
12    register: [IPname/]_facts
13 - name: output the IP
14   debug:
15     msg: "{{ item.0 }} -->  {{ item.1 }}"
16   with_together:
17     - "{{ [IPname/]_facts.results | map(attribute='allocation_id')|list }}"
18     - "{{ [IPname/]_facts.results | map(attribute='public_ip')|list }}"
19 [/template]

```

Figura 6-9 Plantilla de transformación M2T para Ansible

```

AvailabilitySet.mtl
1 [comment encoding = UTF-8 /]
2 [module AvailabilitySet('platform:/resource/edu.cloud.azure/model/Azure.ecore')]
3 [import edu::cloud::argon::m2t::terraform::computing::Tag /]
4
5 [template public getAvailabilitySet(aset:AvailabilitySet)]
6 ## Create a availability set
7 resource "azurerm_availability_set" "[aset.name/]" {
8   name                = "[aset.name/]"
9   location             = "${azurerm_resource_group.azure.location}"
10  resource_group_name  = "${azurerm_resource_group.azure.name}"
11  platform_update_domain_count = "[aset.platform_update_domain_count/]"
12  platform_fault_domain_count = "[aset.platform_fault_domain_count/]"
13  managed              = true
14  [if (aset.tags->size()>0)]
15    [getTags(aset.tags)/]
16  [/if]
17 }
18 [/template]

```

Figura 6-10 Plantilla de transformación M2T para Terraform

Por otro lado, la Figura 6-10 muestra la plantilla de transformación M2T del elemento *Availability Set* del metamodelo de Microsoft Azure (ver Sección 5.4.3). En este caso, a partir de un modelo de Azure se realizan las transformaciones

M2T para obtener un script para la herramienta Terraform (HashiCorp, 2017). El módulo (línea 2) define el nombre de la plantilla y realiza la importación del metamodelo *Azure.ecore*. Además, se realiza la importación de la metaclass *Tag* (línea 3). Las etiquetas *template* (líneas 5-18) contienen las instrucciones para crear un elemento *Availability Set* en Microsoft Azure. Por un lado, el código estático define las instrucciones que comparten todos los scripts de Terraform. Por ejemplo, para crear un recurso *Availability Set* (línea 7) se define la instrucción “*resource azure_rm_availability_set*”. Por otro lado, el código dinámico completa la información específica para cada script de Terraform. Por ejemplo, para completar el nombre del recurso *Availability Set* (línea 8) se utiliza el atributo *name*.

La plantilla para la herramienta Ansible que se muestra en la Figura 6-9 y la plantilla para la herramienta Terraform que se muestra en la Figura 6-10 son ejemplos que han sido implementado satisfactoriamente con la herramienta ARGON. En este caso, a partir de un modelo de infraestructura para un proveedor de servicios IaaS se puede generar scripts para diferentes herramientas de aprovisionamiento, tales como Terraform, Ansible, etc. En esta sección, se han presentado los ejemplos de plantillas para Ansible y Terraform. Sin embargo, ARGON no está limitado a estas herramientas y puede agregar plantillas para otras herramientas de aprovisionamiento, tales como CloudFormation (Amazon Web Services, 2011), OpenStack Heat (OpenStack Foundation, 2013), etc.

6.3 Un pipeline para el aprovisionamiento de infraestructura en la nube

En muchos proyectos de software, la liberación y distribución de aplicaciones de software es un proceso manual y los entornos que alojan los proyectos de software a menudo son diseñados individualmente por el personal de operaciones. En este escenario, existe una gestión manual de la configuración del entorno de producción, y su implementación se lleva a cabo una vez que se ha completado el desarrollo. Debido a todos estos problemas, el aprovisionamiento de infraestructura puede llevar mucho tiempo y ser propenso a errores. Para abordar estos problemas, se propone un *pipeline* (Sandobalín *et al.*, 2017b) para la automatización del aprovisionamiento de infraestructura en la nube. En este caso, el pipeline soporta las tareas de control de versiones, generación de scripts, pruebas y aprovisionamiento de infraestructura en la nube.

El enfoque MoCIP utiliza el *pipeline* para soportar las prácticas de integración continua (*Continuous Integration*, CI) y despliegue continuo (*Continuous Deployment*,

CD) en el aprovisionamiento de infraestructura en la nube. El pipeline utiliza la herramienta ARGON —junto con su lenguaje ArgonML— y un conjunto de herramientas de la comunidad DevOps. En este caso, el encadenamiento de herramientas provee la *abstracción* necesaria para mitigar la complejidad del uso de lenguajes de scripting en la configuración de las prácticas de CI y CD.

Las principales ventajas de utilizar el pipeline con ARGON y las herramientas DevOps son las siguientes:

- La infraestructura puede ser fácilmente creada, destruida, reemplazada y redimensionada.
- Es posible reconstruir cualquier elemento de la infraestructura con menos esfuerzo y de manera confiable.
- Es posible construir elementos idénticos de infraestructura en diferentes entornos. Esto permite alcanzar el principio de *paridad*, que significa tener la misma infraestructura y configuraciones en diferentes entornos.
- Es posible obtener el principio de *reproducibilidad*, es decir, cualquier acción que se lleve a cabo en la infraestructura debe ser repetible.
- Los cambios realizados en el modelo de infraestructura pueden ser reflejados en la infraestructura de forma rápida y segura.

La Figura 6-11 presenta una descripción general del *pipeline* de aprovisionamiento de infraestructura en la nube. El lenguaje ArgonML permite modelar los recursos de infraestructura. Se utiliza la herramienta ARGON para generar un modelo de infraestructura para un proveedor específico de servicios IaaS.

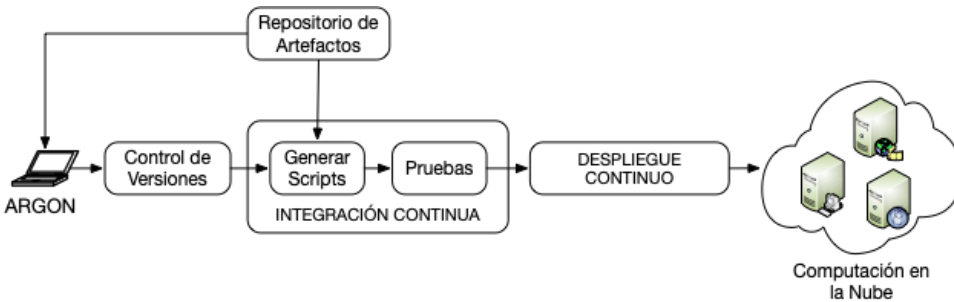


Figura 6-11 Descripción general del *pipeline* de aprovisionamiento de infraestructura

Para iniciar la práctica de CI se envía el modelo de infraestructura hacia un sistema de *Control de Versiones*. La práctica de CI requiere que cada vez que exista un cambio en el modelo de infraestructura, se realiza automáticamente la *Generación de Scripts* y se ejecute un conjunto de *Pruebas* automatizadas. Además, se utiliza un *Repositorio de Artefactos* para proporcionar bibliotecas de software, tales como el motor de transformaciones M2T y el lenguaje de modelado

ArgonML. En este caso, el *Repositorio de Artefactos* proporciona un único proveedor de bibliotecas o artefactos de software en las etapas de desarrollo y generación de scripts.

Por otro lado, la práctica de CD utiliza herramientas de aprovisionamiento con los scripts que han superado el conjunto de pruebas automatizadas para realizar el aprovisionamiento de la infraestructura en la nube. Finalmente, se realizan pruebas en los paquetes de software instalados en la infraestructura para garantizar su correcto funcionamiento.

6.3.1 Proyecto de infraestructura

La herramienta ARGON permite crear un proyecto de infraestructura en el entorno de desarrollo integrado Eclipse. La Figura 6-12 presenta un proyecto de infraestructura (*edu.issi.cloud.testing*) junto con un diagrama de infraestructura. En este caso, el diagrama representa el estado final de la infraestructura que será aprovisionada en Amazon Web Services.

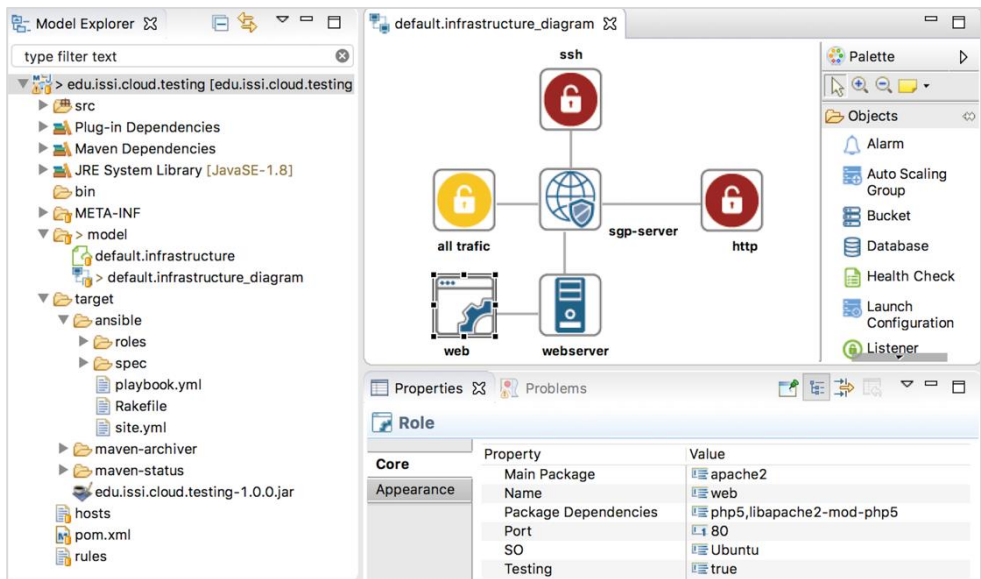


Figura 6-12 Proyecto de infraestructura creado por la herramienta ARGON

El diagrama de infraestructura muestra un grupo de seguridad (*sgp-server*) que trabaja como un cortafuegos (*firewall*) para habilitar las conexiones a una máquina virtual (*webservice*). El grupo de seguridad tiene dos reglas de entrada con el fin de permitir las conexiones hacia la máquina virtual por el puerto 22 (*ssh*) y por el puerto 80 (*http*). Además, una regla de salida (*all traffic*) habilita todas las conexiones desde la máquina virtual hacia el exterior. Adicionalmente, un

elemento rol (*web*) permite instalar en la máquina virtual (*webserv*) un servidor web Apache (*apache2*) y sus dependencias (*php5*, *libapache2-mod-php5*).

El elemento rol (*web*) permite configurar las siguientes propiedades: *My Package* define la instalación de un servidor web Apache (*apache2*); *Name* determina el nombre del elemento rol; *Package Dependencies* especifica todos los paquetes de software que necesita el paquete o software principal (*My Package*); *Port* define el número del puerto que se utilizará para responder a las peticiones de los usuarios; *OS* representa el sistema operativo donde los paquetes de software serán instalados; *Testing* habilita la ejecución de un conjunto de pruebas automatizadas.

```

12  <repositories>
13    <repository>
14      <id>nexus</id>
15      <url>http://nexusserver:8081/nexus/content/groups/public/</url>
16    </repository>
17  </repositories>
18
19  <dependencies>
20    <dependency>
21      <groupId>edu.issi.cloud.ansible.playbook</groupId>
22      <artifactId>transformation-engine</artifactId>
23      <version>1.0.9</version>
24    </dependency>
25    <dependency>
26      <groupId>edu.issi.cloud</groupId>
27      <artifactId>dsl</artifactId>
28      <version>1.0.3</version>
29    </dependency>
30  </dependencies>

```

Figura 6-13 Extracto de código del archivo POM del proyecto de infraestructura

Se utiliza la herramienta Maven junto con su archivo de configuración POM (*Project Object Model*) para configurar el proyecto de infraestructura (*edu.issi.cloud.testing*). POM es la unidad fundamental de trabajo en Maven. POM es un archivo XML (*eXtensible Markup Language*) que contiene información del proyecto y los detalles de la configuración utilizados por Maven. La Figura 6-13 presenta un extracto del archivo POM del proyecto de infraestructura. En este caso, el archivo POM muestra la configuración del *Repositorio de Artefactos* (líneas 13-16) donde el atributo *id* es el único identificador del repositorio y el atributo *url* es el localizador de los recursos o dependencias de software. Además, el archivo POM define la versión del motor de transformaciones M2T (líneas 22-23) y la versión del lenguaje de modelado ArgonML (líneas 27-28).

6.3.2 Gestión de la configuración

La gestión de la configuración se refiere al proceso por el cual todos los artefactos relevantes para un proyecto, y las relaciones entre los artefactos, se

almacenan, recuperan, identifican y modifican de forma única (Humble *et al.*, 2010). Para obtener un encadenamiento efectivo de herramientas DevOps, es necesario proporcionar una estrategia de gestión de la configuración que determine cómo administrar los cambios que ocurren dentro del proyecto de infraestructura.

6.3.2.1 Sistema de control de versiones

Un sistema de control de versiones es un mecanismo para mantener múltiples versiones de archivos, de tal manera que cuando se modifique un archivo, se puede acceder a las versiones anteriores. Una vez que el modelo de infraestructura está listo, se envía el modelo hacia un sistema de control de versiones. En este caso, se utiliza GitHub (Preston-Werner *et al.*, 2008) como un sistema de control de versiones distribuido. GitHub tiene un repositorio local y un repositorio remoto (ver Figura 6-14). El repositorio local le proporciona a cada desarrollador una copia local del historial completo de los proyectos de infraestructura. En cambio, el repositorio remoto importa todas las versiones del proyecto de infraestructura y los fusiona en un único proyecto.

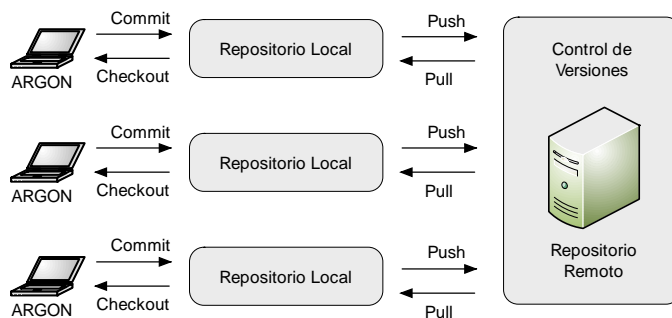


Figura 6-14 Descripción general del control de versiones con GitHub

La Figura 6-14 presenta la descripción general del sistema de control de versiones de GitHub. El flujo de trabajo de GitHub comienza con el envío de un proyecto de infraestructura hacia el repositorio local. Es posible realizar numerosos envíos hacia el repositorio local y mantener un historial local de los cambios realizados en el modelo de infraestructura. Sin embargo, cuando se debe fusionar todas las versiones del modelo, es necesario enviar el proyecto de infraestructura hacia el repositorio remoto.

Por otro lado, cada vez que se realizan cambios en el modelo de infraestructura almacenado en el repositorio remoto, se hace necesario obtener la última versión del modelo de infraestructura en el repositorio local. Por lo tanto, para trabajar con un proyecto de infraestructura se convierte en obligatorio enviar el proyecto

de infraestructura a los repositorios local y remoto. La Figura 6-15 muestra el proyecto de infraestructura (*edu.isi.cloud.testing*) en GitHub. En este caso, el proyecto de infraestructura está listo para pasar a la etapa de CI.

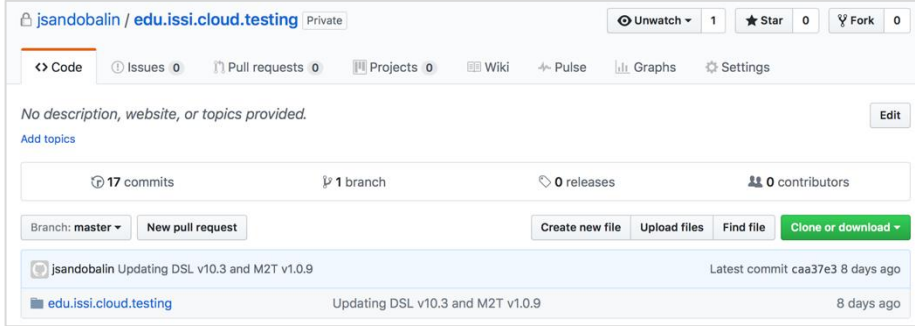


Figura 6-15 Proyecto de infraestructura en GitHub

6.3.2.2 Repositorio de artefactos

Las librerías de software se administran de manera diferente a un sistema de control de versiones, debido a que generalmente las librerías se implementan en forma de archivos binarios o JAR (*Java ARchive*), nunca son modificadas por el equipo de desarrollo, y porque rara vez se actualizan (Humble *et al.*, 2010). Por lo tanto, se hace necesario utilizar un repositorio de artefactos para la gestión de una colección de JAR y sus metadatos.

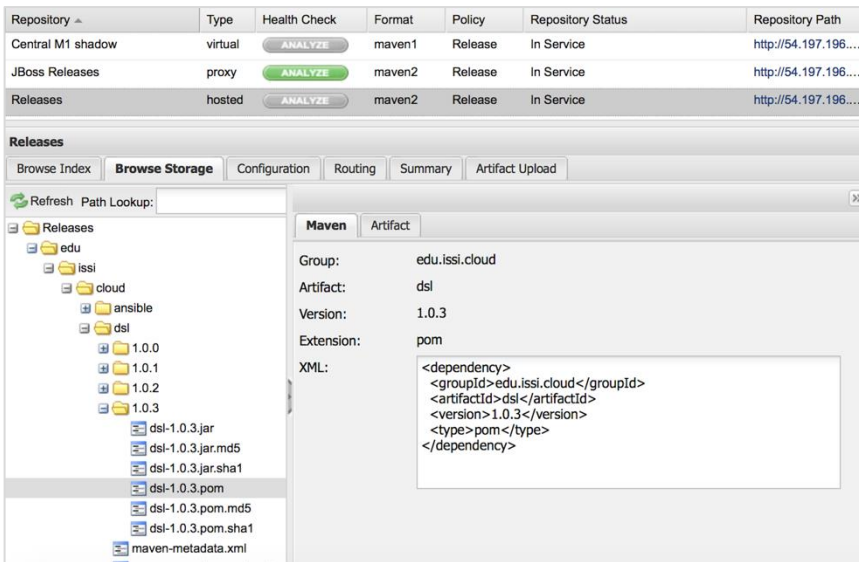


Figura 6-16 Artefactos del proyecto de infraestructura en Nexus

En el caso del proyecto de infraestructura, los artefactos JAR son gestionados por Maven. Se utiliza Nexus (Sonatype, 2008) como repositorio de artefactos. La Figura 6-16 muestra los artefactos del proyecto de infraestructura en el repositorio Nexus. La herramienta ARGON solicita los artefactos al repositorio Nexus a través de un archivo POM (ver Figura 6-13) para resolver las dependencias de sus artefactos JAR, es decir, el lenguaje ArgonML y los motores de transformación de modelos. Sin embargo, ARGON también necesita resolver las dependencias para compilar con Maven y realizar las transformaciones M2T con Acceleo. En este caso, Nexus trabaja como un proxy que proporciona artefactos del repositorio central de Maven y del repositorio de Acceleo. La Figura 6-17 presenta la arquitectura del repositorio Nexus para proveer los artefactos a la herramienta ARGON y a un servidor de CI.

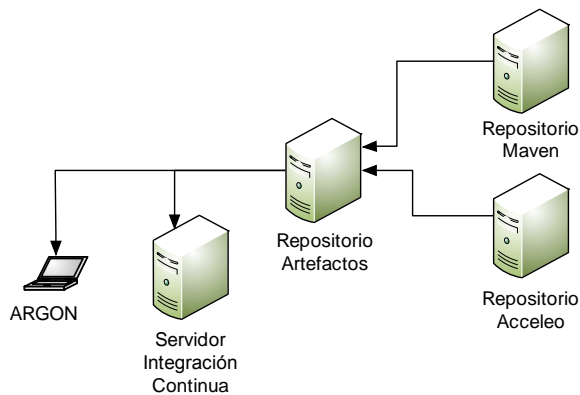


Figura 6-17 Descripción general del repositorio de artefactos

6.3.3 Integración continua

La integración continua requiere que cada vez que se realice un cambio en el modelo de infraestructura, se construya todos los scripts de aprovisionamiento y se ejecute un conjunto integral de pruebas automatizadas contra los scripts generados. Por esta razón, en el caso del aprovisionamiento de infraestructura, el objetivo de la integración continua es que los scripts permanezcan en un estado de aprovisionamiento todo el tiempo.

La integración continua se basa en ciertos requisitos previos, tales como (1) los modelos en el proyecto de infraestructura deben registrarse en el sistema de control de versiones, y (2) el proceso de generación de scripts debe ejecutarse de forma automatizada desde un entorno de CI. Por lo tanto, el proyecto de infraestructura almacenado en GitHub se usa como entrada para la etapa de CI. Se utiliza Jenkins (Kawaguchi, 2011) como servidor de CI para ejecutar el proceso de generación de scripts de manera automatizada. Además, se ejecuta

un conjunto completo de pruebas automatizadas contra los scripts generados. La Figura 6-18 presenta una descripción general del pipeline de CI de un script para la herramienta de aprovisionamiento Ansible (DeHaan, 2012). En este caso, el objetivo del pipeline de CI es obtener —al final del proceso— un script en estado de liberación (*release*), de tal manera que sea la entrada para la etapa de CD.



Figura 6-18 Descripción general del pipeline de CI

6.3.3.1 Generación de scripts

El motor de transformaciones M2T se apoya en la herramienta Maven para trabajar con Jenkins y ejecutar el proceso de generación de scripts de forma automatizada. Se utiliza el servidor Jenkins, debido a que soporta las prácticas de CI y CD en proyectos de software. Es importante resaltar que se puede ejecutar Jenkins mediante línea de comandos. Esta característica permite realizar una configuración basada en Maven para la generación de scripts y la ejecución de pruebas automatizadas sobre los scripts generados.

```

97 <packagesToRegister>
98   <packageToRegister>infrastructure.InfrastructurePackage</packageToRegister>
99 </packagesToRegister>
100 </configuration>
101 </plugin>
102
103 <plugin>
104   <groupId>org.eclipse.acceleo</groupId>
105   <artifactId>org.eclipse.acceleo.maven.launcher</artifactId>
106   <version>3.6.4</version>
107   <executions>
108     <execution>
109       <phase>process-resources</phase>
110     </execution>
111   </executions>
112   <configuration>
113     <generatorClass>edu.issi.cloud.ansible.playbook.main.Generate</generatorClass>
114     <model>${model.infrastructure}</model>
115     <outputFolder>${folder.script}</outputFolder>
116   </configuration>
117 </plugin>
  
```

Figura 6-19 Extracto de código del archivo POM para el motor de transformaciones M2T

La Figura 6-19 muestra un extracto del archivo POM donde se configura el motor de transformaciones M2T. Se establece el paquete de infraestructura (línea 98) en donde está la clase Java que inicia el proceso de transformaciones M2T, es decir, la generación de scripts. Además, se especifica las librerías de Acceleo (líneas 104-106) y la fase (línea 109) de ejecución del servidor Jenkins donde se

realizarán las transformaciones M2T. Además, se debe especificar la clase de Acceleo (línea 113) que inicia las transformaciones M2T. Finalmente, se define la carpeta (línea 114) que contiene el modelo de infraestructura y la carpeta (línea 115) donde se almacenarán los scripts generados.

La Figura 6-20 presenta el panel de proyectos del servidor Jenkins. El pipeline de aprovisionamiento de infraestructura (*infrastructure-provisioning-pipeline*) muestra que el último pipeline creado con éxito es el número 72 con un tiempo de 9 minutos y 16 segundos. En cambio, el último pipeline que falló en el aprovisionamiento de infraestructura es el número 60, hace 8 días con 23 horas.

S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración
●	☀	infrastructure-provisioning-pipeline	9 Min 16 Seg - #72	8 días 23 Hor - #68	8 Min 10 Seg
●	☀	transformation-engine	8 días 12 Hor - #28	10 días - #24	1 Min 17 Seg

Figura 6-20 Panel de proyectos del servidor Jenkins

6.3.3.2 Conjunto de pruebas automatizadas para los scripts

El objetivo de las pruebas automatizadas es mantener la alta calidad de los sistemas mediante la identificación de errores tan pronto como se producen, de tal manera que puedan ser reparados inmediatamente (Morris, 2016). Durante el proceso de desarrollo de software se deben ejecutar tres tipos de pruebas en la etapa de CI, tales como pruebas unitarias, pruebas de componentes y pruebas de aceptación. Sin embargo, estas pruebas no se pueden aplicar al aprovisionamiento de infraestructura debido a que pertenecen a un contexto diferente. Por esta razón, se utiliza el conjunto de pruebas automatizadas para el aprovisionamiento de infraestructura propuesto en (Morris, 2016). Primero, las pruebas de *Verificación de Sintaxis* se ejecutan para validar la estructura del código de los scripts. Debido a que se generaron scripts para Ansible, los scripts utilizan el lenguaje de scripting YAML (Evans, 2001). En este caso, se utiliza la herramienta Ansible para verificar la validez de la sintaxis de los scripts escritos en YAML, así como algunos problemas cosméticos, tales como la longitud de línea, espacios finales, sangría, etc.

La Figura 6-21 muestra el resultado de la prueba de verificación de la sintaxis del script de aprovisionamiento. Se configura Ansible junto con el servidor Jenkins para ejecutar las pruebas de verificación de la sintaxis.


```
[infrastructure-provisioning-pipeline] $ /bin/sh -xe /tmp/hudson6922122130064775578.sh
+ cd edu.issi.cloud.testing/
+ ansible-playbook -i /etc/ansible/ec2.py --private-key /home/ubuntu/.ssh/kp-ireland.pem
target/ansible/playbook.yml --extra-vars pub_key_file=/home/ubuntu/.ssh/id_rsa.pub --syntax-check
```

Figura 6-21 Pruebas de verificación de sintaxis de un script de aprovisionamiento

```
playbook: target/ansible/site.yml
[infrastructure-provisioning-pipeline] $ /bin/sh -xe /tmp/hudson8602701116534883827.sh
+ cd edu.issi.cloud.testing/
+ ansible-playbook -i /etc/ansible/ec2.py --private-key /home/ubuntu/.ssh/kp-ireland.pem
target/ansible/playbook.yml --extra-vars pub_key_file=/home/ubuntu/.ssh/id_rsa.pub --check

PLAY [Infrastructure provisioning in the Cloud] *****

TASK [Provisioning Security Group(s)] *****
changed: [localhost]

TASK [Provisioning EC2 instance(s)] *****
skipping: [localhost]

TASK [Add new instance to host group] *****
fatal: [localhost]: FAILED! => {"failed": true, "msg": "'dict object' has no attribute 'instances'"}
...ignoring

TASK [Waiting for the new instance to be ready] *****
fatal: [localhost]: FAILED! => {"failed": true, "msg": "'dict object' has no attribute 'instances'"}
...ignoring

PLAY RECAP *****
localhost                : ok=3    changed=1    unreachable=0    failed=0
```

Figura 6-22 Pruebas de comprobación estática de un script de aprovisionamiento

Por otro lado, las *Pruebas de Código Estático* analizan el código de los scripts sin ejecutar el aprovisionamiento de infraestructura. La Figura 6-22 presenta el resultado de la prueba de verificación de código. En este caso, se configura Ansible en modo de verificación (*check mode*) junto con el servidor Jenkins para ejecutar las pruebas de código estático. El modo de verificación es solo una simulación y no hará ningún cambio en el proveedor de servicios IaaS. Los módulos de Ansible que admitan el modo de verificación informarán qué cambios habrían tenido lugar, en lugar de ejecutarlos.

6.3.4 Despliegue continuo

El aprovisionamiento de la infraestructura requiere una serie de pasos, tales como la configuración de herramientas, infraestructura, sistemas operativos y middleware. A medida que los proyectos se vuelven más complejos, estos pasos se vuelven más numerosos, más largos y propensos a errores. Por lo tanto, es necesario utilizar herramientas que automaticen el aprovisionamiento de infraestructura en la nube, tales como Ansible o Terraform. Además, estas herramientas deben ayudar en la instalación del middleware y de los paquetes de software en la infraestructura aprovisionada en la nube.

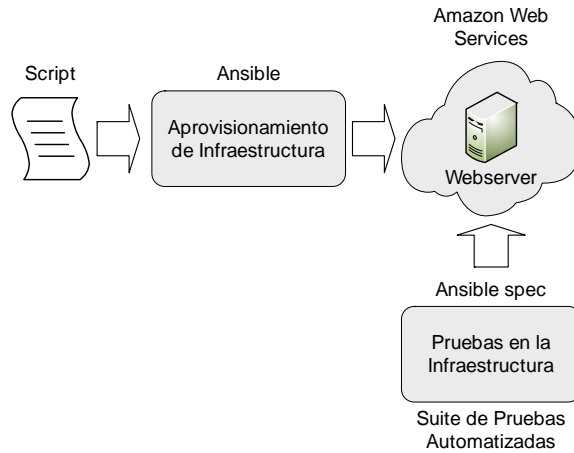


Figura 6-23 Descripción general del despliegue continuo de infraestructura

La Figura 6-23 presenta la descripción general del CD de infraestructura. La etapa de CD empieza una vez que los scripts han superado un conjunto de pruebas automatizadas. Se configura los comandos para ejecutar los scripts con la herramienta Ansible en el servidor Jenkins. En este escenario, el servidor Jenkins inicia y gestiona el proceso de aprovisionamiento de infraestructura. En el caso que el aprovisionamiento sea satisfactorio se mostrará en el panel de proyectos de Jenkins un mensaje de éxito. Es el caso contrario, cuando falle el aprovisionamiento, se mostrará en el panel de Jenkins un mensaje de error y sus posibles causas.

Name	Instance ID	Instance	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)
webservers	i-0cbca7febd11036d7	t1.micro	eu-west-1b	running	2/2 checks ...	None	ec2-34-253-159-2.eu-w...

Instance: i-0cbca7febd11036d7 (webservers) Public DNS: ec2-34-253-159-2.eu-west-1.compute.amazonaws.com			
Description	Status Checks	Monitoring	Tags
Instance ID	i-0cbca7febd11036d7	Public DNS (IPv4)	ec2-34-253-159-2.eu-west-1.compute.amazonaws.com
Instance state	running	IPv4 Public IP	34.253.159.2
Instance type	t1.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172-31-38-241.eu-west-1.compute.internal
Availability zone	eu-west-1b	Private IPs	172.31.38.241
Security groups	sgp-server. view inbound rules	Secondary private IPs	
Scheduled events	No scheduled events	VPC ID	vpc-e75c3983
AMI ID	ubuntu/images/eks/ubuntu-trusty-14.04-amd64-server-20170405-ecd5575e-d805-450e-843e-f2a9872b8c80-ami-6324a775.4 (ami-8fae95e9)	Subnet ID	subnet-694cf831

Figura 6-24 Máquina virtual (*webservers*) en Amazon Web Services

La Figura 6-24 presenta el panel de control de Amazon Web Services, en donde se muestra la máquina virtual (*webserver*) creada con éxito y en funcionamiento (*running*). La máquina virtual está conectada al grupo de seguridad (*sgp-server*) con el fin de habilitar las conexiones hacia la máquina virtual por el puerto 22 y por el puerto 80. Además, el grupo de seguridad (*sgp-server*) habilita todas las conexiones desde la máquina virtual hacia el exterior. Adicionalmente, se presenta información adicional sobre la máquina virtual (*webserver*), por ejemplo: la dirección IP pública (*34.253.159.2*); el tipo de instancia (*t1.micro*) que especifica características de hardware, tales como 1 CPU virtual, 0.5 GB de memoria RAM; el identificado del sistema operativo (*ID AMI*), entre otros.

6.3.4.1 Conjunto de pruebas automatizadas para la infraestructura

Una vez que los scripts han superado con éxito el conjunto de pruebas automatizadas en la etapa de CI y que han apoyado el aprovisionamiento de la infraestructura en la nube, el siguiente paso es ejecutar un conjunto de pruebas en los paquetes de software instalados en la infraestructura aprovisionada en Amazon Web Services. Las pruebas se realizan con el fin de verificar que la máquina virtual (*webserver*) funcione correctamente y que los paquetes de software están instalados satisfactoriamente. En este caso, para ejecutar las pruebas en la infraestructura se utiliza la herramienta *Ansible_spec* (Volanija, 2013).

Ansible_spec es una herramienta que permite escribir pruebas simples destinadas a verificar que una máquina virtual está configurada correctamente. Se puede utilizar *Ansible_spec* para validar el estado de una máquina virtual de manera remota a través de una conexión SSH (*Secure SHell*). *Ansible_spec* se configura en el servidor Jenkins para gestionar las pruebas en infraestructura en la máquina virtual aprovisionada de Amazon Web Services. Es importante mencionar que la herramienta ARGON permite especificar en cada elemento rol (ver Figura 6-12) la creación de un conjunto automatizado de pruebas en la infraestructura. En este caso, junto con los scripts de aprovisionamiento se crean los archivos con toda la información necesaria para ejecutar las pruebas.

A continuación, se enumera los cuatro tipos de pruebas que se ejecutan en la máquina virtual desplegada en Amazon Web Services.

- Comprobación que los paquetes de software están instalados.
- Comprobación que los paquetes de software están habilitados.
- Comprobación que los paquetes de software están ejecutándose.
- Comprobación que los paquetes de software están atendiendo las peticiones de los usuarios a través de un puerto específico.

```
[infrastructure-provisioning-pipeline] $ /bin/sh -xe /tmp/hudson3827965938277251648.sh
+ cd edu.issi.cloud.testing/target/ansible/
+ /home/ubuntu/.rbenv/versions/2.2.3/bin/rake all
Run serverspec for webservers-TDD to {"uri"=>"52.17.231.186", "port"=>22}
/home/ubuntu/.rbenv/versions/2.2.3/bin/ruby -
I/home/ubuntu/.rbenv/versions/2.2.3/lib/ruby/gems/2.2.0/gems/rspec-support-
3.5.0/lib:/home/ubuntu/.rbenv/versions/2.2.3/lib/ruby/gems/2.2.0/gems/rspec-core-3.5.4/lib
/home/ubuntu/.rbenv/versions/2.2.3/lib/ruby/gems/2.2.0/gems/rspec-core-3.5.4/exe/rspec --pattern \{roles\}/\
{web\}/spec/*_spec.rb

Package "apache2"
  should be installed

Service "apache2"
  should be enabled
  should be running

Port "80"
  should be listening

Finished in 3.72 seconds (files took 1.72 seconds to load)
4 examples, 0 failures
```

Figura 6-25 Pruebas en la infraestructura creada en Amazon Web Services

Como resultado, la Figura 6-25 presenta el resultado del conjunto de pruebas realizadas en el servidor web Apache (*apache2*) instalado en la máquina virtual (*webservers*). Es resultado de las pruebas muestra que el servidor web Apache está instalado, habilitado, ejecutándose y atendiendo las peticiones de los usuarios a través del puerto 80.

6.4 Resumen

Las herramientas que soportan el enfoque MoCIP utilizan técnicas dirigidas por modelos para *abstraer* y *automatizar* el aprovisionamiento de infraestructura en la nube. En primer lugar, el lenguaje ArgonML abstrae las capacidades de la computación en la nube para definir un metamodelo de infraestructura. El metamodelo representa la *sintaxis abstracta* del lenguaje y determina todos los modelos de infraestructura válidos. Además, ArgonML define una notación gráfica para modelar los elementos de la infraestructura. La notación gráfica representa la *sintaxis concreta* del lenguaje.

Los modelos no son entidades aisladas ni estáticas. Como parte de un proceso MDE, los modelos se traducen a otros lenguajes o representaciones. En este escenario, el lenguaje ArgonML debe combinarse con herramientas de soporte para maximizar la productividad al definir los recursos de infraestructura. En consecuencia, el lenguaje ArgonML apoya a la herramienta ARGON en el aprovisionamiento de infraestructura en la nube.

ARGON utiliza MDE para *abstraer* y *automatizar* el aprovisionamiento de infraestructura en la nube. Por un lado, ARGON realiza transformaciones M2M para obtener modelos de infraestructura para cada proveedor de servicios en la

nube. En este caso, se obtienen los metamodelos de infraestructura de cada proveedor de servicios IaaS y se realiza los mapeos entre los conceptos de los metamodelos. Los metamodelos y los mapeos se utilizan para definir los módulos de transformación M2M. Por otro lado, ARGON realiza transformaciones M2T para obtener los scripts para realizar el aprovisionamiento de infraestructura en la nube. En este caso, se *abstrae* las características de los lenguajes de scripting —de las herramientas de aprovisionamiento— para definir plantillas de transformación M2T. En las plantillas se agrega toda la información específica de la tecnología subyacente a cada proveedor de servicios IaaS.

Por otro lado, en muchos proyectos de software existe una gestión manual de la creación y configuración del entorno de producción, y su implementación se lleva a cabo una vez que se ha completado el desarrollo. Con el fin de mitigar estos problemas se propone un pipeline para la automatización del aprovisionamiento de infraestructura en la nube. Primero, se utiliza la herramienta ARGON para crear un proyecto de infraestructura en el entorno de desarrollo integrado Eclipse. En este caso, el lenguaje de modelado ArgonML ayuda a definir la infraestructura. Luego, la herramienta ARGON genera el modelo de infraestructura para un proveedor específico de servicios IaaS. Se utilizan herramientas de la comunidad DevOps para gestionar las tareas de control de versiones, generación de scripts, pruebas y aprovisionamiento de infraestructura en la nube. Finalmente, se implementa las prácticas de integración y despliegue continuos en el proceso de aprovisionamiento de la infraestructura.

Para iniciar la etapa de integración continua, se debe enviar el proyecto de infraestructura —junto con el modelo de infraestructura— a un sistema de control de versiones. En este caso, se utiliza GitHub como sistema de control de versiones. Es importante enviar cada cambio realizado en el modelo de infraestructura hacia GitHub con el fin de obtener un historial de la definición de la infraestructura. Luego, el servidor Jenkins toma el modelo de infraestructura desde GitHub y genera automáticamente los scripts de aprovisionamiento a partir del modelo. Una vez generado el script se aplica un conjunto automatizado de pruebas para validar la sintaxis del script y la comprobación estática de la infraestructura. Se utiliza Acceleo para la generación de scripts y Ansible para realizar el conjunto de pruebas. Además, se utiliza Maven para automatizar las tareas de la etapa de integración continua. Por otro lado, con el fin de obtener un único distribuidor de librerías o artefactos de software se utiliza Nexus como un repositorio de artefactos. Nexus provee las diferentes versiones de los artefactos, tales como ArgonML, ARGON, Acceleo, Maven, etc. Finalmente, se obtiene un script listo para ser utilizado en la siguiente etapa.

La etapa de despliegue continuo inicia con un script de aprovisionamiento que ha superado un conjunto automatizado de pruebas. La herramienta Ansible utiliza el script para orquestar el aprovisionamiento de infraestructura en la nube. Para automatizar el proceso de aprovisionamiento, se configura la herramienta Ansible en el servidor Jenkins. Finalmente, se realiza un conjunto de pruebas sobre los paquetes de software instalados en la infraestructura aprovisionada en la nube. Se utiliza la herramienta Ansible_spec para verificar que los paquetes de software estén instalados, habilitados, ejecutándose y que atiendan las peticiones de los usuarios a través de un puerto específico.

Capítulo 7

Validación de MoCIP

En un proyecto *Design Science*, el objeto de estudio es un artefacto en contexto, y sus dos actividades principales son el diseño y la investigación del artefacto en contexto (Wieringa, 2014). En este capítulo se presenta la investigación del artefacto MoCIP en contexto. Se utiliza un ciclo empírico para responder a las preguntas de investigación sobre la interacción entre el artefacto MoCIP y el contexto del problema. En este caso, para realizar el ciclo empírico se diseñó y ejecutó dos experimentos controlados.

En la Sección 7.1 se presentan las actividades de MoCIP a través de las tareas experimentales que los participantes realizaron en los experimentos.

En la Sección 7.2 se presenta el diseño de los experimentos, en donde se define el objetivo de los experimentos, las preguntas de investigación, la selección del contexto de los experimentos y las tareas experimentales.

En la Sección 7.3 se presenta la estadística descriptiva y un análisis de datos exploratorio de los resultados. Además, se presentan las pruebas de hipótesis para contestar a las preguntas de investigación.

En la Sección 7.4 se presentan las amenazas a la validez de los experimentos.

En la Sección 7.5 se presentan las conclusiones de los resultados de los dos experimentos controlados.

7.1 MoCIP

MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*) es un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube. MoCIP provee soporte a la Infraestructura como Código (*Infrastructure as Code, IaC*) en el aprovisionamiento de infraestructura utilizando la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering, MDE*).

En esta sección se presentan las actividades de MoCIP para realizar el proceso de aprovisionamiento de infraestructura Amazon Web Services. En este caso, los participantes realizaron las tres actividades de MoCIP con el fin de realizar el aprovisionamiento de infraestructura en Amazon Web Services.

7.1.1 Elicitación de requisitos

Para realizar la primera actividad se propone dos ejercicios de dominios diferentes. A continuación, se resume los enunciados de los ejercicios.

7.1.1.1 La universidad CEC

La Universidad CEC (Sandobalín *et al.*, 2017a) ofrece cursos masivos y gratuitos a través de su plataforma virtual. En los últimos años la demanda de cursos en línea se ha incrementado debido principalmente a que los MOOC (*Massive Open Online Course*) han supuesto una revolución en el campo de la educación, especialmente en la educación universitaria. Esto ha ocasionado que diferentes cursos en la plataforma virtual tengan cientos de estudiantes utilizando simultáneamente lecciones de video, y otros materiales multimedia. Los cursos están alojados en servidores que están localizados geográficamente en las instalaciones de la Universidad CEC en Dublín, Irlanda.

El principal problema que enfrenta la plataforma virtual de la Universidad CEC es la alta demanda que existe para ciertos cursos, lo que ocasiona una sobre carga de trabajo en los servidores donde están alojados. La Universidad CEC ha decidido solucionar su problema comprando nuevos servidores con mayor capacidad de hardware, con el fin de crear un clúster. Sin embargo, los nuevos servidores no trabajarán cuando no exista demanda para los cursos, por ejemplo, en las vacaciones de verano. Para resolver este dilema la Universidad CEC ha decidido migrar su infraestructura hacia Amazon Web Services y pagar únicamente por el uso de la infraestructura desplegada bajo demanda. En este caso, el personal de operaciones decide utilizar una arquitectura con un balanceador de carga que distribuya la carga de trabajo entre varias máquinas virtuales creadas bajo demanda.

7.1.1.2 La empresa MODAFIN

La empresa MODAFIN (Nitto *et al.*, 2017) se especializa en aplicaciones TI para servicios financieros. Su línea principal de productos está relacionada con operaciones bursátiles, administración de efectivo y administración de préstamos. Las actividades más rentables de MODAFIN son la personalización del software y la gestión del ciclo de vida de su línea de productos. La personalización implica el desarrollo de módulos que se adapten a los nuevos requisitos funcionales de sus clientes.

El equipo de consultoría ha trabajado durante mucho tiempo en una aplicación de software para operaciones bursátiles que se implementará en las máquinas virtuales de Amazon Web Services. El equipo de operaciones está preocupado por el tráfico exponencial de peticiones que reciben otras aplicaciones de software que están en producción. Con el fin de dar una solución al tráfico de peticiones se decide usar una arquitectura escalable para crear o eliminar máquinas virtuales bajo demanda. Sin embargo, primero se debe realizar pruebas para distribuir la carga de trabajo entre varias máquinas virtuales. En consecuencia, el equipo de operaciones decide utilizar un balanceador de carga.

7.1.2 Modelo de infraestructura de Amazon

A pesar de que los ejercicios propuestos para la actividad de requisitos son de dominios diferentes, se obtiene (en los dos casos) una arquitectura para un balanceador de carga. Es importante mencionar que, aunque exista una arquitectura común, las propiedades de los elementos de infraestructura son diferentes para cada ejercicio.

La Figura 7-1 presenta el diagrama de infraestructura con la arquitectura para un balanceador de carga. La infraestructura ha sido modelada con el lenguaje ArgonML. Un balanceador de carga (*LB01alucloud00*) distribuye la carga de trabajo entre varias máquinas virtuales (*VM01alucloud00*). El elemento máquina virtual tiene la propiedad *Count* donde se define el número de instancias que se deben crear. Un elemento oyente (*L01alucloud00*) resuelve todas las peticiones que los clientes realizan al balanceador de carga y distribuye las peticiones a las máquinas virtuales. Además, un elemento chequeo de salud (*HC01alucloud00*) valida que todas las máquinas virtuales conectadas al balanceador de carga estén disponibles y funcionando. Por otro lado, los elementos zona (*ue-west-1a*) definen una zona de disponibilidad en una región de Amazon Web Services donde se debe aprovisionar la infraestructura. Finalmente, los grupos de seguridad funcionan como un cortafuegos (*Firewall*) que permiten las conexiones desde y hacia los recursos de infraestructura a través de reglas. El balanceador de carga tiene un grupo de seguridad (*SG01alucloud00*) con una regla de entrega (*Port80*)

que permite atender las peticiones de los usuarios a través del puerto 80, mientras que una regla de salida (*PortALL*) permite todas las conexiones desde el balanceador de carga hacia el exterior. En cambio, las máquinas virtuales tienen un grupo de seguridad (*SG02alucloud00*) con tres reglas de entrada para permitir las conexiones de los usuarios al servidor web a través del puerto 8080 (*Port8080*), una conexión FTP a través del puerto 25 (*Port25*) y una conexión SSH a través del puerto 22 (*Port22*). Adicionalmente, las máquinas virtuales tienen una regla de salida (*PortALL*) que permite todas las conexiones desde las máquinas virtuales hacia el exterior.

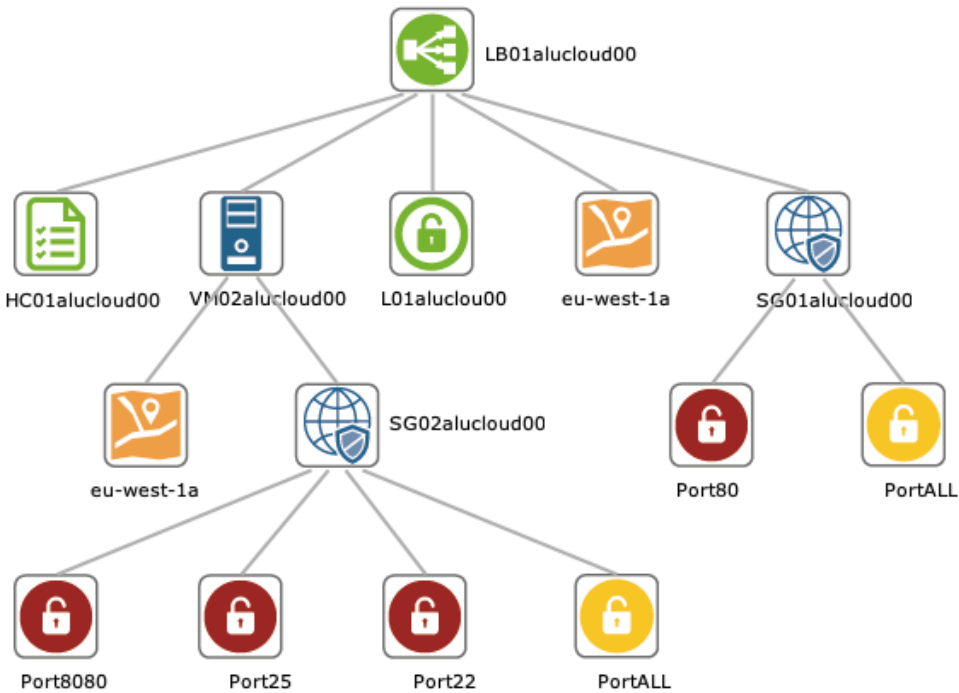


Figura 7-1 Diagrama de infraestructura modelado con el lenguaje ArgonML

7.1.3 Aprovechamiento de infraestructura

En esta actividad los participantes deben generar los scripts de aprovisionamiento para la herramienta Ansible. Se utiliza el motor de transformaciones de modelo a texto (*Model-to-Text*, M2T) de la herramienta ARGON. La entrada de las transformaciones M2T es el modelo de infraestructura definido en la actividad de modelado. En cambio, la salida de las transformaciones M2T es un script con las instrucciones para crear y ejecutar la infraestructura junto con la información de la tecnología subyacente de Amazon

Web Services. La Figura 7-2 presenta un extracto del script de aprovisionamiento para la herramienta Ansible. El script define las instrucciones para crear un grupo de seguridad (línea 18) para un balanceador de carga y la información relacionada al proveedor de servicios IaaS, tales como el código de la región (línea 14) donde se debe aprovisionar la infraestructura y el código de las claves de acceso (línea 16) para crear la infraestructura en Amazon Web Services.

```

1|--
2#####
3# ARGON: An Infrastructure Modeling Tool for Cloud Provisioning #
4# Project developed by Julio Sandobalín #
5# julio.sandobalín@epn.edu.ec #
6# Script tested with Ansible 2.6.4 and Amazon Web Services #
7# #####
8
9- hosts: localhost
10 connection: local
11 gather_facts: no
12 vars:
13 #your region
14 region: eu-west-1
15 #your key
16 key: kp-ireland
17 tasks:
18 - name: create security group sgpLoadBalancer
19 ec2_group:
20 region: "{{ region }}"
21 name: sgpLoadBalancer
22 description: Security group for load balancer
23 rules:
24 - proto: tcp
25 from_port: 80
26 to_port: 80
27 cidr_ip: 0.0.0.0/0

```

Figura 7-2 Extracto del script de aprovisionamiento para la herramienta Ansible

La siguiente tarea es utilizar el script con la herramienta Ansible para orquestar el aprovisionamiento de infraestructura en Amazon Web Services. La Figura 7-3 muestra un extracto de los mensajes de la consola de Ansible durante el proceso de aprovisionamiento de la infraestructura. En este caso, los mensajes describen la creación de cada elemento de infraestructura.

```

TASK [Print data of the load balancer loadBalancer] *****
ok: [localhost] => {
  "msg": "loadBalancer --> loadBalancer-1254521782.eu-west-1.elb.amazonaws.com"
}

TASK [Register instances virtualMachine from load balancer loadBalancer] *****
changed: [localhost -> localhost] => (item=i-06e5f80fd8f3a591d)
changed: [localhost -> localhost] => (item=i-0b238122e43d1fd3d)
changed: [localhost -> localhost] => (item=i-0d2f0b60d8f0ff982)

PLAY RECAP *****
localhost : ok=9 changed=5 unreachable=0 failed=0

```

Figura 7-3 Extracto de los mensajes de la consola de la herramienta Ansible

La última tarea es validar que la infraestructura y los paquetes de software funcionen correctamente. En este caso, se utiliza la dirección web de cada máquina virtual y del balanceador de carga que se muestran en la consola de

Ansible (ver Figura 7-3). Se copia y pega la dirección web en un navegador para visualizar una página web alojada en cada máquina virtual. La Figura 7-4 presenta la página web para verificar el funcionamiento de una máquina virtual. Adicionalmente, cada página web muestra la dirección IP privada de cada máquina virtual. Finalmente, se copia y pega la dirección web del balanceador de carga en un navegador para verificar su funcionamiento. Al recargar la página web, el balanceador de carga distribuye las peticiones —es decir, la carga de trabajo— entre las máquinas virtuales y en consecuencia cambiará la dirección IP interna que se visualiza en cada página web.



Figura 7-4 Verificación del funcionamiento de una máquina virtual

7.2 Experimentos controlados

Con el fin de validar MoCIP se realizaron dos experimentos controlados con los estudiantes del grado en Ingeniería Informática de la Universitat Politècnica de València en Valencia, España. Cada experimento se diseñó y ejecutó siguiendo las directrices del proceso experimental propuesto por (Wohlin *et al.*, 2012).

7.2.1 Objetivo

Se debe establecer un objetivo común para el conjunto de experimentos con el fin de permitir un análisis efectivo de los resultados individuales. Además, un objetivo permite definir el alcance de los experimentos. De acuerdo con el paradigma *Goal-Question-Metric* (Basili *et al.*, 1994) (GQM), el objetivo de los experimentos controlados es:

Analizar el aprovisionamiento de la infraestructura en Amazon Web Services utilizando MoCIP, con el propósito de evaluar la percepción de los usuarios, con respecto a la facilidad de uso percibida, utilidad percibida e intención de uso, desde el punto de vista de los ingenieros de software noveles, y en el contexto de los estudiantes del grado en Ingeniería Informática.

Aunque los participantes seleccionados deberían ser profesionales de la computación en la nube, los experimentos se centran en el perfil de los ingenieros de software noveles. En este caso, los ingenieros de software noveles son estudiantes de los últimos años de la carrera de Ingeniería Informática. Debido a que MoCIP propone un enfoque para el aprovisionamiento de infraestructura en la nube para ayudar a los usuarios menos experimentados, se seleccionó estudiantes como ingenieros de software noveles. Las preguntas de investigación (*Research Question*, RQ) para direccionar los experimentos son:

- **RQ1.** ¿MoCIP se percibe como fácil de usar?
- **RQ2.** ¿MoCIP se percibe como útil?
- **RQ3.** ¿Existe la intención de usar MoCIP en el futuro?

7.2.2 Selección del contexto

El contexto de estudio es el aprovisionamiento de infraestructura en Amazon Web Services por parte de ingenieros de software noveles. El contexto se define por los objetos experimentales, MoCIP y la selección de participantes.

7.2.2.1 Objetos experimentales

Los enunciados de los ejercicios, así como los requisitos para elicitar, modelar y aprovisionar la infraestructura en Amazon Web Services se seleccionaron y adaptaron de la literatura:

- O1—Empresa MODAFIN (Nitto *et al.*, 2017): el propósito es resolver el tráfico exponencial de solicitudes recibidas a aplicaciones de software instaladas en Amazon Web Services. Se debe utilizar un balanceador de carga para distribuir la carga de trabajo entre varias máquinas virtuales.
- O2—Universidad CEC (Sandobalín *et al.*, 2017a): el objetivo es pagar por el uso de la infraestructura aprovisionada en Amazon Web Services para resolver la alta demanda de cursos específicos que causan una sobrecarga en la plataforma virtual (servidores) ubicada en la Universidad CEC. En este caso, un balanceado de carga debe distribuir la carga de trabajo entre las máquinas virtuales, que se crearán o destruirán bajo demanda.

7.2.2.2 *MoCIP y su soporte de herramientas*

MoCIP propone un enfoque para ayudar a los usuarios menos experimentados en las actividades de aprovisionamiento de infraestructura en la nube. Primero, se propone un conjunto de tareas para identificar los elementos de la infraestructura y sus atributos con el fin de seleccionar la arquitectura de la infraestructura a modelar.

La actividad de modelado se realiza con el lenguaje de modelado ArgonML. Luego, con el motor de transformaciones M2T de la herramienta ARGON se generan los scripts de aprovisionamiento. La herramienta Ansible utiliza los scripts para realizar el aprovisionamiento de infraestructura en Amazon Web Services. Finalmente, se debe verificar que la infraestructura y los paquetes de software estén creados, instalados y en funcionamiento.

7.2.2.3 *Selección de participantes*

De acuerdo con (Kitchenham *et al.*, 2002), los estudiantes son la próxima generación de profesionales del software y, por lo tanto, están relativamente cerca de la población de interés. Adicionalmente, (Höst *et al.*, 2000) investigaron la idoneidad de los estudiantes de los últimos años de universidad como participantes y concluyeron que si están bien entrenados, pueden considerarse como sujetos experimentales apropiados. En consecuencia, se seleccionó el siguiente grupo de participantes:

- Estudiantes de matriculados en el grado de Ingeniería Informática de la Universitat Politècnica de València. Los estudiantes asistieron al curso de Desarrollo de Software Dirigido por Modelos a lo largo del curso académico 2018-19. El propósito del curso es proporcionar el conocimiento sobre la construcción de modelos de software en diferentes niveles de abstracción como principales artefactos en el proceso de desarrollo de software.

No se estableció una clasificación de los estudiantes en función de su experiencia en el aprovisionamiento de infraestructura en la nube, debido principalmente a que los estudiantes no tenían conocimientos previos de la computación en la nube.

7.2.3 *Diseño de los experimentos individuales*

La Figura 7-5 presenta un resumen de los experimentos controlados para validar MoCIP. Además, se muestra el contexto de cada experimento, el número de participantes de cada experimento, el lugar donde se realizaron y el orden de ejecución de los experimentos.



Figura 7-5 Resumen de los experimentos controlados

Antes de ejecutar los experimentos controlados, se realizó un experimento piloto con 9 estudiantes de doctorado. En este caso, el objetivo del experimento piloto fue validar los materiales, procedimientos, instrucciones y tiempo de finalización de cada tarea experimental. Es importante mencionar que los estudiantes de doctorado no participaron en los experimentos controlados.

Primero se realizó un experimento de línea base (*UPV1*) con 38 estudiantes del grado en Ingeniería Informática. Con el fin de verificar los resultados obtenidos en el experimento de línea base, se realizó una réplica estricta (*UPV2*) con 36 estudiantes del grado en Ingeniería Informática. Se utilizaron las directrices propuestas en (Carver, 2010) para realizar la réplica estricta del experimento de línea base. La réplica fue operativa porque se cambió la población en cada experimento. Este cambio permitió verificar si los resultados son independientes del perfil de los participantes.

7.2.3.1 *Diseño del experimento de línea base*

El objetivo del experimento de línea base es evaluar la percepción de los participantes luego de utilizar MoCIP en el aprovisionamiento de infraestructura en Amazon Web Services.

7.2.3.1.1 *Selección del contexto*

Se utilizan los objetos experimentales presentados en la Sección 7.2.2.1 junto con el enfoque MoCIP presentado en la Sección 7.2.2.2. Los requisitos de los objetos experimentales fueron tomados y adaptados de la literatura científica (Nitto *et al.*, 2017)(Sandobalín *et al.*, 2017a). Los objetos experimentales son de dominios diferentes y no requieren de conocimientos especializados para comprenderlos, pero tienen una complejidad similar.

Tabla 7-1 Requisitos para aprovisionar la infraestructura en Amazon Web Services

No.	Descripción del requisito
R1	La infraestructura debe implementarse en una región específica de Amazon Web Services.
R2	Un grupo de seguridad debe habilitar conexiones entrantes TCP a máquinas virtuales a través de tres puertos específicos. Además, el grupo de seguridad debe habilitar todas las conexiones salientes desde las máquinas virtuales.
R3	Un grupo de seguridad debe habilitar las conexiones TCP entrantes a un balanceador de carga a través del puerto 80. Además, el grupo de seguridad debe habilitar todas las conexiones salientes desde el balanceador de carga.
R4	Se debe ejecutar un conjunto de máquinas virtuales en una zona de disponibilidad específica de una región seleccionada de Amazon Web Services. Cada máquina virtual debe tener una CPU y RAM explícitas. Además, cada máquina virtual debe tener un sistema operativo específico y un servidor web instalado.
R5	Un balanceador de carga debe distribuir la carga de trabajo entre las máquinas virtuales. El balanceador de carga debe responder a las solicitudes de los clientes y validar que todas las máquinas virtuales conectadas estén disponibles. Además, el equilibrador de carga debe funcionar en una zona de disponibilidad específica de una región seleccionada de Amazon Web Services.
R6	El balanceador de carga utiliza un elemento de comprobación de estado para validar el estado de las máquinas virtuales conectadas al balanceador de carga. En este caso, se emplea intervalos de comprobación utilizando el protocolo TCP y un número de puerto específico. Es necesario esperar un tiempo en segundos para notificar una verificación de error. El balanceador de carga debe recibir una cantidad precisa de errores consecutivos para pasar a un estado "no saludable" a una máquina virtual, mientras que debe recibir un número específico de éxitos de la sonda de verificación de estado para cambiar a un estado "saludable" a una máquina virtual.
R7	El balanceador de carga utiliza un elemento de escucha para responder a las solicitudes de los clientes a través del protocolo TCP y del puerto 80, así como para distribuir toda la carga de trabajo a las máquinas virtuales mediante un puerto determinado.
R8	Las máquinas virtuales deben registrarse en el balanceador de carga, de modo que el balanceador de carga pueda distribuir la carga de trabajo entre todas las máquinas virtuales disponibles.

La Tabla 7-1 presenta los requisitos para el aprovisionamiento de infraestructura en Amazon Web Services. Es importante mencionar que se presentan los

requisitos de manera general. Sin embargo, cada objeto experimental tiene sus propios atributos y configuraciones de los recursos de infraestructura.

7.2.3.1.2 *Participantes*

El experimento de línea base involucró a 38 estudiantes de matriculados en el grado de Ingeniería Informática de la Universitat Politècnica de València. Los participantes asistieron al curso de otoño de 2019 sobre Ingeniería de Software Dirigida por Modelos, en donde obtuvieron conocimientos sobre técnicas para el desarrollo de software dirigidas por modelos. Para el ciclo lectivo 2018-2019 existió dos cursos de la asignatura, un curso por la mañana y otro por la tarde. El experimento de línea base se ejecutó con los estudiantes del curso de la mañana. Se pidió a los participantes que realizaran las tareas experimentales como parte de los ejercicios de laboratorio del curso.

7.2.3.1.3 *Selección de variables*

La variable o factor independiente es el aprovisionamiento de la infraestructura en Amazon Web Services. En este caso, la variable independiente tiene un como único nivel o tratamiento a MoCIP.

Se utiliza el modelo de evaluación del método (*Method Evaluation Model*, MEM) (Moody, 2003) como base teórica para diseñar el experimento de línea base. De acuerdo con este modelo, hay dos tipos de variables dependientes: variables basadas en el rendimiento (*performance-based variables*) que miden qué tan bien los participantes realizan las tareas experimentales y variables basadas en la percepción (*perception-based variables*) que miden las creencias y actitudes de los participantes al utilizar MoCIP. En este caso, es importante mencionar que las variables basadas en el rendimiento se centran en las actividades de modelado y aprovisionamiento de infraestructura en la nube, mientras que las variables basadas en la percepción se centran en la percepción de los participantes luego de utilizar MoCIP.

Existen dos variables basadas en el rendimiento:

- **Efectividad**, mide el grado en qué cada actividad de MoCIP logra sus objetivos.
- **Eficiencia**, mide el esfuerzo requerido para implementar cada una de las actividades de MoCIP.

Existe tres variables basadas en la percepción:

- **Facilidad de uso percibida** (*Perceived Ease of Use*, PEOU), se refiere al grado en qué un participante cree que aprender y utilizar MoCIP sería fácil.

- **Utilidad percibida** (*Perceived Usefulness*, PU), se refiere al grado en qué un participante cree que MoCIP será efectivo para lograr sus objetivos.
- **Intención de uso** (*Intention to Use*, ITU), es la medida en que un participante intenta utilizar MoCIP.

Para operacionalizar las variables basadas en el rendimiento, se utiliza la ISO/IEC 9126-4 (ISO/IEC, 1991) para obtener las métricas para medir la efectividad y la eficiencia. En cambio, para operacionalizar las variables basadas en la percepción, se adapta el instrumento de medición propuesto por (Davis *et al.*, 1989) para medir la facilidad de uso percibida, la utilidad percibida y la intención de uso. La Tabla 7-2 resume las métricas utilizadas para medir cada variable dependiente.

Tabla 7-2 Resumen de variables dependientes

Nombre	Medida	Escala
Efectividad	$\frac{\# \text{ Requistos correctos}}{\# \text{ Total de requisitos}}$	Ratio
Eficiencia	$\frac{\text{Efectividad}}{\text{Tiempo}}$	Ratio
PEOU	Escala Likert de 5 puntos	Ordinal
PU	Escala Likert de 5 puntos	Ordinal
ITU	Escala Likert de 5 puntos	Ordinal

Para la actividad de modelado de infraestructura, se modela un conjunto de requisitos con el lenguaje ArgonML y se calcula la efectividad como el número de requisitos definidos correctamente dividido por el número total de requisitos propuestos. En cambio, para la actividad de aprovisionamiento, la efectividad se calcula como el aprovisionamiento satisfactorio de la infraestructura en Amazon Web Services. Es importante mencionar que no se calcula la efectividad de la elicitación de requisitos debido principalmente a que los requisitos están estrechamente relacionados con la actividad de modelado. Además, se evitó que el experimento se ejecute en un largo período de tiempo para evadir el cansancio y/o aburrimiento de los participantes.

Tabla 7-3 Ítems para medir las variables basadas en la percepción

Tipo	Descripción del Ítem
PEOU1	Encontré el procedimiento para usar MoCIP complejo y difícil de seguir.
PEOU2	En general, MoCIP me pareció difícil de usar.
PEOU3	Encontré MoCIP fácil de aprender.
PEOU4	Me resultó difícil aprovisionar la infraestructura en la nube usando MoCIP.
PEOU5	El uso de MoCIP me pareció claro y fácil de entender.
PU1	Creo que MoCIP reduciría el esfuerzo requerido para aprovisionar la infraestructura en la nube.
PU2	En general, MoCIP me pareció útil.
PU3	El aprovisionamiento de infraestructura en la nube utilizando MoCIP sería difícil de entender.
PU4	En general, creo que MoCIP no proporciona una solución efectiva para el aprovisionamiento de infraestructura en la nube.
PU5	En general, creo que MoCIP es una mejora en el aprovisionamiento de infraestructura en la nube.
PU6	En general, creo que MoCIP facilitaría a los profesionales el aprovisionamiento de infraestructura en la nube.
PU7	Creo que MoCIP facilitaría la comunicación entre profesionales para lograr un aprovisionamiento de infraestructura en la nube.
ITU1	Recomendaría MoCIP para aprovisionar la infraestructura en la nube.
ITU2	Si en el futuro estoy trabajando en una empresa, me gustaría usar MoCIP para aprovisionar la infraestructura en la nube.
ITU3	Sería fácil para mí ser hábil al usar MoCIP para aprovisionar la infraestructura en la nube.

Por otro lado, la eficiencia se calculó como la efectividad de cada actividad dividida por el tiempo que el/la participante dedicó a realizar dicha actividad. La eficiencia aumenta al aumentar la efectividad y reducir el tiempo de cada actividad. En este contexto, se define la siguiente métrica de agregación para decidir si una actividad es válida o no:

- **Métrica de todo o nada:** se considera que una tarea experimental es correcta solo si se alcanza su objetivo satisfactoriamente. Por un lado, modelar un requisito tiene solo dos valores posibles: éxito o fracaso (1 o 0). Por otro lado, aprovisionar la infraestructura en la nube tiene solo dos valores posibles: éxito o fracaso (1 o 0).

Se propone la *métrica de todo o nada* porque los recursos de infraestructura modelados con ArgonML deben funcionar en Amazon Web Services. En el caso que exista un requisito mal definido, la infraestructura no se aprovisionará.

La Tabla 7-3 presenta los ítems para medir las variables basadas en percepción. En este caso, se utiliza el cuestionario adaptado de (Moody, 2003) para medir PEOU, PU e ITU. Los ítems del cuestionario fueron formulados utilizando una escala de Likert de cinco puntos y adoptando el formato de preguntas de enunciado opuesto. Además, se ordenó de manera aleatoria los ítems dentro del grupo de preguntas de cada variable para prevenir el sesgo de respuesta sistémica.

7.2.3.1.4 Formulación de hipótesis

Las hipótesis nulas se formulan en base a las variables dependientes. Es importante mencionar que el experimento tiene como objetivo evaluar la percepción de los participantes después de utilizar MoCIP. Las hipótesis nulas del experimento se pueden resumir de la siguiente manera:

- $H1_0$: MoCIP se percibe como difícil de usar. $H1_1 = \neg H1_0$
- $H2_0$: MoCIP se percibe como no útil. $H2_1 = \neg H2_0$
- $H3_0$: No hay intención de usar MoCIP en el futuro. $H3_1 = \neg H3_0$

El objetivo del análisis estadístico es rechazar las hipótesis nulas y posiblemente aceptar las hipótesis alternativas, por ejemplo, $H1_1 = \neg H1_0$.

7.2.3.1.5 Diseño

El diseño del experimento se elige en función de las hipótesis y las variables seleccionadas. En este caso, la variable independiente es el aprovisionamiento de infraestructura en Amazon Web Services y tiene como único tratamiento a MoCIP. En cambio, las hipótesis deben evaluar la percepción de los participantes luego de usar MoCIP. Adicionalmente, es importante mencionar que actualmente no existe un método estándar para aprovisionar la infraestructura en la nube y, por lo tanto, no es posible evaluar MoCIP contra un método de control.

La Tabla 7-4 presenta el diseño del experimento de línea base. Debido a que existe dos objetos experimentales, se dividió a los participantes en dos grupos.

Tabla 7-4 Diseño del experimento de línea base

Tratamiento	MoCIP	
	Período 1	
Objeto	O1: MODAFIN	O2: CEC
Grupo 1	X	–
Grupo 2	–	X

Los tratamientos se aplican a la combinación de objetos experimentales y sujetos (Wohlin *et al.*, 2012). Adicionalmente, un período se define mediante la aplicación de un tratamiento por un participante a un objeto experimental, mientras que una sesión es una parte del tiempo que un sujeto dedica a completar (una o más) tareas experimentales (Vegas *et al.*, 2016). Como resultado, se utilizó un período (*Período 1*) para que el *Grupo 1* utilice MoCIP con el objeto experimento *O1*, mientras que el *Grupo 2* utilizó MoCIP con el objeto experimental *O2*.

Al diseñar un experimento, se deben considerar muchos aspectos, tales como aleatorización, bloqueo y el balanceo (Wohlin *et al.*, 2012). Los participantes del experimento de línea base fueron seleccionados por conveniencia, mientras que la asignación de los objetos experimentales (*O1* y *O2*) a los participantes fue aleatoria. Finalmente, a cada grupo (*Grupo 1* y *Grupo 2*) se le asignó un número igual de participantes, con el fin de obtener un diseño balanceado.

7.2.3.1.6 Operación

El experimento de línea base se realizó en tres sesiones de dos horas. La Figura 7-6 presenta un resumen del proceso para realizar el experimento controlado. Los números en el diagrama significan pasos en el proceso experimental y, en este caso, representan las sesiones.

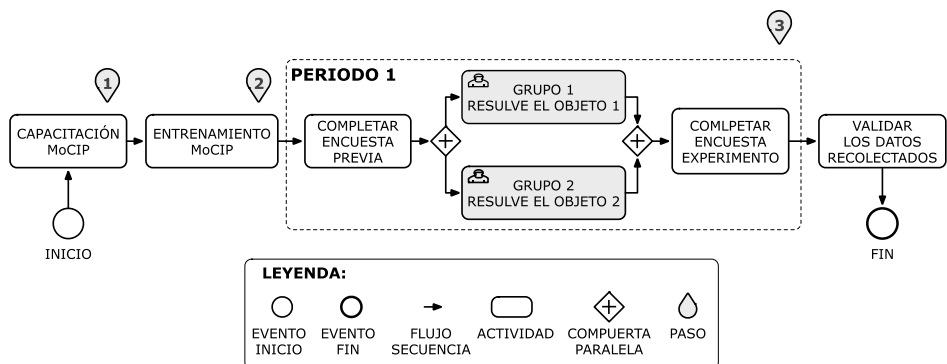


Figura 7-6 Proceso de operación del experimento de línea base

En la primera sesión de dos horas (*Paso 1*) se realizó la capacitación de los fundamentos de la computación en la nube. Además, se explicó el enfoque MoCIP para aprovisionar la infraestructura en la nube. Es importante mencionar que la capacitación se enfocó en explicar los conceptos fundamentales que los participantes tuvieron que utilizar en el experimento. En la segunda sesión de dos horas (*Paso 2*) los participantes se familiarizaron con MoCIP. En la segunda sesión, se realizó el entrenamiento de MoCIP y los participantes realizaron un ejercicio similar al experimento controlado.

El período (*Período 1*) del experimento controlado se realizó en la tercera sesión de dos horas (*Paso 3*). A continuación, se describen las actividades realizadas durante el período:

- Primero, los participantes completaron una encuesta relacionada con sus conocimientos y experiencia en la computación en la nube, así como del proceso de aprovisionar la infraestructura en la nube.
- Después, los participantes fueron asignados aleatoriamente al *Grupo 1* y al *Grupo 2*, considerando que ambos grupos tendrán el mismo número de participantes. Para el *Período 1*, el *Grupo 1* resolvió el objeto experimental *O1—MODAFIN*, mientras que el *Grupo 2* resolvió el objeto experimental *O2—CEC*. En cada grupo, los participantes deben utilizar las actividades de MoCIP para elicitar lo requisitos, modelar y aprovisionar la infraestructura en Amazon Web Services.
- Finalmente, los participantes deben completar una encuesta para expresar su percepción sobre la facilidad de uso, la utilidad y la intención de usar MoCIP en el futuro.

Después del *Período 1* se verificó que los datos —del experimento controlado— están completos y que se recolectaron correctamente.

7.2.3.2 Segundo experimento (UPV2)

El segundo experimento es una réplica interna estricta del experimento de línea base (*UPV1*). Se aplicó el mismo protocolo experimental, pero a una población diferente. En este caso, los participantes fueron distintos del experimento de línea base, mientras que el sitio, los experimentadores, el diseño, las variables y la instrumentación permanecieron igual. Se cambió a los participantes con el propósito de evaluar hasta qué punto los resultados experimentales podrían generalizarse. La muestra de participantes estaba compuesta por 36 estudiantes matriculados en el curso de Desarrollo de Software Dirigido por Modelos y, por lo tanto, obtuvieron conocimiento en las técnicas de desarrollo de software dirigidas por modelos. Para el ciclo lectivo 2018-2019 existió dos cursos de la asignatura, un curso por la mañana y otro por la tarde. La réplica del experimento

se ejecutó con los estudiantes del curso de la tarde. Se pidió a los participantes que realizaran las tareas experimentales como parte de los ejercicios de laboratorio del curso.

Al igual que en el experimento de línea de base, la réplica tuvo lugar en una habitación individual y no se permitió la interacción entre los participantes. Se utilizó el diseño del experimento de línea base.

7.2.4 Tareas experimentales y materiales

En este caso, las tareas experimentales que los participantes deben realizar para validar MoCIP son: elicitar los requisitos, modelar la infraestructura y aprovisionar la infraestructura en Amazon Web Services. Es importante mencionar que las tareas experimentales se corresponden con las actividades propuestas para MoCIP en la Sección 5.3.

Se entregó a cada participante un objeto experimental ($O1$, $O2$) que consta de un conjunto de requisitos para ejemplificar un problema real. En este caso, cada participante debe elicitar, modelar y aprovisionar un balanceador de carga que distribuye la carga de trabajo entre varias máquinas virtuales. A continuación, se describen las tareas experimentales:

- Identificar los elementos de la infraestructura y sus atributos con el fin de seleccionar la arquitectura de la infraestructura.
- Modelar un diagrama de infraestructura con el lenguaje de modelado ArgonML. En este caso, ArgonML permite modelar directamente los elementos de infraestructura para Amazon Web Services. Sin embargo, para obtener modelos de infraestructura para diferentes proveedores de servicios en la nube, se utiliza el motor de transformaciones de M2M de la herramienta ARGON.
- A partir del modelo de infraestructura de Amazon, se genera los scripts de aprovisionamiento. En este caso, se utiliza el motor de transformaciones de M2T de la herramienta ARGON.
- La herramienta Ansible utiliza los scripts generados para orquestar el aprovisionamiento de infraestructura en Amazon Web Services.
- Se debe verificar que las aplicaciones de software instaladas en la infraestructura —es decir, máquinas virtuales— estén instaladas, habilitadas y funcionando.

El material de capacitación incluyó: (1) diapositivas para enseñar los fundamentos de la computación en la nube, Amazon Web Services y la Infraestructura como Código; (2) diapositivas para explicar MoCIP junto con un ejemplo ilustrativo sobre cómo elicitar, modelar y aprovisionar la infraestructura

en Amazon Web Services. El material experimental que respalda las tareas del experimento incluye:

- Dos folletos con cada objeto experimental (*O1* y *O2*). El propósito de estos folletos es describir cada objeto experimental junto con sus requisitos.
- Un cuestionario previo para recopilar los conocimientos y habilidades de los participantes, y un cuestionario posterior para recopilar las percepciones de los participantes sobre la facilidad de uso, la utilidad y la intención de uso.
- Una guía que explica los pasos para modelar los recursos de infraestructura con el lenguaje ArgonML. La guía incluye instrucciones para crear un proyecto de infraestructura. Además, se explica cómo usar los elementos de infraestructura y completar sus propiedades.
- Una guía que explica las regiones y zonas de disponibilidad para Amazon Web Services. El balanceador de carga y las máquinas virtuales deben especificar el lugar donde se aprovisionarán.
- Una guía que contiene una lista con las claves para acceder a las regiones de Amazon Web Services. Para aprovisionar los recursos de infraestructura en una región específica, es necesario utilizar su código de acceso.
- Una guía que contiene una lista de los códigos de los tipos de instancia. En este caso, el código especifica las características de hardware de las máquinas virtuales.
- Una guía que contiene una lista de los códigos de imagen. En este caso, cada código de imagen especifica un sistema operativo para una máquina virtual en Amazon Web Services.
- Una máquina virtual configurada y probada con Ansible y ARGON. El objetivo es proporcionar a cada participante un entorno idéntico para realizar el experimento controlado. Cada máquina virtual se configuró con Windows Server 2012R2, Java JDK v1.8, Eclipse Modeling Framework v4.8, ARGON v1.0 y Ansible v2.6.

El cuestionario posterior al experimento contenía un conjunto de preguntas cerradas que permiten a los participantes expresar sus opiniones sobre MoCIP en términos de su facilidad de uso percibida, utilidad percibida e intención de usar MoCIP el futuro.

7.3 Resultados

En esta sección, se presenta la evidencia empírica recolectada al realizar los dos experimentos controlados, así como un análisis de la estadística descriptiva e inferencial. Se utiliza el paquete estadístico para ciencias sociales SPSS v24 para realizar los análisis estadísticos de los datos recolectados en los experimentos.

7.3.1 Estadística descriptiva y análisis de datos exploratorios

La Tabla 7-5 presenta un resumen de las estadísticas descriptivas para las variables basadas en rendimiento y en percepción. En este caso, se presenta la media y la desviación estándar (*Standard Deviation*, SD) como los estadísticos de cada variable dependiente. Aunque no se realiza un análisis a profundidad de la *Duración*, se incluye la variable para dar una primera idea sobre la complejidad de las tareas experimentales. La *Duración* es el tiempo que el/la participante dedica a realizar una tarea experimental.

Tabla 7-5 Resumen de estadísticas descriptivas.

Variables	UPV1		UPV2	
	Media	SD	Media	SD
Efectividad Modelado	0.855	0.118	0.854	0.188
Eficiencia Modelado	0.030	0.008	0.027	0.009
Duración Modelado (min)	29.763	4.588	33.583	7.303
Efectividad Aprovisionamiento	0.711	0.460	0.667	0.478
Eficiencia Aprovisionamiento	0.037	0.032	0.042	0.034
Duración Aprovisionamiento (min)	22.605	9.977	20.500	9.614
PEOU	4.316	0.462	4.311	0.518
PU	4.282	0.444	4.135	0.421
ITU	4.132	0.545	4.009	0.640

La Tabla 7-5 ayuda a comprender las características de los participantes al dar un resumen de los datos para cada muestra. Los participantes presentan una efectividad similar al modelar la infraestructura. La efectividad de modelado de la muestra UPV1 es 0.855 (85%), mientras la muestra UPV2 tiene una efectividad de modelado de 0.854 (85%). En cambio, la efectividad del aprovisionamiento de la infraestructura en Amazon Web Services tiene valores muy cercanos. La efectividad de aprovisionamiento de la muestra UPV1 es 0.711 (71%), mientras que la muestra UPV2 tiene una efectividad de aprovisionamiento de 0.667 (67%). En cuanto al tiempo empleado para realizar

las tareas experimentales, las dos muestras tienen una duración similar en cada tarea experimental. Por un lado, la muestra UPV1 presenta un tiempo promedio de 29 minutos para el modelado de infraestructura, mientras que la muestra UPV2 tiene un tiempo promedio de 33 minutos. Por otro lado, la muestra UPV1 presenta un tiempo promedio de 22 minutos para el aprovisionamiento de infraestructura en Amazon Web Services, mientras que la muestra UPV2 tiene un tiempo promedio de 20 minutos.

Los resultados de las variables basadas en la percepción (PEOU, PU, ITU) del experimento de línea base (UPV1) y su réplica (UPV2) muestran que los participantes percibieron que MoCIP es fácil de usar y útil. Además, los participantes expresaron su intención de usar MoCIP en el futuro. Es importante mencionar que para medir las variables basadas en la percepción se utilizó una escala de Likert de 5 puntos y para su evaluación se utilizó la puntuación neutral, es decir, puntuación 3 (Abrahão *et al.*, 2011).

Tabla 7-6 Resultados del aprovisionamiento de infraestructura en Amazon Web Services

	Aprovisionamiento de la Infraestructura			
	SI		NO	
	Cantidad	Porcentaje	Cantidad	Porcentaje
UPV1	27	71.1%	11	28.9%
UPV2	24	66.7%	12	33.3%

La Tabla 7-6 presenta un resumen de los resultados del aprovisionamiento de infraestructura en Amazon Web Services. Es importante mencionar que, aunque los participantes tengan un modelo de infraestructura definido correctamente, en ocasiones no utilizaban la nomenclatura establecida para nombrar los elementos de infraestructura. Como resultado, al intentar crear un elemento de infraestructura ocurría un error porque Amazon Web Services no permite la creación de dos elementos con nombres idénticos. Para el caso de la muestra UPV1 el 71.1% de participantes aprovisionaron la infraestructura satisfactoriamente, mientras que el 28.9% tuvieron algún tipo de error. En cambio, para la muestra UPV2 el 66.7% de participantes aprovisionaron la infraestructura correctamente, mientras que el 33.3% tuvieron algún tipo de problema.

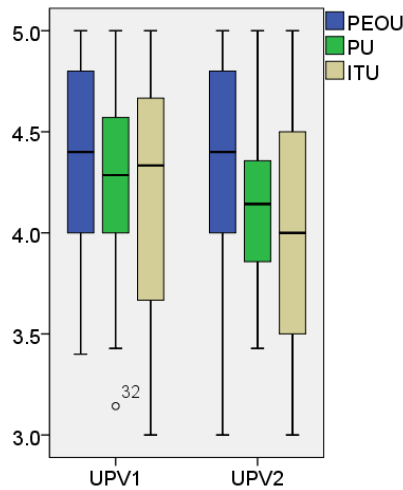


Figura 7-7 Diagrama de caja para las variables basadas en la percepción

La Figura 7-7 muestra el diagrama de caja para las variables basadas en la percepción de los experimentos. Se puede observar un contraste entre los resultados de las variables PEOU, PU e ITU del experimento de línea base (UPV1) y de su réplica (UPV2). En este caso, los valores medios de cada variable tienen un valor superior a la puntuación neutral de 3. Incluso el valor del primer cuartil (Q1) de las variables tiene un valor superior a 3.

Como resultado, los participantes percibieron que MoCIP es fácil de usar y útil. Además, los participantes tienen la intención de utilizar MoCIP en el futuro.

7.3.2 Prueba de hipótesis

Una vez realizado el experimento de línea base (UPV1) y su réplica (UPV2) se procede a realizar las pruebas de hipótesis para evaluar la percepción de los participantes luego de utilizar MoCIP. Para obtener la percepción de los sujetos se utilizó y adaptó el cuestionario propuesto por (Davis *et al.*, 1989). En este caso, primero se debe evaluar la fiabilidad del cuestionario. Para evaluar la fiabilidad del cuestionario se utiliza la Prueba Alfa de Cronbach (α). La fiabilidad para todo el cuestionario de la muestra UPV1 es 0.896, mientras que para la muestra UPV2 es 0.883. Los resultados son más altos que el umbral 0.70 (Maxwell, 2002).

La Tabla 7-7 muestra los resultados de las variables basadas en la percepción donde la muestra UPV1 tiene los resultados PEOU=0.742, PU=0.809 y ITU=0.743, mientras que la muestra UPV2 tiene los resultados PEOU=0.772, PU=0.756 y ITU=0.740. En consecuencia, las variables basadas en la percepción tienen un valor de fiabilidad (α) superior al umbral 0.70.

Tabla 7-7 Resumen de estadísticas inferencial

	Variable	Ítems	α (valor-p)	Normalidad (valor-p)	Sig. (valor-p)
UPV1 n = 38	PEOU	5	0.742	0.005	<0.001‡
	PU	7	0.809	0.331	<0.001*
	ITU	3	0.743	0.021	<0.001‡
UPV2 n = 36	PEOU	5	0.772	0.008	<0.001‡
	PU	7	0.756	0.274	<0.001*
	ITU	3	0.740	0.029	<0.001‡

‡ Prueba de los rangos con signo de Wilcoxon. * Prueba T de Student.

Se aplicó una Prueba de Normalidad a cada variable para conocer que prueba estadística utilizar en el contraste de hipótesis. Para evaluar si los datos se distribuyen normalmente o no, se utilizó la Prueba de Shapiro-Wilk porque el número de sujetos (*n*) en cada experimento es inferior a 50 (Maxwell, 2002).

La Tabla 7-7 presenta los resultados de la normalidad de cada variable. En este caso, cuando los datos de distribuyen normalmente —la Prueba de Shapiro-Wilk tiene un valor $p \geq 0,05$ — se utiliza la *Prueba T de Student* de una muestra. En cambio, cuando los datos no presentan una distribución normal, se utiliza la *Prueba de los rangos con signo de Wilcoxon* de una muestra.

La Tabla 7-7 muestra en su columna *Sig. (valor-p)* que todas las variables basadas en la percepción para las muestras UPV1 y UPV2 tienen un valor $p < 0,001$.

7.3.2.1 Respondiendo las preguntas de investigación

Con respecto a las preguntas de investigación (*Research Question*, RQ) que motivaron los experimentos controlados, se responden las preguntas utilizando los resultados empíricos para respaldar las afirmaciones.

RQ1. ¿MoCIP se percibe como fácil de usar?

Se encontró evidencia empírica para afirmar que los participantes percibieron que MoCIP es fácil de usar para realizar el aprovisionamiento de infraestructura en Amazon Web Services. Este resultado puede deberse a que los participantes percibieron que el aprendizaje y el uso de un enfoque dirigido por modelos (MoCIP) fue fácil.

En consecuencia, los experimentos son estadísticamente significativos en términos de PEOU para rechazar las hipótesis nulas $H1_0$.

RQ2. ¿MoCIP se percibe como útil?

Se encontró evidencia empírica para afirmar que los participantes percibieron que MoCIP es útil para realizar el aprovisionamiento de infraestructura en Amazon Web Services. Este resultado podría deberse a que los participantes percibieron que MoCIP fue eficaz para lograr su objetivo, es decir, aprovisionar la infraestructura en la nube.

Por lo tanto, los experimentos son estadísticamente significativos en términos de PU para rechazar las hipótesis nulas H_{2_0} .

RQ3. ¿Existe la intención de usar MoCIP en el futuro?

Se encontró evidencia empírica para afirmar que los participantes tienen la intención de usar MoCIP en el futuro. Este resultado puede deberse a que los participantes completaron la tarea experimental y se sintieron cómodos al usar MoCIP para aprovisionar la infraestructura en Amazon Web Services.

Como resultado, todos los experimentos son estadísticamente significativos en términos de ITU para rechazar las hipótesis nulas H_{3_0} .

7.4 Amenazas a la validez del experimento

En esta sección, se utilizan las recomendaciones propuestas por (Wohlin *et al.*, 2012) para revisar los posibles problemas que podría haber amenazado a la validez de los experimentos.

7.4.1 Validez de la conclusión

Las amenazas a la validez de la conclusión se refieren a los problemas que afectan la capacidad de obtener una conclusión correcta (Wohlin *et al.*, 2012). En este contexto, para lograr la confiabilidad de la implementación del tratamiento, se entregó a los participantes una máquina virtual configurada con todas las herramientas necesarias para ejecutar las tareas experimentales. Es importante mencionar que las máquinas virtuales se configuraron con una versión idéntica del sistema operativo, herramientas y espacio de trabajo. Además, con el fin de evitar las interrupciones en el entorno experimental, se utilizó un laboratorio como espacio físico controlado para evitar las distracciones o interrupciones.

Para disminuir las amenazas en la recopilación de los datos, se aplicó los mismos procedimientos de extracción de datos a cada experimento y se verificó que cada variable dependiente se calcule de manera consistente.

7.4.2 Validez interna

Las amenazas a la validez interna son influencias que pueden afectar la variable independiente con respecto a la causalidad, sin el conocimiento del investigador (Wohlin *et al.*, 2012). En este escenario, no existieron diferencias respecto al conocimiento y experiencia de los participantes, porque ningún participante tenía experiencia previa en computación en la nube. Además, se evitó el intercambio de información mediante el uso de diferentes objetos experimentales y monitoreando a los participantes durante los experimentos. Finalmente, para evaluar la comprensibilidad de los materiales, se realizó un experimento piloto para descubrir errores y corregirlos.

7.4.3 Validez del constructo

La validez del constructo se refiere a generalizar el resultado del experimento teniendo en cuenta el diseño del experimento y su capacidad para reflejar los constructos a estudiar (Wohlin *et al.*, 2012). En este caso, las medidas utilizadas para obtener las variables cualitativas y cuantitativas pueden influir en la validez del constructo. Se mitigó la amenaza mediante el uso de medidas que se aplican comúnmente en otros estudios empíricos de ingeniería de software. Por un lado, se utilizó las variables basadas en el rendimiento, tales como efectividad y eficiencia de acuerdo al estándar ISO/IEC 9126-4 (ISO/IEC, 1991) para evaluar las tareas experimentales. Además, se utiliza un conjunto de ocho requisitos para implementar el tratamiento. Por otro lado, se utilizó un cuestionario para medir las variables basadas en la percepción en términos de facilidad de uso percibida (PEOU), utilidad percibida (PU) e intención de uso (ITU) del tratamiento. Los constructos PEOU, PU e ITU se utilizan ampliamente para medir la percepción del participante y se basan en el modelo de aceptación de tecnología (*Technology Acceptance Model*, TAM) (Davis, 1989). La fiabilidad del cuestionario se evaluó mediante la Prueba Alfa de Cronbach. La Tabla 7-7 muestra que el coeficiente alfa de Cronbach para todas las variables fue más alto que el nivel umbral 0.70 (Maxwell, 2002) .

Finalmente, para evitar la preocupación de la evaluación, los participantes no fueron calificados según los resultados que obtuvieron. Sin embargo, los participantes ganaron un punto extra por participar en el experimento. Además, los participantes no estaban al tanto de las hipótesis experimentales para prevenir el sesgo en el tratamiento realizado.

7.4.4 Validez externa

Las amenazas a la validez externa son condiciones que limitan la capacidad de generalizar los resultados del experimento a la práctica industrial (Wohlin *et al.*, 2012). La primera preocupación fue seleccionar grupos de participantes que sean

representativos de la población objetivo, es decir, desarrolladores de software y personal de operaciones. Sin embargo, los participantes elegidos fueron estudiantes que no conocían la computación en la nube o las herramientas de aprovisionamiento. En este caso, los estudiantes son la próxima generación de profesionales del software y, por lo tanto, están relativamente cerca de la población objetivo (Kitchenham *et al.*, 2002). En consecuencia, los participantes seleccionados fueron ingenieros de software noveles.

El tamaño y la complejidad de los objetos experimentales es una amenaza que podría afectar la validez externa. En este escenario, se utilizó tareas experimentales con una complejidad moderada porque el experimento requiere que los participantes completen la tarea asignada en un tiempo limitado (sesiones de 2 horas). En consecuencia, las tareas realizadas por los participantes pueden considerarse como representativas en un entorno práctico.

7.5 Conclusiones

Los dos experimentos controlados aportan evidencia empírica sobre como MoCIP apoya el aprovisionamiento de infraestructura en la nube utilizando la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering*, MDE) y la Infraestructura como Código (*Infrastructure as Code*, IaC). Además, MoCIP utiliza la herramienta ARGON para abstraer y automatizar el modelado y aprovisionamiento de infraestructura en Amazon Web Services.

La evidencia empírica de los dos experimentos es una contribución al conjunto de conocimiento sobre IaC y MDE. En particular, esta evidencia empírica respalda la afirmación que técnicas MDE pueden *abstraer* la definición de los recursos de infraestructura y *automatizar* la generación de artefactos que soportan el aprovisionamiento de infraestructura en la nube. Por otro lado, los participantes aportaron evidencia empírica para afirmar que —bajo ciertas condiciones— MoCIP es fácil de usar y útil para elicitar, modelar y aprovisionar recursos de infraestructura en la nube. Adicionalmente, los participantes tienen la intención de utilizar MoCIP en el futuro. Sin embargo, se necesitan más repeticiones para confirmar o refutar estos resultados.

Los hallazgos tienen implicaciones prácticas. Los experimentos controlados demostraron que los participantes —sin conocimiento previos en computación en la nube— lograron identificar, modelar y aprovisionar la infraestructura en Amazon Web Services. En consecuencia, MoCIP y su herramienta de soporte ARGON prometen apoyar a los ingenieros de software noveles en el proceso de aprovisionar infraestructura en la nube.

Es importante tener en cuenta que los dos experimentos controlados presentan resultados preliminares sobre la percepción de los participantes sobre facilidad de uso, utilidad e intención de uso. Aunque los resultados son prometedores, deben interpretarse con precaución, ya que son válidos dentro del contexto establecido en los experimentos.

Capítulo 8

Validación de ARGON

La herramienta ARGON (Sandobalín *et al.*, 2017a) *abstrae* y *automatiza* la definición y el aprovisionamiento de la infraestructura en la nube. ARGON es una herramienta que soporta las actividades de MoCIP. En este capítulo se presenta un *ciclo empírico* (Wieringa, 2014) para responder a las preguntas de investigación sobre la comparación de las herramientas ARGON y Ansible respecto a la definición de los recursos de infraestructura de Amazon Web Services. Para realizar el ciclo empírico se diseñó y ejecutó una familia de experimentos.

En la Sección 8.1 se presentan las herramientas ARGON y Ansible, así como las tareas experimentales que los participantes deben realizar.

En la Sección 8.2 se presenta el diseño de la familia de experimentos, en donde se define el objetivo de los experimentos, las preguntas de investigación, la selección del contexto de los experimentos y las tareas experimentales.

En la Sección 8.3 se presenta la estadística descriptiva y un análisis de datos exploratorio de los resultados. Además, se presentan las pruebas de hipótesis para contestar a las preguntas de investigación.

En la Sección 8.4 se presentan las amenazas a la validez de la familia de experimentos.

En la Sección 8.5 se presentan las conclusiones de los resultados de la familia de experimentos.

8.1 Herramientas de aprovisionamiento de infraestructura

En esta sección se ofrece una descripción general de las herramientas de aprovisionamiento de infraestructura seleccionadas como tratamientos. Por un lado, se seleccionó ARGON (Sandobalín *et al.*, 2017a) porque es una herramienta que abstrae la complejidad de definir la infraestructura de diferentes proveedores de servicios IaaS (*Infrastructure as a Service*) a través del lenguaje de modelado ArgonML. Por otro lado, se escogió Ansible (DeHaan, 2012) como el tratamiento de control porque, actualmente, es una herramienta ampliamente utilizada en la industria y la academia para definir la infraestructura en scripts y luego implementarla en diferentes proveedores de nube (Hochstein *et al.*, 2017)(Geerling, 2015). Tanto Ansible como ARGON definen la infraestructura de la nube, aunque sus anotaciones y niveles de abstracción son significativamente diferentes.

8.1.1 Ansible: Una herramienta centrada en el código

Ansible (DeHaan, 2012) es una herramienta para definir el estado final de los recursos de infraestructura, así como para realizar el aprovisionamiento de infraestructura en la nube. Ansible utiliza un lenguaje específico de dominio para escribir código con el fin de especificar cada elemento de infraestructura para un proveedor de nube en particular.

Ansible utiliza un script llamado *Playbook* para describir la configuración de aprovisionamiento junto con una lista ordenada de tareas (*Task*) que se realizarán en un proveedor de nube específico. La sintaxis del *Playbook* está construida sobre YAML (Evans, 2001), que es un lenguaje de formato de datos que fue diseñado para ser fácil de leer y escribir. Cada *Playbook* se compone de uno o más *Plays* en una lista. Un *Play* asigna algunas tareas (*Task*) bien definidas a un grupo de hosts remotos, es decir, máquinas virtuales en un proveedor de nube específico. Finalmente, un *Task* es una llamada a un módulo de Ansible para crear y administrar recursos de infraestructura en hosts remotos.

La Figura 8-1 muestra un extracto del *Playbook* (script) de Ansible para la Empresa MODAFIN (Nitto *et al.*, 2017). MODAFIN se especializa en aplicaciones de TI para servicios financieros, y su línea principal de productos es una solución patentada para las operaciones del mercado de valores, administración de efectivo y gestión de préstamos. MODAFIN está preocupado por el tráfico exponencial de solicitudes recibidas a aplicaciones de software instaladas en Amazon Web Services. Para resolver este problema, MODAFIN decide usar un balanceador de carga para distribuir la carga de trabajo entre varias máquinas virtuales.

El *Playbook* (Figura 8-1) comienza con `---` (línea 1), que indica el inicio de un script de infraestructura para Ansible. Un *Play* tiene tres secciones: la sección de host, la sección de variables y la sección de tareas (*Tasks*). La sección de host define en qué host remoto (línea 2) de un proveedor de la nube se ejecutará el *Play* y además cómo se implementará el *Play* (línea 3).

La sección de variables define variables que se utilizarán en todo el *Play*. En este caso, se define la variable de región (línea 6) para especificar el código de región donde se aprovisionará la infraestructura en Amazon Web Services, así como el par de claves (línea 7) utilizadas por Ansible para obtener acceso a la región seleccionada.

```

1
2 ---
3 - hosts: localhost
4   connection: local
5   gather_facts: no
6   vars:
7     region: ap-southeast-2
8     key: kp-sydney
9   tasks:
10  - name: create security group
11    ec2_group:
12      region: "{{ region }}"
13      name: SG01alucoud00
14      description: Group for Virtual Machine
15      rules:
16        - proto: tcp
17          from_port: 8080
18          to_port: 8080
19          cidr_ip: 0.0.0.0/0
20        - proto: tcp
21          from_port: 22
22          to_port: 22
23          cidr_ip: 0.0.0.0/0
24        - proto: tcp
25          from_port: 25
26          to_port: 25
27          cidr_ip: 0.0.0.0/0
28      rules_egress:
29        - proto: all
30          cidr_ip: 0.0.0.0/0
31      register: SG01alucoud00
32  - name: create security group
33    ec2_group:
34      region: "{{ region }}"
35      name: SG02alucoud00
36      description: Group for Load Balancer
37      rules:
38        - proto: tcp
39          from_port: 80
40          to_port: 80
41          cidr_ip: 0.0.0.0/0
42      rules_egress:
43        - proto: all
44          cidr_ip: 0.0.0.0/0
45      register: SG02alucoud00
46
47  - name: create virtual machines
48    ec2:
49      region: "{{ region }}"
50      key_name: "{{ key }}"
51      instance_type: t2.micro
52      image: ami-05286bb73f65a1cbf
53      instance_tags:
54        Name: VM01alucoud00
55      exact_count: 8
56      count_tag:
57        Name: VM01alucoud00
58      group: SG01alucoud00
59      zone: ap-southeast-2a
60      wait: yes
61      register: VM01alucoud00
62  - name: create load balancer
63    ec2_elb_lb:
64      region: "{{ region }}"
65      name: LB01alucoud00
66      state: present
67      security_group_names:
68        - SG02alucoud00
69      zones:
70        - ap-southeast-2a
71      health_check:
72        ping_protocol: tcp
73        ping_port: 8080
74        interval: 35
75        response_timeout: 7
76        healthy_threshold: 8
77        unhealthy_threshold: 4
78      listeners:
79        - protocol: tcp
80          load_balancer_port: 80
81          instance_port: 8080
82      register: LB01alucoud00
83  - name: registers virtual machines from load balancer
84    local_action:
85      module: ec2_elb
86      state: present
87      region: "{{ region }}"
88      ec2_elbs: LB01alucoud00
89      instance_id: "{{ item }}"
90      with_items: "{{ VM01alucoud00.instance_ids }}"
91      ignore_errors: "{{ ansible_check_mode }}"

```

Figura 8-1 Un *Playbook* de Ansible para la Empresa MODAFIN

En la sección de tareas (*Tasks*), todos los elementos de infraestructura detallados se ejecutarán, en orden, uno a la vez, contra el host remoto que coincida en la sección del host. La primera tarea (*Task*) especifica un grupo de seguridad para máquinas virtuales (de la línea 9 a la 30). El grupo de seguridad funciona como un corta fuegos (*Firewall*) para habilitar las conexiones entrantes a través del puerto 8080, el puerto 22 y el puerto 25, así como todas las conexiones salientes. La segunda tarea (*Task*) detalla un grupo de seguridad para el balanceador de

carga (de la línea 31 a la 44), que habilita las conexiones entrantes a través del puerto 80 y todas las conexiones salientes. La tercera tarea (*Task*) define una máquina virtual (de la línea 45 a la 59) en la que se especifican todas las características del hardware. Por ejemplo, el tipo de instancia (procesador, RAM, almacenamiento, etc.), el código de imagen (sistema operativo), así como la región y la zona de disponibilidad donde se aprovisionarán las máquinas virtuales. La cuarta tarea (*Task*) describe un balanceador de carga (de la línea 60 a la 80), que distribuye la carga de trabajo entre varias máquinas virtuales. El balanceador de carga tiene un elemento de comprobación de estado (de la línea 69 a la 75) que valida que las máquinas virtuales conectadas al balanceador de carga estén disponibles. Además, el balanceador de carga tiene un elemento de escucha (de la línea 76 a la 79), que atiende las solicitudes de conexión al balanceador de carga. Finalmente, la quinta tarea (*Task*) registra (de la línea 81 a la 89) todas las máquinas virtuales creadas en el balanceador de carga.

8.1.2 ARGON: Una herramienta dirigida por modelos

ARGON (Sandobalín *et al.*, 2017a) es una herramienta de modelado de infraestructura para el aprovisionamiento en la nube. ARGON utiliza la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering*, MDE) para soportar la Infraestructura como Código (*Infrastructure as Code*, IaC). ARGON abstrae la complejidad de trabajar con diferentes proveedores de la nube mediante el uso de un lenguaje de modelado ArgonML. ARGON utiliza el lenguaje ArgonML para modelar los recursos de infraestructura y luego utiliza transformaciones de modelo a texto para generar los scripts correspondientes para administrar diferentes herramientas de aprovisionamiento, tales como Ansible (DeHaan, 2012) y Terraform (HashiCorp, 2017). Por otro lado, ARGON se construyó siguiendo una arquitectura basada en plugins y funciona en Eclipse Modeling Framework (Steinberg *et al.*, 2008).

La Figura 8-2 muestra un modelo de infraestructura modelado con ArgonML para la Universidad CEC (Sandobalín *et al.*, 2017a). CEC ofrece cursos masivos en línea a través de una plataforma virtual. El problema de la plataforma virtual de CEC es la gran demanda de cursos específicos, que están causando una sobrecarga en los servidores donde se ejecutan los cursos.

CEC podría comprar nuevos servidores para crear un clúster y resolver su problema. Sin embargo, los nuevos servidores no funcionarían si no hay demanda de cursos. Por lo tanto, la solución es migrar la infraestructura hacia Amazon Web Services y pagar por el uso de la infraestructura aprovisionada. En este escenario, CEC debería usar un balanceador de carga para distribuir la carga de trabajo entre varias máquinas virtuales.

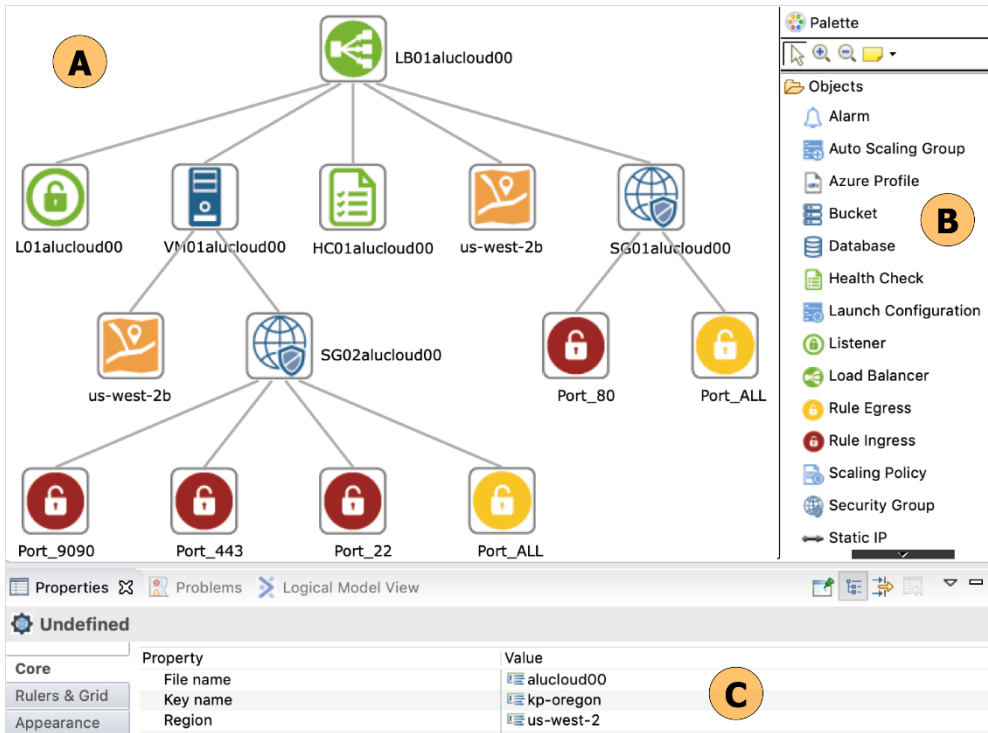


Figura 8-2 Un modelo de infraestructura de ARGON para la Universidad CEC

La Figura 8-2A muestra el modelo de infraestructura, en donde un elemento máquina virtual (*VM02alucloud00*) está conectado a un grupo de seguridad (*SG02alucloud00*) que funciona como un corta fuegos (*Firewall*). Es importante mencionar que el elemento máquina virtual (*VM02alucloud00*) define la cantidad de instancias que se crearán. El grupo de seguridad (*SG02alucloud00*) tiene tres reglas de entrada (*Port_9090*, *Port_443* y *Port_22*) y una regla de salida (*Port_ALL*). Las reglas de entrada permiten conexiones entrantes a máquinas virtuales y la regla de salida permite conexiones salientes desde las máquinas virtuales a servidores externos. Además, un elemento de zona (*us-west-2b*) especifica la zona de disponibilidad donde se aprovisionarán las máquinas virtuales (*VM02alucloud00*). El balanceador de carga (*LB01alucloud00*) distribuye la carga de trabajo entre las máquinas virtuales (*VM02alucloud00*). El balanceador de carga tiene un elemento de escucha (*L01alucloud00*) y un elemento de comprobación de estado (*HC01alucloud00*). El elemento de escucha (*L01alucloud00*) verifica las solicitudes de conexión al balanceador de carga. El elemento de comprobación de estado (*HC01alucloud00*) valida que las máquinas virtuales conectadas al balanceador de carga estén disponibles. Además, el balanceador de carga (*LB01alucloud00*) tiene un elemento de zona (*us-west-2b*),

que especifica la zona de disponibilidad donde se aprovisionará. Finalmente, el balanceador de carga (*LB01alucloud00*) tiene un grupo de seguridad (*SG01alucloud00*) con una regla de entrada (*Port_80*) y una regla de salida (*Port_ALL*).

Por otro lado, la Figura 8-2B muestra la paleta de elementos de infraestructura que se utilizan para modelar un modelo de infraestructura. Se debe tener en cuenta que cada elemento de infraestructura tiene sus propiedades. En este caso, la Figura 8-2C muestra las propiedades del modelo de infraestructura, donde la propiedad *File name* especifica el nombre del script que se generará, la propiedad *Key name* permite escribir el par de claves para tener un acceso seguro hacia Amazon Web Services, y la propiedad *Region* permite definir el código de región donde se aprovisionarán los recursos de infraestructura.

8.1.3 Comparación de herramientas

Tanto Ansible como ARGON brindan soporte para definir el estado final de los recursos de infraestructura de la nube. Por un lado, Ansible permite especificar los elementos de infraestructura en un script utilizando el lenguaje de scripting YAML. Por otro lado, ARGON permite modelar los recursos de infraestructura en un modelo de infraestructura y, a partir del modelo, genera los scripts correspondientes para administrar diferentes herramientas de aprovisionamiento, tales como Ansible, Terraform, etc.

Tabla 8-1 Comparación entre los elementos de infraestructura de ARGON y los módulos de Ansible

ARGON	Ansible
ec2	Virtual Machine
ec2_group	Security Group
-	Inbound Rule
-	Outbound Rule
ec2_eip	Static IP
ec2_elb_lb	Load Balancer
-	Listener
-	Health Check
rds	Database
s3_bucket	Bucket
ec2_lc	Launch Configuration
ec2_asg	Auto Scaling Group
ec2_scaling_policy	Scaling Policy
ec2_metric_alarm	Alarm

La Tabla 8-1 presenta un mapeo entre los elementos de infraestructura de las herramientas Ansible y ARGON. En esta comparación, es importante mencionar que tanto Ansible como ARGON pueden definir la infraestructura para diferentes proveedores de servicios en la nube. Por lo tanto, la comparación se centra en especificar la infraestructura para Amazon Web Services.

ARGON proporciona más elementos de infraestructura que Ansible porque propone un modelo de infraestructura en donde se especifican los elementos en detalle. Sin embargo, Ansible incluye uno o más elementos de infraestructura en un módulo, por ejemplo, el módulo `ec2_group` permite configurar un grupo de seguridad junto con sus reglas de entrada y salida. Además, el módulo `ec2_elb_lb` permite configurar un balanceador de carga junto con sus elementos de escucha y comprobación de estado. Los elementos restantes coinciden, uno por uno, entre los módulos de Ansible y los elementos de infraestructura de ARGON.

8.2 La familia de experimentos

De acuerdo con (Basili *et al.*, 1999) una familia de experimentos es un grupo de experimentos que persiguen el mismo objetivo. En este caso, los resultados se pueden combinar y potencialmente son más maduros que los resultados en experimentos aislados. Además, (Santos *et al.*, 2018) contribuyen a esta definición agregando que al menos dos tratamientos deben compararse dentro de todos los experimentos en una variable de respuesta común y al menos tres experimentos deben incluirse dentro de la familia. Por lo tanto, se realiza una familia de tres experimentos controlados para proveer resultados en la comparación de dos tratamientos, es decir, Ansible versus ARGON.

La familia de experimentos se ha definido de acuerdo con el marco propuesto por (Ciolkowski *et al.*, 2002) y cada experimento ha sido diseñado y ejecutado siguiendo el proceso experimental propuesto por (Wohlin *et al.*, 2012).

8.2.1 Objetivo

Una familia de experimentos tiene que establecer un objetivo para permitir un análisis efectivo de los resultados individuales (Ciolkowski *et al.*, 2002), así como para definir el alcance de los experimentos.

De acuerdo con el paradigma *Goal-Question-Metric* (Basili *et al.*, 1994) (GQM), el objetivo de la familia de experimentos es:

Analizar la definición de la infraestructura de la nube especificada por Ansible y ARGON, *con el fin* de evaluar las herramientas IaC, *con respecto a* su eficiencia, efectividad, facilidad de uso percibida, utilidad percibida e intención de uso, *desde el punto de vista de* los ingenieros de software noveles, *en el contexto de* los estudiantes del grado en Ingeniería Informática y del máster en Ingeniería y Tecnología de Sistemas Software.

Aunque los profesionales de la computación en la nube serían la mejor selección, la familia de experimentos se centra en el perfil de los ingenieros de software noveles. En este caso, el propósito de los experimentos es proporcionar una herramienta que ayudará a los usuarios menos experimentados a especificar los recursos de infraestructura. En consecuencia, las preguntas de investigación (*Research Question*, RQ) son la siguientes:

- **RQ1:** ¿Cuál herramienta IaC es más efectiva al definir la infraestructura de la nube, ARGON o Ansible?
- **RQ2:** ¿Cuál herramienta IaC es más eficiente al definir la infraestructura de la nube, ARGON o Ansible?
- **RQ3:** ¿Cuál herramienta IaC se percibe como más fácil de usar, ARGON o Ansible?
- **RQ4:** ¿Cuál herramienta IaC se percibe como más útil, ARGON o Ansible?
- **RQ5:** ¿Cuál herramienta IaC tienen la mayor intención de uso en el futuro, ARGON o Ansible?

8.2.2 Selección del contexto

El contexto de este estudio es la definición de la infraestructura de la nube definidas por ingenieros de software noveles. En este caso, el contexto de la familia de experimentos se define por (1) los objetos experimentales, (2) las herramientas IaC y (3) la selección de los participantes.

8.2.2.1 Objetos experimentales

Los recursos de infraestructura a especificar se seleccionaron y adaptaron de la literatura:

- O1—Empresa MODAFIN (Nitto *et al.*, 2017): el propósito es resolver el tráfico exponencial de solicitudes recibidas a aplicaciones de software instaladas en Amazon Web Services mediante el uso de un balanceador de carga que distribuye la carga de trabajo entre varias máquinas virtuales. La Figura 8-1 muestra el *Playbook* (script) de Ansible para la empresa MODAFIN.

- O2—Universidad CEC (Sandobalín *et al.*, 2017a): el objetivo es pagar por el uso de la infraestructura aprovisionada en Amazon Web Services con el fin de resolver la alta demanda de cursos específicos que están causando una sobrecarga en la plataforma virtual (servidores) de la Universidad CEC. En este caso, un balanceador de carga distribuye la carga de trabajo entre varias máquinas virtuales. La Figura 8-2 muestra el modelo de infraestructura de ARGON para la Universidad CEC.

8.2.2.2 Herramientas IaC

Ansible es ampliamente utilizada por los profesionales para definir, actualizar y ejecutar los recursos de infraestructura en la nube. Utiliza el lenguaje de scripting YAML para definir las instrucciones necesarias para especificar los elementos de infraestructura en un *Playbook* (script). En este caso, se pidió a los participantes que escribieran las instrucciones necesarias para especificar un balanceador de carga que distribuye la carga de trabajo entre varias máquinas virtuales. La Figura 8-1 muestra la especificación del script para la empresa MODAFIN. Primero, el *Playbook* debe comenzar con tres guiones (---) que indican el inicio del script. A continuación, los participantes deben especificar hosts remotos en los que se ejecutará el script. Además, deben definir las variables que se utilizarán a lo largo de todo el script. Finalmente, deben definir los módulos de Ansible para crear los elementos de infraestructura en la nube.

Por otro lado, ARGON proporciona el lenguaje ArgonML para modelar los recursos de infraestructura. En este caso, se pidió a los participantes que modelaran un balanceador de carga que distribuye la carga de trabajo entre varias máquinas virtuales. La Figura 8-2 muestra el modelo de infraestructura para la Universidad CEC. Primero, se debe crear un modelo de infraestructura en Eclipse Modeling Framework. A continuación, desde la paleta de elementos de infraestructura se arrastra y suelta cada elemento sobre el lienzo de Eclipse. Además, es necesario hacer las conexiones entre elementos. Finalmente, se deben completar las propiedades del diagrama de infraestructura, así como las propiedades de cada elemento de infraestructura.

8.2.2.3 Selección de participantes

De acuerdo con (Kitchenham *et al.*, 2002), los estudiantes son la próxima generación de profesionales del software y, por lo tanto, están relativamente cerca de la población de interés. Además, (Höst *et al.*, 2000) investigaron la idoneidad de los estudiantes de los últimos años como sujetos y concluyeron que si están bien entrenados, pueden considerarse como sujetos experimentales apropiados. En consecuencia, se seleccionó el siguiente grupo de participantes:

- Estudiantes de posgrado matriculados en el máster de Ingeniería y Tecnología de Sistemas Software en la Universitat Politècnica de València (UPV). Los estudiantes asistieron al curso de Ingeniería de Software Empírica a lo largo del curso académico 2017-18. El propósito del curso es proporcionar a los estudiantes los conocimientos necesarios para planificar, ejecutar y presentar los resultados de estudios empíricos.
- Estudiantes de pregrado matriculados en el grado de Ingeniería Informática en la UPV. Estos estudiantes asistieron al curso de Desarrollo de Software Dirigido por Modelos a lo largo del curso académico 2017-18. El propósito del curso es proporcionar a los estudiantes el conocimiento sobre la construcción de modelos de software en diferentes niveles de abstracción como artefactos principales en el desarrollo de software.
- Estudiantes de pregrado matriculados en el grado de Ingeniería Informática en la UPV. Estos estudiantes asistieron al curso de Ingeniería de Requisitos a lo largo del curso académico 2017-2018. El propósito del curso es comprender las necesidades de los usuarios, así como obtener, analizar, negociar y documentar los requisitos de software.

No se estableció una clasificación de los participantes en función de su experiencia en la definición de la infraestructura de la nube, porque los estudiantes de pregrado y posgrado no tenían experiencia previa en la computación en la nube.

8.2.3 *Diseño de experimentos individuales*

La Figura 8-3 resume la familia de experimentos. Se incluye el contexto de cada experimento, el número de participantes involucrados y el lugar donde tuvieron lugar los experimentos. Además, se especifica el orden de ejecución de los experimentos individuales.



Figura 8-3 Resumen de la familia de experimentos

Dado que las condiciones experimentales son difíciles de controlar, una forma de satisfacer los requisitos estadísticos es ejecutando réplicas internas, es decir, en el mismo lugar y por los mismos experimentadores (Juristo *et al.*, 2012). Tener más repeticiones internas del mismo experimento reduce considerablemente el error de Tipo I, y también se requieren repeticiones idénticas para poder estimar el tamaño del efecto en estudio (Juristo *et al.*, 2012). Un error Tipo I (error α , falsos positivos) ocurre cuando la hipótesis nula (H_0) se rechaza a favor de la hipótesis alternativa (H_1), cuando la hipótesis “nula” es realmente verdadera. El tamaño del efecto indica la magnitud del efecto observado o la relación entre las variables.

La familia de experimentos está compuesta del experimento de línea de base realizado con estudiantes de maestría (UPV1). Antes de realizar este experimento, se realizó un experimento piloto con 12 estudiantes de doctorado. Los estudiantes evaluaron los materiales experimentales con respecto a los procedimientos experimentales, las instrucciones y el tiempo de finalización de la tarea. Cabe señalar que los estudiantes de doctorado no participaron en los experimentos controlados.

Para verificar los resultados obtenidos en el experimento de línea base, se realizó dos réplicas estrictas de acuerdo con las directrices propuestas en (Carver, 2010). La primera replicación se realizó con estudiantes de pregrado matriculados en el curso de Desarrollo de Software Dirigido por Modelos (UPV2), mientras que la segunda replicación se realizó también con estudiantes de pregrado, pero inscritos en el curso de Ingeniería de Requisitos (UPV3). Estas réplicas fueron operativas, ya que se cambió algunas dimensiones de la configuración experimental (Gómez *et al.*, 2014). En particular, se cambió la población. Este cambio permitió verificar si los resultados son independientes del perfil de los participantes.

8.2.3.1 Experimento de línea base

El experimento tuvo como objetivo evaluar si los participantes que usan la herramienta ARGON para modelar la infraestructura pueden obtener un rendimiento más alto —en términos de efectividad y eficiencia— y una mejor percepción que cuando usan la herramienta Ansible.

8.2.3.1.1 Selección del contexto

Se utilizó los objetos experimentales explicados en la Sección 8.2.2.1 junto con las herramientas de IaC descritas en la Sección 8.2.2.2. La Tabla 8-2 presenta los requisitos de infraestructura utilizados para especificar la infraestructura de la nube. Los requisitos fueron tomados y adaptados de la literatura (Sandobalín *et al.*, 2017a), (Nitto *et al.*, 2017).

Tabla 8-2 Requisitos para definir los recursos de infraestructura de la nube

No.	Descripción del requisito
R1	La infraestructura debe implementarse en una región específica de Amazon Web Services.
R2	Un grupo de seguridad debe habilitar conexiones entrantes TCP a máquinas virtuales a través de tres puertos específicos. Además, el grupo de seguridad debe habilitar todas las conexiones salientes desde las máquinas virtuales.
R3	Un grupo de seguridad debe habilitar las conexiones TCP entrantes a un balanceador de carga a través del puerto 80. Además, el grupo de seguridad debe habilitar todas las conexiones salientes desde el balanceador de carga.
R4	Se debe ejecutar un conjunto de máquinas virtuales en una zona de disponibilidad específica de una región seleccionada de Amazon Web Services. Cada máquina virtual debe tener una CPU y RAM explícitas. Además, cada máquina virtual debe tener un sistema operativo específico y un servidor web instalado.
R5	Un balanceador de carga debe distribuir la carga de trabajo entre las máquinas virtuales. El balanceador de carga debe responder a las solicitudes de los clientes y validar que todas las máquinas virtuales conectadas estén disponibles. Además, el balanceador de carga debería funcionar en una zona de disponibilidad específica de una región seleccionada de Amazon Web Services.
R6	El balanceador de carga utiliza un elemento de comprobación de estado para validar el estado de las máquinas virtuales conectadas al balanceador de carga. En este caso, se emplea intervalos de comprobación utilizando el protocolo TCP y un número de puerto específico. Es necesario esperar un tiempo en segundos para notificar una verificación de error. El balanceador de carga debe recibir una cantidad precisa de errores consecutivos para pasar a un estado "no saludable" a una máquina virtual, mientras que debe recibir un número específico de éxitos de la sonda de verificación de estado para cambiar a un estado "saludable" a una máquina virtual.
R7	El balanceador de carga utiliza un elemento de escucha para responder a las solicitudes del cliente a través del protocolo TCP y el puerto 80, así como para distribuir toda la carga de trabajo a las máquinas virtuales mediante un puerto determinado.
R8	Las máquinas virtuales deben registrarse en el balanceador de carga, de modo que el balanceador de carga pueda distribuir la carga de trabajo entre todas las máquinas virtuales disponibles.

Es importante mencionar que los requisitos son independientes tanto de los objetos experimentales como de las herramientas IaC. Además, los objetos experimentales son de diferentes dominios de aplicación y no se requiere conocimientos especializados para comprenderlos, pero tienen una complejidad similar.

8.2.3.1.2 *Participantes*

El experimento de línea base involucró a 22 estudiantes de posgrado matriculados en el máster de Ingeniería y Tecnología de Sistemas Software en la UPV.

El conocimiento previo y la experiencia de los participantes fueron evaluados a través de un cuestionario previo al experimento. El cuestionario previo reportó que 14 participantes tenían experiencia profesional en el desarrollo de software, que variaba entre 1 y 4 años, con un promedio de 2 años, pero que no tenían conocimientos previos de computación en la nube ni del proceso de aprovisionamiento de infraestructura. Los participantes fueron elegidos por muestreo de conveniencia. Asistieron al curso de otoño de 2017 sobre Ingeniería de Software Empírica. Se pidió a los participantes que realizaran la tarea experimental como parte de los ejercicios del laboratorio del curso.

8.2.3.1.3 *Selección de variables*

El factor o variable independiente es la herramienta IaC, que tiene dos niveles o tratamientos: Ansible y ARGON. Ansible es una herramienta centrada en el código que utiliza scripts para especificar los recursos de infraestructura. En cambio, ARGON utiliza un enfoque dirigido por modelos para modelar los recursos de infraestructura.

Se utilizó el modelo de evaluación del método (*Method Evaluation Model*, MEM) (Moody, 2003) como base teórica para diseñar el experimento. De acuerdo con este modelo, hay dos tipos de variables dependientes: variables basadas en el desempeño (*performance-based variables*) que miden qué tan bien los participantes realizan la tarea experimental (es decir, definir los recursos de infraestructura) y las variables basadas en la percepción (*perception-based variables*) que miden las creencias y actitudes de los participantes hacia el uso de las herramientas IaC.

Existe dos variables basadas en el rendimiento:

- **Efectividad**, que mide el grado en que una herramienta IaC logra sus objetivos.
- **Eficiencia**, que mide el esfuerzo requerido para usar una herramienta IaC.

Existe tres variables basadas en la percepción:

- **Facilidad de uso percibida** (*Perceived Ease of Use*, PEOU), que se refiere al grado en que un participante cree que aprender y usar una herramienta IaC sería fácil.
- **Utilidad percibida** (*Perceived Usefulness*, PU), que se refiere al grado en que un participante cree que una herramienta IaC será efectiva para lograr sus objetivos.
- **Intención de uso** (*Intention to Use*, ITU), que es la medida en que un participante intenta utilizar una herramienta IaC.

Con el fin de operacionalizar las variables basadas en el rendimiento, se utiliza la norma ISO/IEC 9126-4 (ISO/IEC, 1991) para obtener las métricas para medir la efectividad y la eficiencia. Además, para operacionalizar las variables basadas en la percepción, se adaptó el instrumento de medición propuesto por (Davis *et al.*, 1989) para medir la facilidad de uso percibida, la utilidad percibida y la intención de uso. La Tabla 8-3 resume las métricas utilizadas para medir cada variable dependiente.

Tabla 8-3 Resumen de las variables dependientes de la familia de experimentos

Nombre	Medida	Escala
Efectividad	$\frac{\# \text{ Requistos correctos}}{\# \text{ Total de requisitos}}$	Ratio
Eficiencia	$\frac{\text{Efectividad}}{\text{Tiempo}}$	Ratio
PEOU	Escala Likert de 5 puntos	Ordinal
PU	Escala Likert de 5 puntos	Ordinal
ITU	Escala Likert de 5 puntos	Ordinal

Dado que la tarea experimental es un conjunto de requisitos que deben codificarse en Ansible o modelarse en ARGON, la *Efectividad* se calcula como el número de requisitos definidos correctamente dividido por el número total de requisitos propuestos. Esto refleja la exactitud de la definición de los recursos de infraestructura de la nube especificada por los participantes. Además, la *Eficiencia* se calculó como la *Efectividad* de cada participante dividida por el *Tiempo* que él/ella dedica a realizar la tarea. Esto refleja la proporción de requisitos logrados para cada unidad de tiempo. La eficiencia aumenta al aumentar la efectividad y reducir el tiempo de la tarea. De hecho, se definió la siguiente

métrica de agregación para decidir si una tarea (es decir, un conjunto de requisitos) es válida o no:

- *Métrica de todo o nada*: se considera que un requisito es correcto solo si se define correctamente. Por lo tanto, un requisito tiene solo dos valores posibles: éxito o fracaso (1 o 0).

En este caso, se propone la *métrica de todo o nada* porque los recursos de infraestructura definidos en el experimento deberían funcionar en un proveedor de servicios en la nube y, por lo tanto, la infraestructura no se aprovisionará si existe algún requisito que esté mal definido.

Tabla 8-4 Ítems para mediar las variables basadas en la percepción

Tipo	Descripción del Ítem
PEOU1	El procedimiento para usar la herramienta IaC me pareció complejo y difícil de seguir.
PEOU2	En general, la herramienta IaC me pareció difícil de usar.
PEOU3	Encontré la herramienta IaC fácil de aprender.
PEOU4	Me resultó difícil definir la infraestructura de la nube con la herramienta IaC.
PEOU5	El uso de la herramienta IaC me pareció claro y fácil de entender.
PU1	Creo que la herramienta IaC reduciría el esfuerzo requerido para definir la infraestructura de la nube.
PU2	En general, encontré que la herramienta IaC es útil.
PU3	La infraestructura en la nube definida con la herramienta IaC sería más difícil de entender.
PU4	En general, creo que la herramienta IaC no proporciona una solución efectiva para definir la infraestructura de la nube.
PU5	En general, creo que la herramienta IaC es una mejora en la definición de la infraestructura de la nube.
PU6	La herramienta IaC facilitaría a los profesionales la definición de la infraestructura de la nube.
PU7	El uso de la herramienta IaC facilitaría la comunicación de la definición de la infraestructura de la nube a otros profesionales.
ITU1	Recomendaría la herramienta IaC para definir la infraestructura de la nube.
ITU2	Si estoy trabajando en una empresa en el futuro, me gustaría usar la herramienta IaC para definir la infraestructura de la nube.
ITU3	Sería fácil para mí ser hábil al usar la herramienta IaC para definir la infraestructura de la nube.

En contraste, para operacionalizar las variables basadas en la percepción, se utiliza un cuestionario de la encuesta adaptada de (Moody, 2003) para medir PEOU, PU e ITU. La Tabla 8-4 presenta los ítems para medir las variables basadas en la percepción. Los ítems del cuestionario fueron formulados usando una escala Likert de 5 puntos y adoptando el formato de pregunta de enunciado opuesto. Los ítems fueron ordenados de manera aleatoria dentro del mismo grupo de constructos, para prevenir el sesgo de respuesta sistémica.

8.2.3.1.4 *Formulación de hipótesis*

Se formuló las hipótesis nulas basadas en las variables dependientes. En este caso, es importante mencionar que el experimento tiene como objetivo evaluar la definición de los recursos de infraestructura en lugar de evaluar las herramientas IaC de manera integral.

Las hipótesis nulas del experimento se pueden resumir de la siguiente manera:

- H1₀: Efectividad (Ansible) = Efectividad (ARGON)
- H2₀: Eficiencia (Ansible) = Eficiencia (ARGON)
- H3₀: PEOU (Ansible) = PEOU (ARGON)
- H4₀: PU (Ansible) = PU (ARGON)
- H5₀: ITU (Ansible) = ITU (ARGON)

El objetivo del análisis estadístico es rechazar las hipótesis nulas y posiblemente aceptar las alternativas (por ejemplo, $H1_1 = \neg H1_0$). Todas las hipótesis son bilaterales porque no se postula que ocurrirá ningún efecto como resultado del uso de cada herramienta IaC.

8.2.3.1.5 *Diseño*

El experimento fue diseñado como un diseño cruzado AB/BA, que tiene un factor y dos tratamientos. En este contexto, la herramienta IaC para especificar la infraestructura es el factor, y los dos tratamientos son Ansible y ARGON. Se eligió el diseño cruzado AB/BA porque aborda el problema de los tamaños de muestra pequeñas y aumenta la sensibilidad de los experimentos. Se utilizó las directrices propuestas por (Vegas *et al.*, 2016) para definir el diseño cruzado. En consecuencia, se utilizó los factores fijos, tales como período, secuencia y carryover.

La Tabla 8-5 muestra un tipo especial de diseño llamado diseño cruzado factorial que tiene el mismo número de períodos y de tratamientos, en donde todos los participantes aplican cada tratamiento una sola vez (Vegas *et al.*, 2016). En este escenario, se debe diferenciar entre los conceptos de período y sesión. Un período se define mediante la aplicación de un tratamiento por un participante a un objeto experimental, mientras que una sesión es una parte del tiempo que un sujeto dedica a completar (una o más) tareas experimentales (Vegas *et al.*, 2016).

En consecuencia, el experimento tiene dos períodos debido a que cada participante debe realizar dos tratamientos. Además, es importante mencionar que, debido al horario de clases de los estudiantes, se llevó a cabo un período en una sesión.

Tabla 8-5 Diseño cruzado AB/BA para configura el experimento de línea base

Objeto Período		O1: MODAFIN Período 1		O2: CEC Período 2	
Técnica	Secuencia	Ansible	ARGON	Ansible	ARGON
Grupo 1	Ansible-ARGON	X	-	-	X
Grupo 2	ARGON-Ansible	-	X	X	-

Debido a que existe dos períodos y dos tratamientos, las secuencias resultantes son dos, es decir, Ansible-ARGON y ARGON-Ansible. En el primer período, el Grupo 1 resuelve el objeto experimental O1 con Ansible, mientras que el Grupo 2 resuelve el mismo objeto experimental con ARGON. En el segundo período, el Grupo 1 resuelve el objeto experimental O2 con ARGON, mientras que el Grupo 2 resuelve el mismo objeto experimental con Ansible. En este escenario, no se plantea la posibilidad que ninguna de las secuencias mejore los resultados experimentales, ya que Ansible y ARGON definen la infraestructura de la nube en base a diferentes principios e insumos.

El experimento utiliza dos objetos experimentales diferentes (O1—MODAFIN y O2—CEC) y cada objeto experimental describe ocho requisitos que los participantes deben implementar. Según el diseño del experimento, cada objeto experimental debe usarse en un período diferente. Similar al caso de las secuencias, no se plantea la posibilidad que el orden de uso de los objetos experimentales mejore los resultados del experimento.

A lo largo del experimento, si el efecto de un tratamiento continúa después de que se retira el tratamiento, entonces la respuesta a un segundo tratamiento puede deberse en parte al tratamiento anterior, y se produce el carryover (Vegas *et al.*, 2016). En este caso, en un diseño cruzado es complicado identificar el efecto de la secuencia, así como la interacción entre el período y el tratamiento y, por lo tanto, se debe considerar la posibilidad de que exista el carryover. Como resultado, se reconoce el efecto del carryover en la etapa de diseño, y sus resultados serán examinados en la etapa de análisis. Finalmente, el diseño cruzado está balanceado para los efectos del carryover porque cada tratamiento sigue a cada uno de los otros tratamientos un número igual de veces (Kuehl, 2000).

8.2.3.1.6 Operación

El procedimiento para ejecutar el experimento coincide con el diseño cruzado factorial elegido. El experimento se realizó a lo largo de cinco sesiones de 2 horas. La Figura 8-4 muestra un resumen de cómo se realizó el experimento. Los números en el diagrama significan pasos en el proceso experimental y, en este caso, representan las sesiones. La primera sesión de 2 horas (*Paso 1*) fue la capacitación de los fundamentos de la computación en la nube, así como de las herramientas Ansible y ARGON. Es importante mencionar la capacitación se enfocó en enseñar los conceptos principales que los participantes utilizaron para el experimento en lugar de explicar en detalle la computación en la nube o las herramientas IaC. La segunda sesión de 2 horas (*Paso 2*) fue para familiarizarse con Ansible, mientras que la tercera sesión de 2 horas (*Paso 3*) fue para familiarizarse con ARGON. Los participantes realizaron un ejercicio similar al experimento. Los primeros tres pasos se centran en la capacitación y entrenamiento sobre cómo definir la infraestructura de la nube debido que era un nuevo conocimiento que los participantes tenían que adquirir.

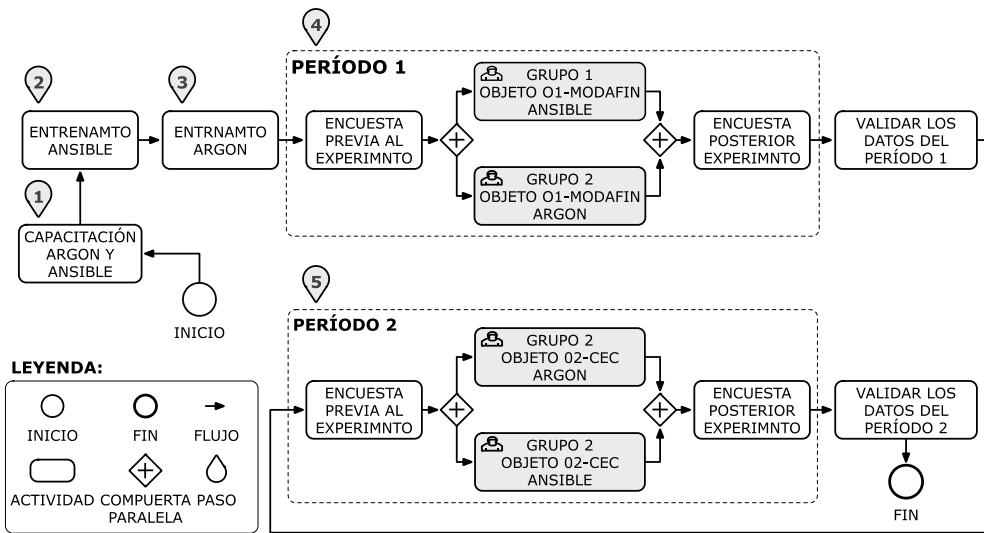


Figura 8-4 Resumen del proceso de operación del experimento de línea base

El primer período del experimento se realizó en la cuarta sesión de 2 horas (*Paso 4*), mientras que el segundo período se realizó en la quinta sesión de 2 horas (*Paso 5*). Las actividades realizadas en cada período se describen a continuación:

- Primero, los participantes completaron una encuesta relacionada con sus conocimientos y experiencia con la computación en la nube y las herramientas de IaC.

- Luego, los participantes fueron asignados aleatoriamente al Grupo 1 y al Grupo 2, considerando que ambos grupos tendrán el mismo número de participantes. En el primer período, el Grupo 1 resuelve el objeto experimental O1—MODAFIN con Ansible, mientras que el Grupo 2 resuelve el mismo objeto experimental con ARGON. En el segundo período, el Grupo 1 resuelve el objeto experimental O2—CEC con ARGON, mientras que el Grupo 2 resuelve el mismo objeto experimental con Ansible.
- Finalmente, los participantes completaron una encuesta para expresar sus percepciones sobre la facilidad de uso, la utilidad y la intención de usar los tratamientos (es decir, Ansible o ARGON). Después de cada período, se verificó que los datos estaban completos y que se habían recopilado correctamente.

8.2.3.2 Segundo experimento (UPV2)

El segundo experimento de la familia de experimentos es una replicación interna estricta del experimento de línea de base (UPV1). Se aplicó el mismo protocolo experimental, pero a una población diferente, lo que significa que cambió los participantes, mientras que el sitio, los experimentadores, el diseño, las variables y la instrumentación permanecieron igual. Se cambió únicamente a los participantes con el propósito de probar hasta qué punto los resultados experimentales podrían generalizarse. La muestra de participantes estaba compuesta por 24 estudiantes de pregrado matriculados en el curso de Desarrollo de Software Dirigido por Modelos en el tercer año de su grado y, por lo tanto, obtuvieron conocimiento de las técnicas basadas en modelos. Se pidió a los participantes que realizaran la tarea experimental como parte de los ejercicios de laboratorio del curso.

Al igual que en el experimento de línea de base, la réplica tuvo lugar en una habitación individual y no se permitió la interacción entre los participantes. Se llevó a cabo el diseño cruzado AB/BA para establecer dos tratamientos con dos períodos y los participantes fueron asignados aleatoriamente a dos grupos.

8.2.3.3 Tercer experimento (UPV3)

El tercer experimento de la familia de experimentos es una réplica estricta del experimento de línea de base (UPV1). El protocolo experimental, el sitio, el diseño, las variables, la instrumentación y los experimentadores permanecieron igual. La muestra de participantes estaba compuesta por 21 estudiantes de pregrado matriculados en el curso de Ingeniería de Requisitos en el cuarto año de su licenciatura. Se pidió a los participantes que realicen la tarea experimental como parte de los ejercicios de laboratorio del curso.

La replicación tuvo lugar en una habitación individual y no se permitió la interacción entre los participantes. También se utilizó el diseño cruzado AB/BA y, por lo tanto, se configuraron dos tratamientos con dos períodos y los participantes fueron asignados aleatoriamente a dos grupos.

8.2.4 Tareas experimentales y materiales

Las tareas experimentales consistieron en especificar la infraestructura de la nube utilizando las herramientas IaC seleccionadas con los objetos experimentales. Estas tareas se estructuraron para permitir la comparación de las herramientas IaC en términos de especificar los recursos de infraestructura. Se proporcionó a los participantes el objetivo y la descripción de la infraestructura de la nube y luego se solicitó a los participantes que especifiquen los recursos de infraestructura siguiendo los pasos y las guías de cada herramienta. En este escenario, se entregó un conjunto de requisitos a los participantes que intentaban ejemplificar un problema real en el que debían usar un balanceador de carga para distribuir la carga de trabajo entre varias máquinas virtuales.

En el caso de Ansible, la tarea experimental consistió en: (1) definir las secciones de host y variables; (2) definir la sección de tareas y luego especificar los recursos de infraestructura; y (3) registrar o cancelar el registro de los componentes, lo que significa conectar los elementos de infraestructura entre ellos para que funcionen en Amazon Web Services. En el caso de ARGON, la tarea experimental consistió en: (1) modelar los recursos de infraestructura; (2) conectar los elementos de infraestructura entre sí; y (3) llenar los valores de las propiedades de cada elemento de infraestructura.

Los materiales experimentales están formados por un conjunto de documentos para apoyar la tarea experimental, sesiones de capacitación y cuestionarios previos y posteriores.

El material de capacitación incluyó: (1) diapositivas para capacitar en los fundamentos de la computación en la nube, Amazon Web Services e Infraestructura como Código; (2) diapositivas para explicar la herramienta Ansible y un ejemplo sobre cómo especificar los elementos de infraestructura en un script; y (3) diapositivas para explicar la herramienta ARGON junto con un ejemplo ilustrativo sobre cómo modelar los elementos de infraestructura.

El material experimental que respalda la tarea experimental de cada experimento incluye:

- Cuatro folletos que cubren las cuatro combinaciones posibles entre objetos experimentales (O1 y O2) y tratamientos (Ansible y ARGON). El propósito de estos folletos era describir cada objeto experimental

junto con sus requisitos, así como describir los elementos de infraestructura, sus propiedades y sus relaciones.

- Un cuestionario previo para recopilar los conocimientos y habilidades de los participantes, y un cuestionario posterior para recopilar las percepciones de los participantes sobre la facilidad de uso, la utilidad y la intención de uso.
- Una guía que explica los pasos de cómo crear un *Playbook*, es decir, un script para Ansible. La guía incluye definiciones de las secciones del script y fragmentos de código con el propósito de que los participantes las usen como referencia para crear su código. El experimento tiene como objetivo que los participantes aprendan a crear un script en lugar de memorizar fragmentos de código.
- Una guía que explica los pasos para modelar los recursos de infraestructura de la nube con ARGON. La guía incluye las instrucciones para crear un proyecto de infraestructura. Además, la guía incluye las instrucciones para utilizar los elementos de infraestructura y completar sus propiedades.
- Una guía que explica las regiones y zonas de disponibilidad para Amazon Web Services. El balanceador de carga y las máquinas virtuales tienen que especificar la ubicación donde se aprovisionarán.
- Una guía que contiene una lista de pares de claves para acceder a las regiones de Amazon Web Services. Para aprovisionar la infraestructura en una región particular, es necesario especificar su código.
- Una guía que contiene una lista con los códigos de tipo de instancia. Los códigos tienen las características de hardware de las máquinas virtuales de Amazon Web Services.
- Una guía que contiene una lista con los códigos de imagen, es decir, el sistema operativo para una máquina virtual en Amazon Web Services.
- Una máquina virtual configurada y probada con Ansible y ARGON. El objetivo es proporcionar a cada participante un entorno idéntico para trabajar. Cada máquina virtual se configuró con Windows Server 2012R2, Java JDK v1.8, Eclipse Modeling Framework v4.8, ARGON v1.0 y Ansible v2.6.

El cuestionario posterior al experimental contenía un conjunto de preguntas cerradas que permiten a los participantes expresar sus opiniones sobre Ansible y ARGON en términos de su facilidad de uso percibida, utilidad percibida e intención de uso. Para garantizar el equilibrio de los ítems en el cuestionario, la mitad de las preguntas fueron negadas, y todos los ítems se organizaron en un orden aleatorio para reducir el efecto de techo potencial que podría inducir a

respuestas monótonas de preguntas que miden el mismo constructo (Moody, 2003). La Tabla 8-4 presenta los ítems utilizados en el cuestionario.

8.2.5 Análisis de los datos de los experimentos y metaanálisis

Se utilizó estadísticas descriptivas, diagramas de caja y pruebas estadísticas para analizar los datos recopilados de cada experimento. Como es habitual, en todas las pruebas estadísticas, se aceptó una probabilidad del 5% de cometer un error de Tipo I (*Type-I Error*). El análisis de los datos se realizó considerando los siguientes pasos:

1. Primero se realiza un estudio descriptivo de las variables dependientes.
2. Debido a que se utiliza un diseño cruzado, es necesario analizar los factores del experimento, como los períodos, las secuencias y el carryover. En este caso, se utiliza el modelo lineal mixto para evaluar si estos factores influyen en los resultados.
3. Para aplicar el modelo lineal mixto, los residuos deben cumplir la condición de normalidad (Vegas *et al.*, 2016). Por lo tanto, para garantizar que el modelo sea válido, se utiliza la prueba de Shapiro-Wilk para confirmar la normalidad de los residuos.
4. Luego, se aplica el modelo lineal mixto a cada variable dependiente para evaluar si el período (confundido con el objeto experimental), las secuencias (confundido con la interacción período*técnica y carryover) o la técnica (tratamiento) tienen significancia estadística.
5. Debido a que la significancia estadística no es suficiente para explicar la diferencia causada por los tratamientos, se mide el tamaño del efecto (*effect size*) para evaluar la magnitud de esas diferencias. El tamaño del efecto de los tratamientos solo debe medirse si el período, la secuencia o cualquier variable de bloqueo no tienen relación, y no hay carryover (Vegas *et al.*, 2016). Debido a que la familia de experimentos compara dos grupos en lugar de evaluar la fuerza de la asociación entre dos variables, se utiliza la *d* de Cohen para medir el tamaño del efecto.
6. Para fortalecer los resultados de cada experimento individual, se realizó un metaanálisis. Específicamente se realizó un metaanálisis de datos agregados (*Aggregated Data*, AD) basado en la *d* de Cohen, ya que las condiciones experimentales fueron similares para todos los experimentos. El análisis permitió obtener resultados más sólidos y extraer conclusiones más generales al considerar el conjunto de experimentos.

8.3 Resultados

En esta sección, se presenta la evidencia empírica recolectada de la familia de experimentos, así como también se discute los resultados de cada experimento. Se analiza cuantitativamente los datos de los experimentos de acuerdo con las hipótesis establecidas. Se utilizó el paquete estadístico para ciencias sociales SPSS v24 y R v3.5.2 para obtener los resultados de cada experimento.

8.3.1 Estadística descriptiva y análisis de datos exploratorio

La Tabla 8-6 presenta un resumen de las estadísticas descriptivas, tales como media y desviación estándar (*Standard Deviation*, SD) para las variables basadas en el rendimiento y en la percepción.

Tabla 8-6 Estadísticas descriptivas de la familia de experimentos

Datos	Variable	Ansible		ARGON	
		Media	SD	Media	SD
UPV1	Efectividad	0.648	0.293	0.858	0.160
	Eficiencia	0.025	0.017	0.042	0.021
	Duración	30.00	8.211	23.42	8.622
	PEOU	3.80	0.866	4.49	0.500
	PU	4.08	0.667	4.53	0.494
	ITU	4.06	0.814	4.53	0.606
UPV2	Efectividad	0.646	0.260	0.818	0.198
	Eficiencia	0.022	0.009	0.034	0.022
	Duración	30.13	6.442	27.53	9.631
	PEOU	3.65	0.813	4.11	0.584
	PU	3.60	0.646	4.07	0.516
	ITU	3.49	0.674	4.18	0.581
UPV3	Efectividad	0.649	0.359	0.869	0.134
	Eficiencia	0.019	0.011	0.041	0.020
	Duración	34.51	7.567	24.31	8.704
	PEOU	3.80	0.832	4.16	0.695
	PU	3.82	0.601	4.16	0.651
	ITU	3.76	0.518	4.14	0.688

Aunque no se analiza la *Duración*, la Tabla 8-6 incluye la variable en este resumen para dar una primera idea de la complejidad de las tareas experimentales. La

duración es el tiempo —en minutos— que el/la participante dedica a realizar una tarea experimental. La celda en negrita indica los valores de los participantes para cada experimento con la duración (tiempo) más baja y la desviación estándar más pequeña. En general, los resultados muestran que los participantes lograron un mejor rendimiento (efectividad y eficiencia) y también lograron las mejores percepciones (facilidad de uso, utilidad e intención de uso) al utilizar ARGON. Con respecto a la efectividad, se puede observar que las medidas de tendencia central son más altas para ARGON que para Ansible (los valores medios varían de 0.818 a 0.869 para ARGON y de 0.646 a 0.649 para Ansible). El significado práctico de esto es que el número de definiciones válidas de la infraestructura era mayor cuando se usaba ARGON. Los resultados también indican que la desviación estándar es mayor para Ansible, lo que indica que los participantes presentan un comportamiento más uniforme cuando usan ARGON. En cuanto a la duración de la tarea experimental, las *medidas de tendencia central* son más bajas para ARGON que para Ansible (los valores medios varían de 23.42 a 27.53 minutos para ARGON y de 30 a 34.51 minutos para Ansible).

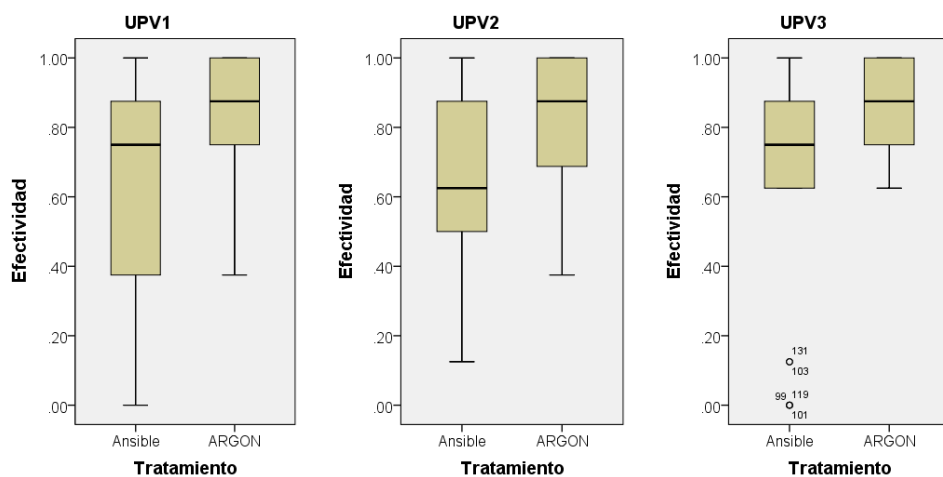


Figura 8-5 Diagramas de caja para la efectividad del tratamiento

La Figura 8-5 muestra los diagramas de caja para la efectividad del tratamiento de cada experimento. Los resultados, en general, indican que los participantes fueron más efectivos en la definición de la infraestructura de la nube al usar ARGON. Por ejemplo, el diagrama de caja para la efectividad del tratamiento de la muestra UPV1 presenta que ARGON obtuvo un mejor resultado porque el 75% de la efectividad del ARGON se puntúa por encima de 0.75, mientras que solo el 50% de la efectividad Ansible se puntúa por encima de 0.75. Del mismo

modo, los diagramas de caja para las muestras UPV2 y UPV3 indican que ARGON obtuvo mejores resultados al definir los recursos de infraestructura.

Estos resultados podrían explicarse por el hecho de que ARGON emplea un enfoque dirigido por modelos que permite a los participantes concentrarse en especificar la infraestructura de la nube en un nivel más alto de abstracción, en lugar de concentrarse en los estilos de código de los lenguajes de scripting como en el caso de Ansible. Además, debido a que los participantes no tenían conocimientos previos en computación en la nube, otra posibilidad es que ARGON podría permitir a los participantes mejorar su comprensión de los conceptos de la computación en la nube y de cómo se pueden definir los recursos de infraestructura.

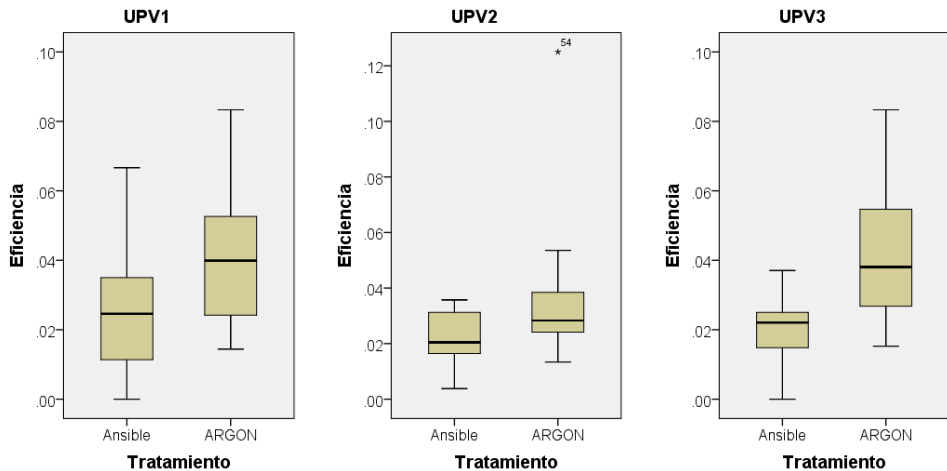


Figura 8-6 Diagramas de caja para la eficiencia del tratamiento

La Figura 8-6 presenta los diagramas de caja para la eficiencia del tratamiento de cada experimento. La muestra UPV1 muestra que ARGON obtuvo un mejor resultado porque el 75% de la eficiencia de ARGON se puntúa por encima de 0.024, mientras que solo el 50% de la eficiencia Ansible se puntúa por encima de 0.024. Del mismo modo, los diagramas de caja para las muestras UPV2 y UPV3 indican que ARGON obtuvo mejores resultados de eficiencia al especificar los recursos de infraestructura.

Dado que la eficiencia se calculó como la relación entre la efectividad y la duración de la tarea, una posibilidad para obtener una mejor eficiencia es que los participantes mejoren su efectividad cuando usan ARGON, y otra opción podría ser que los participantes especifiquen recursos de infraestructura más rápido cuando usan ARGON.

La Tabla 8-6 muestra que los valores de duración promedio varían de 23.42 a 27.53 minutos para ARGON y de 30 a 34.51 minutos para Ansible. Este resultado podría explicarse por el hecho de que ARGON abstrae la complejidad del uso de lenguajes de scripting a través de un lenguaje específico de dominio ArgonML para modelar los recursos de infraestructura de la nube.

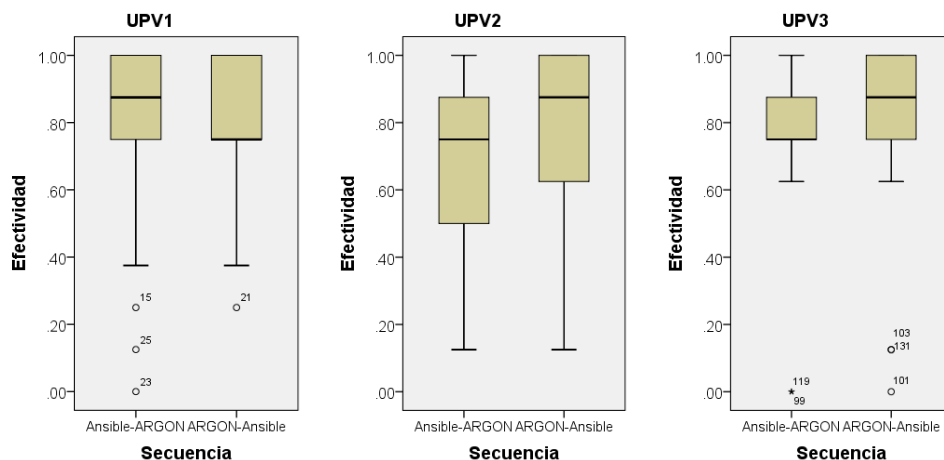


Figura 8-7 Diagramas de caja para la efectividad de la secuencia

La Figura 8-7 muestra los diagramas de caja para la efectividad de la secuencia de cada experimento. En este caso, los grupos experimentales son las secuencias, es decir, el orden en que los sujetos aplican los tratamientos (Vegas *et al.*, 2016). Los diagramas de caja muestran, en general, que ni la secuencia S1 (Ansible-ARGON) ni la secuencia S2 (ARGON-Ansible) tienen una diferencia significativa en términos de efectividad. Por ejemplo, los diagramas de caja para la efectividad de la secuencia de las muestras UPV1 y UPV3 muestran que tanto S1 como S2 tienen un 75% de efectividad con una puntuación superior a 0.75. Sin embargo, en el caso de la muestra UPV2, S2 obtuvo un mejor resultado que S1 porque el 75% de la efectividad de S2 se puntúa por encima de 0.62, mientras que el 75% de la efectividad de S1 se puntúa por encima de 0.50. Es importante mencionar que los participantes de la muestra UPV2 asistieron al curso sobre Desarrollo de Software Dirigido en Modelos. Por lo tanto, los resultados más altos de esta muestra podrían ser causados por la secuencia S2 que tiene el orden ARGON-Ansible y, por lo tanto, el resultado obtenido al usar ARGON podría haber motivado a los participantes a mejorar el resultado al usar Ansible. Sin embargo, la diferencia en la efectividad de la secuencia de muestra UPV2 es pequeña, y sería necesario realizar más experimentos con participantes que

tengan conocimientos previos sobre técnicas de desarrollo de software dirigidos por modelos para afirmar que existe una secuencia óptima.

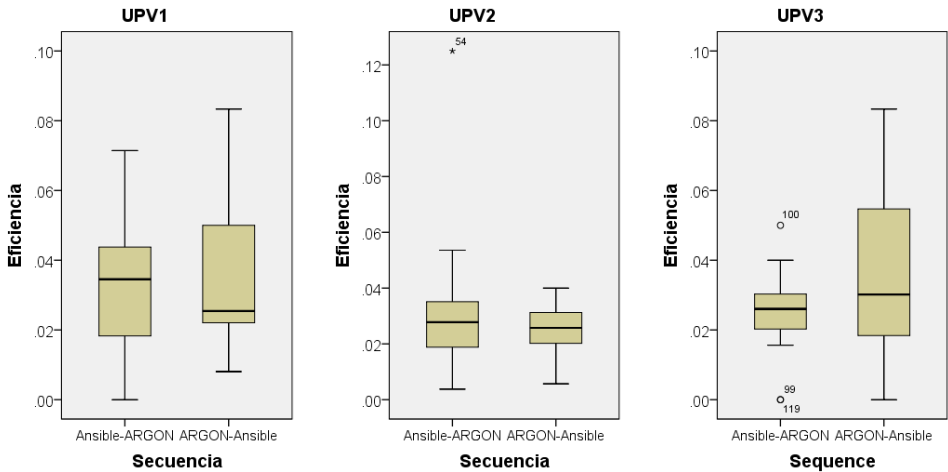


Figura 8-8 Diagramas de caja para la eficiencia de la secuencia

La Figura 8-8 muestra los diagramas de caja para la eficiencia de la secuencia de cada experimento. Los diagramas de caja presentan, en este caso, una ligera diferencia en la eficiencia de cada secuencia. Por ejemplo, el diagrama de caja para la eficiencia de la secuencia de la muestra UPV1 expone que la secuencia S1 (Ansible-ARGON) obtuvo un mejor resultado que la secuencia S2 (ARGON-Ansible) porque el 50% de la eficiencia de S1 se puntúa por encima de 0.034, mientras que el 50% de la eficiencia S2 se puntúa por encima de 0.025. Del mismo modo, las muestras UPV2 y UPV3 muestran resultados diferentes en sus diagramas de caja para la eficiencia de la secuencia.

Dado que la eficiencia es la relación entre la efectividad y la duración, el tiempo que lleva realizar las tareas experimentales podría haber afectado la eficiencia de cada secuencia. De la misma forma, la efectividad de los tratamientos podría haber afectado la eficiencia de cada secuencia.

Como se presenta en los diagramas de caja para la eficiencia de la secuencia, la eficiencia no tiene una comparación clara con respecto a las secuencias y, por lo tanto, es la razón por la que el diseño cruzado no centra su atención en la eficiencia para obtener un análisis estadístico de las secuencias (Vegas *et al.*, 2016).

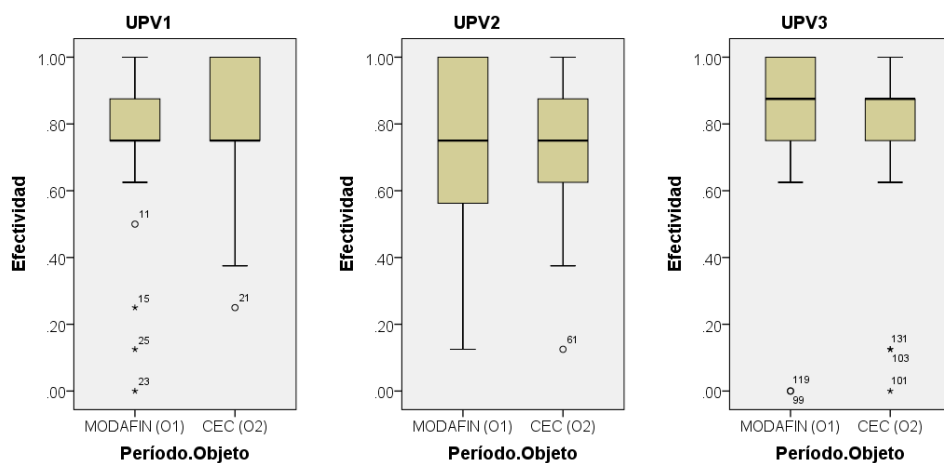


Figura 8-9 Diagramas de caja para la efectividad del período

La Figura 8-9 muestra los diagramas de caja para la efectividad del período de cada experimento. En este caso, un período se define mediante la aplicación de un tratamiento por un participante a un objeto experimental (Vegas *et al.*, 2016). Es importante mencionar que un período usa solo un objeto experimental a la vez. Los diagramas de caja de la Figura 8-9 muestran, en general, que ni el período MODAFIN (O1) ni el período CEC (O2) tienen mejor efectividad. Por ejemplo, los diagramas de caja para la efectividad del período de las muestras UPV1 y UPV3 muestran que tanto MODAFIN como CEC tienen un 75% de efectividad con una puntuación superior a 0.75. Sin embargo, en la muestra UPV2, parece que MODAFIN tiene un mejor resultado que CEC, pero tanto CEC como MODAFIN tienen un 50% de efectividad con una puntuación superior a 0.75. En consecuencia, ningún período tiene mejor efectividad que otro. Estos resultados también indican que los dos objetos experimentales tienen una complejidad similar y que podría no existir un efecto de aprendizaje.

La Figura 8-10 presenta los diagramas de caja para la eficiencia del período de cada experimento. Los diagramas de caja presentan, en este caso, una pequeña diferencia en la eficiencia de los períodos. Por ejemplo, el diagrama de caja para la eficiencia del período de la muestra UPV1 expone que el período CEC (O2) obtuvo un mejor resultado que el período MODAFIN (O1) porque el 50% de la eficiencia de CEC se puntúa por encima de 0.034, mientras que el 50% de la eficiencia de MODAFIN se puntúa por encima de 0.025. Del mismo modo, las muestras UPV2 y UPV3 muestran resultados diferentes en sus diagramas de caja para la eficiencia del período. Nuevamente, debido a que la eficiencia es la relación entre la efectividad y la duración de la tarea, el tiempo necesario para

realizar las tareas experimentales podría haber afectado la eficiencia de cada período. De la misma forma, la efectividad de los tratamientos podría haber afectado la eficiencia de cada período. Como resultado, la eficiencia no tiene una comparación clara con respecto a los períodos y, por lo tanto, es la razón por la cual el diseño cruzado no enfoca su atención en la eficiencia para obtener un análisis estadístico de los períodos (Vegas *et al.*, 2016).

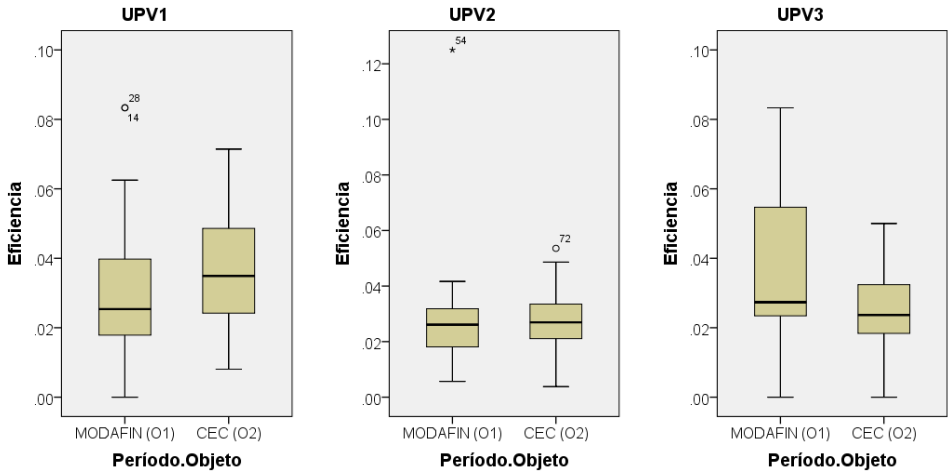


Figura 8-10 Diagramas de caja para la eficiencia del período

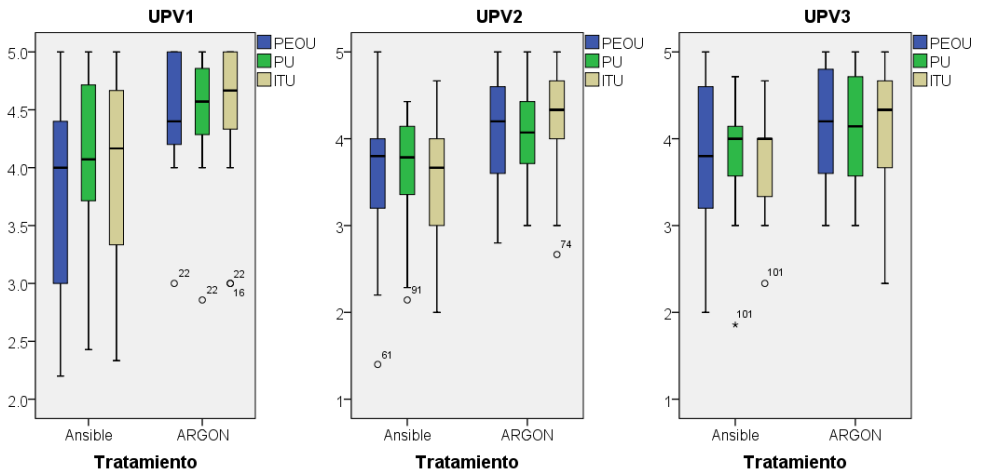


Figura 8-11 Diagramas de caja para las variables basadas en la percepción

La Figura 8-11 muestra los diagramas de caja para las variables basadas en la percepción de la familia de experimentos. La *mediana* de cada herramienta IaC se muestra como la línea horizontal en el diagrama de caja. Los diagramas de caja

muestran el juicio de los participantes después de usar cada tratamiento (Ansible y ARGON). Cada herramienta se evaluó en términos de su facilidad de uso percibida (PEOU) y su utilidad percibida (PU) para apoyar a los participantes en la definición de la infraestructura de la nube. Además, los participantes expresaron su intención de usar (ITU) estas herramientas en el futuro. En este caso, se utilizó una escala Likert de 5 puntos para medir estas variables y su valor neutral Likert se estableció en 3 puntos.

La Tabla 8-6 indica que tanto ARGON como Ansible tienen un valor *medio* por encima del valor neutral de la escala de medición y, por lo tanto, estas herramientas lograron buenos resultados en términos de la percepción de los participantes. Con respecto a la muestra UPV1, los resultados indican que los participantes expresaron mejores percepciones hacia ARGON (PEOU=4.4, PU=4.5 y ITU=4.6) que Ansible (PEOU=4.0, PU=4.0 y ITU=4.1) en términos de definición de la infraestructura. De manera similar, los participantes obtuvieron mejores resultados en las medidas de tendencia central con respecto a las variables basadas en la percepción al usar ARGON en las muestras UPV2 y UPV3, como se muestra en la Tabla 8-6.

En general, estos resultados sugieren que los participantes percibieron que ARGON es más fácil de usar y más útil que Ansible para especificar los recursos de infraestructura. Además, los participantes han expresado una mayor intención de utilizar ARGON en el futuro.

8.3.2 Prueba de hipótesis

Se utiliza el modelo lineal mixto (*Linear Mixed Model*, LMM) para realizar la prueba de hipótesis. De acuerdo con (Vegas *et al.*, 2016), el LMM incluye los siguientes términos: técnica (tratamiento), período (confundido con el objeto experimental) y secuencia (confundido con la interacción período*técnica y el carryover) como factores fijos, y los sujetos como un factor aleatorio anidado dentro de la secuencia .

Es importante verificar que los residuos del LMM tengan una distribución normal. En el caso de ausencia de normalidad, se utiliza la estrategia del logaritmo o el enfoque de dos pasos para transformar las variables continuas en normales (Templeton, 2011). Como el tamaño de la muestra es menor que 50, se aplica la prueba de Shapiro-Wilk para verificar si los residuos siguen una distribución normal. La Tabla 8-7 muestra los resultados de normalidad para las variables basadas en el rendimiento y en la percepción. Los residuos expresan que el LMM es válido, porque cumplen la condición de normalidad, es decir, la *sig.* es mayor que 0.05.

Tabla 8-7 Resumen de la prueba Shapiro-Wilk para los residuos del LMM

		Efect_r	Efic_r	PEOU_r	PU_r	ITU_r
UPV1	Sig.	0.999	0.211	0.999	0.825	0.532
UPV2	Sig.	0.170	1.000	0.299	0.084	0.465
UPV3	Sig.	0.982	0.106	0.259	0.161	0.580

De acuerdo con los resultados de las pruebas de efectos fijos que se muestran en la Tabla 8-8, la efectividad de la herramienta ARGON es significativamente diferente de la herramienta Ansible para todos los experimentos (UPV1=0.002, UPV2=0.001 y UPV3=0.018; son menores que 0.05). En base a la Tabla 8-6, Ansible es menos efectivo que ARGON considerando los valores medios de efectividad de los experimentos. Para la muestra UPV1, Ansible tiene una efectividad media de 0.648 mientras que ARGON tiene una efectividad media de 0.858. Las réplicas UPV2 y UPV3 tienen resultados similares en términos de efectividad. Como resultado, se puede rechazar las hipótesis nulas H_{10} para todos los experimentos. Esto significa que existe una diferencia estadística significativa entre las dos herramientas, que en términos de la efectividad en la definición de los recursos de infraestructura está a favor de ARGON.

Tabla 8-8 Prueba Tipo III de Efectos Fijos para la Efectividad

	UPV1	UPV2	UPV3
	Sig.	Sig.	Sig.
Período/Objeto	0.496	0.095	0.903
Técnica	0.002	0.001	0.018
Secuencia	0.742	0.970	0.758

Con respecto a la eficiencia, los resultados de las pruebas de efectos fijos que se muestran en la Tabla 8-9 corroboran que la eficiencia de la herramienta ARGON es significativamente diferente de la eficiencia de la herramienta Ansible en todos los experimentos (UPV1=0.000, UPV2=0.007 y UPV3=0.000) La Tabla 8-6 muestra los valores medios de eficiencia de los experimentos en los que se encuentra que Ansible es menos eficiente que ARGON. Por ejemplo, Ansible tiene una eficiencia media de 0.025, mientras que ARGON tiene una eficiencia media de 0.042 en UPV1. Además, se observa resultados similares en los experimentos UPV2 y UPV3. En consecuencia, se puede rechazar las hipótesis nulas H_{20} en todos los experimentos. Esto significa que existe una diferencia estadística significativa entre las dos herramientas IaC a favor de ARGON respecto al número de requisitos logrados en la unidad de tiempo.

Tabla 8-9 Prueba Tipo III de Efectos Fijos para la Eficiencia

	UPV1	UPV2	UPV3
	Sig.	Sig.	Sig.
Período/Objeto	0.051	0.149	0.110
Técnica	<0.001	0.007	<0.001
Secuencia	0.718	0.068	0.073

Es importante mencionar que los factores fijos (período y secuencia) no son significativos para la efectividad y ni para la eficiencia. Por un lado, que el período no sea estadísticamente significativo expresa que los objetos experimentales (MODAFIN y CEC) no afectan a las variables de respuesta (es decir, efectividad y eficiencia). Por otro lado, que la secuencia no sea estadísticamente significativa indica que no hay efecto de carryover entre los tratamientos y tampoco existen interacciones período*tratamiento. En general, los resultados indican que la diferencia observada de efectividad y eficiencia se debe a la herramienta IaC empleada.

Antes de aplicar el LMM en las variables basadas en la percepción, se utiliza la prueba Alfa de Cronbach para examinar la fiabilidad de cada cuestionario. El resultado de la prueba para todos los cuestionarios fue UPV1=0.962, UPV2=0.950 y UPV3=0.938, que son más altos que el nivel umbral 0.70 (Maxwell, 2002). La Tabla 8-10 muestra que los ítems utilizados para medir PEOU, PU e ITU obtuvieron un coeficiente Alfa de Cronbach de 0.928, 0.912 y 0.880 para el experimento de línea de base (UPV1), que son más altos que el nivel umbral. Las réplicas muestran resultados similares, lo que sugiere que el instrumento de la encuesta puede considerarse confiable.

Tabla 8-10 Resumen de las pruebas Alfa de Cronbach

	PEOU		PU		ITU	
	No	α	No	α	No	α
UPV1	5	0.938	7	0.912	3	0.880
UPV2	5	0.896	7	0.904	3	0.780
UPV3	5	0.894	7	0.886	3	0.732

Con respecto a las percepciones de los participantes sobre la facilidad de uso (PEOU), los resultados de las pruebas de efectos fijos que se muestran en la Tabla 8-11 confirman que la diferencia en las percepciones de la facilidad de uso entre las dos herramientas es significativamente diferente para todos los experimentos (UPV1=0.002, UPV2=0.037 y UPV3=0.0034). En este escenario, la Tabla 8-6 muestra los valores medios de PEOU de los experimentos, en donde

los sujetos percibieron que ARGON es más fácil de usar que Ansible. Por ejemplo, en el primer experimento (UPV1), Ansible tiene un valor medio de PEOU de 3.80, mientras que ARGON tiene un valor medio de PEOU de 4.49.

Los resultados obtenidos en la Tabla 8-6 respecto a los valores medios de PEOU para las replicaciones (UPV2 y UPV3) fueron similares al primer experimento (UPV1). Por lo tanto, se puede rechazar las hipótesis nulas H_{30} para todos los experimentos. Esto significa que los participantes percibieron que ARGON es más fácil de usar que Ansible en términos de definir la infraestructura de la nube. Además, el análisis de las respuestas a las preguntas abiertas en el cuestionario posterior al experimento reveló que los participantes expresaron que ARGON era fácil de usar: "*Modelar las características de la infraestructura es más fácil que escribir líneas de código*", "*Aunque no estaba familiarizado con la computación en la nube, pude definir una infraestructura en poco tiempo*".

Tabla 8-11 Prueba Tipo III de Efectos Fijos para la PEOU

	UPV1	UPV2	UPV3
	Sig.	Sig.	Sig.
Período/Objeto	0.511	0.317	0.330
Técnica	0.002	0.037	0.034
Secuencia	0.824	0.723	0.467

Con respecto a la utilidad percibida (PU), los resultados de las pruebas de efectos fijos presentados en la Tabla 8-12 indican que existe una diferencia estadísticamente significativa entre las dos herramientas en términos de la utilidad percibida por los participantes al aplicar cada herramienta (UPV1=0.007, UPV2=0.019 y UPV3=0.045). La Tabla 8-6 confirma que esta diferencia está a favor de ARGON, ya que la herramienta tiene un valor medio de PU de 4.53, mientras que Ansible tiene un valor medio de PU de 4.08 en el experimento de línea de base (UPV1). Resultados similares se obtuvieron en las replicaciones (UPV2 y UPV3). Como resultado, se pueden rechazar las hipótesis nulas H_{40} para todos los experimentos.

Tabla 8-12 Prueba Tipo III de Efectos Fijos para la PU

	UPV1	UPV2	UPV3
	Sig.	Sig.	Sig.
Período/Objeto	1.000	0.179	0.565
Técnica	0.007	0.019	0.045
Secuencia	0.764	0.630	0.768

Con respecto a la intención de uso (ITU), los resultados de la prueba de efectos fijos presentados en la Tabla 8-13 revelan que existe una diferencia estadísticamente significativa entre las dos herramientas para todos los experimentos (UPV1=0.032, UPV2=0.001 y UPV3=0.045). La Tabla 8-6 muestra que esta diferencia está a favor de ARGON. Por ejemplo, en el primer experimento (UPV1), ARGON tiene un valor medio de ITU de 4.53, mientras que Ansible tiene un valor medio de ITU de 4.06. Además, se obtuvieron resultados similares en las repeticiones (UPV2 y UPV3). Como resultado, se puede rechazar las hipótesis nulas H_{50} para todos los experimentos, lo que significa que los participantes percibieron que es más probable que se use ARGON en el contexto de esta familia de experimentos. Esto puede deberse a que, dado que son ingenieros de software noveles, perciben que la herramienta ARGON les proporciona mecanismos para llevar a cabo la definición de los recursos de infraestructura. Los participantes confirmaron esto en su respuesta a las preguntas abiertas del cuestionario posterior al experimento: “*Usaría esta herramienta para definir recursos de infraestructura en la nube ya que es intuitiva y los modelos permiten identificar la arquitectura y los componentes de la infraestructura rápidamente*”, “*Me gustaría utilizar ARGON porque es fácil de entender para modelar la infraestructura de la nube*”.

Tabla 8-13 Prueba Tipo III de Efectos Fijos para la ITU

	UPV1	UPV2	UPV3
	Sig.	Sig.	Sig.
Período/Objeto	0.942	0.740	0.565
Técnica	0.032	0.001	0.045
Secuencia	0.659	0.468	0.786

Una vez más, los factores fijos como el período y la secuencia para PEOU, PU e ITU no son significativos. En cuanto al hecho de que los períodos no son significativos, representa que los objetos experimentales (MODAFIN y CEC) no están afectando las percepciones de los participantes en términos de facilidad de uso, utilidad e intención de usar uno de los tratamientos. Además, las secuencias no son significativas, lo que significa que no existe el efecto de carryover entre los tratamientos y no hay interacciones período*tratamiento. En general, los resultados indican que las percepciones de los participantes se deben a la herramienta de aprovisionamiento empleada.

8.3.3 *Análisis de datos de la familia de experimentos*

En esta sección, se presenta un metaanálisis que agrega los hallazgos empíricos obtenidos en los experimentos individuales. Después se contesta las preguntas

de investigación establecidas para la familia de experimentos considerando los resultados obtenidos en cada experimento y en el metaanálisis.

8.3.3.1 Metaanálisis

Dado que la significación estadística (valor p) no es suficiente para afirmar la diferencia causada por los tratamientos, se utiliza el tamaño del efecto para medir la magnitud de esa diferencia. Debido a que se utiliza un diseño cruzado, el efecto de los tratamientos debe medirse si el período, la secuencia o cualquier variable de bloqueo no tienen relación y no hay carryover (Vegas *et al.*, 2016). En este contexto, la evidencia empírica muestra que los experimentos no tienen significancia estadística con respecto al período, secuencia y carryover. Además, el tamaño del efecto debe calcularse solo cuando el factor principal de los experimentos es la única variable estadísticamente significativa (Vegas *et al.*, 2016). En consecuencia, la familia de experimentos cumple los requisitos establecidos anteriormente porque solo los tratamientos tienen significancia estadística ($p < 0.05$). Por lo tanto, se debe calcular el tamaño del efecto para conocer su poder estadístico.

Tabla 8-14 Resumen de la d de Cohen para las variables dependientes de los experimentos

	Variable	Valor p	d de Cohen	Interpretación d de Cohen	Error Estándar	Intervalo de Confianza
UPV1	Efectividad	0.002	0.89	Grande	0.0712	0.14
	Eficiencia	<0.001	0.90	Grande	0.0057	0.01
	PEOU	0.002	0.98	Grande	0.2133	0.42
	PU	0.007	0.75	Medio	0.1786	0.35
	ITU	0.032	0.65	Medio	0.2163	0.42
UPV2	Efectividad	0.001	0.74	Medio	0.0667	0.13
	Eficiencia	0.007	0.72	Medio	0.0048	0.01
	PEOU	0.037	0.65	Medio	0.2044	0.40
	PU	0.019	0.80	Grande	0.1687	0.33
	ITU	0.001	1.10	Grande	0.1816	0.36
UPV3	Efectividad	0.018	0.81	Grande	0.0837	0.16
	Eficiencia	<0.001	1.40	Muy Grande	0.0049	0.01
	PEOU	0.034	0.47	Pequeño	0.2365	0.46
	PU	0.045	0.54	Medio	0.1934	0.38
	ITU	0.045	0.63	Medio	0.1879	0.37

En este estudio, se compara dos grupos en lugar de evaluar la fuerza de la asociación entre dos variables. Por lo tanto, el índice utilizado para comparar dos grupos es la d de Cohen. La Tabla 8-14 presenta un resumen de las medidas d de

Cohen para las variables dependientes de los experimentos. La primera columna muestra los conjuntos de datos correspondiente a cada experimento. La segunda columna detalla las variables dependientes con las que se mide la efectividad y la eficiencia de cada tratamiento, así como la facilidad de uso, la utilidad y la intención de uso percibida por los participantes. La tercera columna presenta los valores p para demostrar el cumplimiento del requisito para calcular el tamaño del efecto. La cuarta columna muestra el valor d de Cohen, mientras que la quinta columna presenta la interpretación del tamaño del efecto de la comparación de dos grupos. Cohen proporciona pautas para comparar diferentes tamaños de efectos, sugiriendo que los efectos pequeños, medianos y grandes se traducen en valores de d de 0.2, 0.5 y 0.8, respectivamente (Davey *et al.*, 2009). La sexta columna muestra el error estándar de las distribuciones de muestreo. Según (Ellis, 2012), todos los valores medios de una muestra se denominan distribución de la muestra, mientras que el error estándar es la desviación estándar de una distribución de la muestra. En este escenario, el error estándar es necesario para determinar los intervalos de confianza. Finalmente, la última columna presenta los intervalos de confianza, que se utilizan para combinar la ubicación y la precisión del nivel de tamaño del efecto. Por ejemplo, la efectividad de la muestra UPV1 tiene un valor d de Cohen de 0.89 y usando el intervalo de confianza correspondiente (0.14) tenemos 0.89 ± 0.14 . Como resultado, se obtiene un rango de 0.75 y 1.03, que corresponde a un efecto medio y un efecto grande, respectivamente.

Un metaanálisis es la opción más adecuada para evaluar y generalizar los resultados de una familia de experimentos. Según (Ellis, 2012), un metaanálisis es un análisis estadístico de los análisis estadísticos, que describe un conjunto de procedimientos para revisar sistemáticamente la investigación que examina un efecto particular y combina los resultados de estudios independientes para estimar el tamaño del efecto en la población. En este contexto, se utiliza un metaanálisis para combinar los valores d de Cohen y analizarlos para estimar el tamaño del efecto en la población objetivo, es decir, ingenieros de software noveles. El resultado del metaanálisis es un tamaño de efecto medio ponderado, que refleja el tamaño del efecto de la población objetivo con mayor precisión que cualquiera de los experimentos individuales (Ellis, 2012).

El metaanálisis se realizó con la herramienta RStudio versión 1.1.463 y el paquete Metaviz versión 0.3.0. La Figura 8-12 resume los resultados del metaanálisis utilizando un diagrama de bosque (*Forest Plot*). La primera columna muestra la etiqueta del conjunto de datos de cada experimento, así como la etiqueta de cada variable dependiente. La segunda columna muestra los valores de media y desviación estándar (*Standard Deviation*, SD) del tratamiento con ARGON. De la misma forma, la tercera columna muestra la media y la desviación estándar (SD)

del tratamiento con Ansible. La cuarta columna presenta el diagrama de bosque que resume los resultados del metaanálisis. El eje horizontal representa la escala para los tamaños de efecto, mientras que la línea vertical (0.0) representa la línea de efecto nulo. Cada línea horizontal en el diagrama de bosque representa los intervalos de confianza (*Confidence Intervals*, CI) del 95% para cada variable dependiente que se analiza. Cada cuadro negro es proporcional al tamaño del estudio, y su posición sobre el eje horizontal indica el valor d de Cohen. Cada diamante en el diagrama de bosque representa la estimación puntual y los intervalos de confianza cuando se combina y promedia los resultados de cada variable dependiente. En el metaanálisis, ningún estudio cruza la línea del efecto nulo y, por lo tanto, todos los tamaños de efectos muestran resultados estadísticamente significativos. Finalmente, la quinta columna indica el valor d de Cohen de cada variable dependiente junto con el intervalo de confianza (CI) del 95% entre paréntesis.

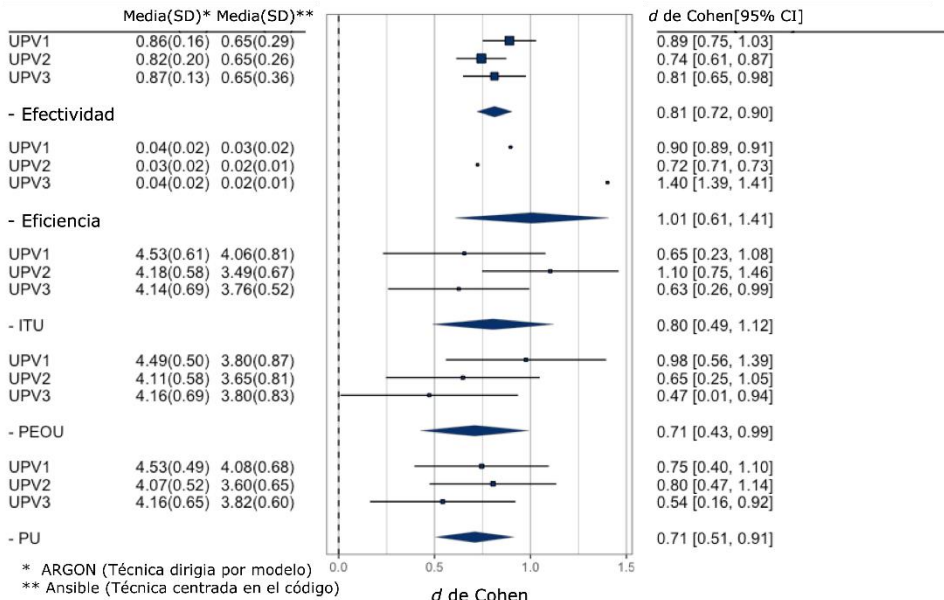


Figura 8-12 Diagrama de bosque para el metaanálisis de la familia de experimentos

8.3.3.2 Respondiendo las preguntas de investigación

Una vez que se han llevado a cabo los experimentos, se realiza un análisis global de los resultados para responder las preguntas de investigación. En este estudio, se obtiene evidencia empírica sobre cómo un enfoque dirigido por modelos (ARGON) puede ayudar a los ingenieros de software noveles a especificar los recursos de infraestructura, en lugar de una técnica centrada en el código (Ansible) que se usa ampliamente en entornos industriales. Esta evidencia

empírica, es una contribución a los hallazgos experimentales sobre la definición de los recursos de infraestructura de la nube, y proporciona datos reales sobre qué herramienta es más adecuada bajo ciertas condiciones para la definición de recursos de infraestructura.

Tabla 8-15 Resumen del resultado de las hipótesis de la familia de experimentos

	Experimento	Nº de Sujetos	Tipo de Sujetos	Hipótesis Confirmadas	Hipótesis No Confirmadas
UPV1	1.º Experimento	22	Posgrado	H1, H2, H3, H4, and H5.	
UPV2	2.º Experimento Réplica estricta del 1.º experimento (sujetos diferentes)	24	Pregrado	H1, H2, H3, H4, and H5.	
UPV3	3.º Experimento Réplica estricta del 1.º experimento (sujetos diferentes)	21	Pregrado	H1, H2, H3, H4, and H5.	

La Tabla 8-15 presenta un resumen de los resultados de cada hipótesis evaluada en la familia de experimentos. Los resultados de todos los experimentos, en términos de efectividad, fueron estadísticamente significativos ($p < 0.05$) para afirmar que ARGON es más efectivo que Ansible para definir los recursos de infraestructura de la nube. En este contexto, considerando las estadísticas descriptivas, los diagramas de caja para la efectividad del tratamiento sugirieron que ARGON obtuvo mejores resultados porque el 75% de la efectividad de ARGON es comparable con solo el 50% de la efectividad de Ansible. Con respecto a los diagramas de caja para la efectividad de la secuencia de las muestras UPV1 y UPV3, la secuencia S1 (Ansible-ARGON) y la secuencia S2 (ARGON-Ansible) tienen un 75% de efectividad puntuada por encima del percentil 25. Finalmente, los gráficos de caja para la efectividad del período de las muestras UPV1 y UPV3 muestran que MODAFIN (O1) y CEC (O2) tienen un 75% de efectividad por encima del percentil 25 y, en el caso de la muestra UPV2, MODAFIN (O1) y CEC (O2) tienen un 50% de efectividad por encima del percentil 50. En consecuencia, ARGON demostró ser más efectivo para especificar los recursos de infraestructura de la nube. Además, no hay evidencia para afirmar que un período es más efectivo que otro, así como tampoco hay

evidencia para afirmar que una secuencia es más efectiva que otra. En el caso de la muestra UPV2, la posible razón de que S2 sea más efectiva que S1 fue el conocimiento obtenido por los participantes sobre técnicas basadas en modelos en el curso de Desarrollo de Software Dirigido por Modelos.

Con respecto a la eficiencia, los resultados muestran que ARGON es más eficiente que Ansible para definir la infraestructura de la nube. Para realizar el análisis estadístico, fue necesario considerar el tiempo necesario para llevar a cabo las tareas experimentales. En este escenario, ARGON demostró ser más eficiente que Ansible, pero las secuencias y los períodos no tienen una comparación clara entre los experimentos. Esta es la razón por la cual el diseño cruzado de experimentos no centra su atención en la eficiencia para extraer análisis estadísticos y conclusiones (Vegas *et al.*, 2016).

Con respecto a las variables dependientes, tales como facilidad de uso percibida (PEOU), utilidad percibida (PU) e intención de uso (ITU), los resultados son estadísticamente significativos ($p < 0.05$) para afirmar que el ARGON es más fácil de usar y más útil para especificar la infraestructura de la nube. Además, los participantes indicaron su intención de usar ARGON en el futuro. En términos generales, las variables PEOU, PU e ITU de los tratamientos (es decir, ARGON y Ansible) tienen los valores medios por encima del valor neutral de Likert establecido en 3 puntos. En consecuencia, los participantes percibieron que ambas herramientas son fáciles de usar y útiles, y también han expresado su probable intención de usar estas herramientas. Sin embargo, debido a que el tiempo promedio empleado por los participantes al usar Ansible fue de 32 minutos, mientras que el tiempo promedio al usar ARGON fue de 21 minutos, podría ser el factor para decidir qué tratamiento percibieron los participantes como el mejor.

Con respecto a las preguntas de investigación (*Research Questions*, RQ) que motivan la familia de experimentos, las preguntas se responden utilizando los resultados empíricos para respaldar las afirmaciones.

RQ1. ¿Cuál herramienta IaC es más efectiva al definir la infraestructura de la nube, ARGON o Ansible?

Se encontró evidencia empírica para afirmar que ARGON es más efectiva que Ansible para especificar los recursos de infraestructura, es decir, la exactitud en la definición de los recursos de infraestructura de la nube fue superior con ARGON. Además, no existe pruebas de que los períodos o secuencias afectaron la efectividad de los tratamientos. En consecuencia, todos los experimentos son estadísticamente significativos en términos de efectividad para rechazar las hipótesis nulas H_{10} . El metaanálisis también confirmó que el coeficiente d de

Cohen para la variable de efectividad tiene una importancia práctica, con un tamaño de efecto medio para la muestra UPV2 y grande para las muestras UPV1 y UPV3. En general, los resultados muestran que, aunque Ansible fue diseñado para hacer que los entornos de infraestructura sean accesibles para cualquier persona con conocimientos básicos de técnicas y estructuras de codificación, el enfoque dirigido por modelos utilizado por ARGON ayudó a los participantes a obtener un modelo de infraestructura de nube más correcto.

RQ2. ¿Cuál herramienta IaC es más eficiente al definir la infraestructura de la nube, ARGON o Ansible?

Se encontró evidencia empírica para afirmar que ARGON es más eficiente que Ansible para especificar recursos de infraestructura. Todos los experimentos son estadísticamente significativos en términos de eficiencia para rechazar las hipótesis nulas H_{20} . El metaanálisis también confirmó que el coeficiente d de Cohen para la variable de eficiencia tiene una importancia práctica, con un tamaño de efecto medio, grande y muy grande para los conjuntos de datos UPV2, UPV1 y UPV3.

Los resultados de este estudio son alentadores y sugieren que es útil aplicar un enfoque dirigido por modelos en la definición de la infraestructura de nube. Sin embargo, se requiere más evidencia sobre el efecto del uso de ARGON en grandes especificaciones de infraestructura para determinar las implicaciones de costos a largo plazo. Además, se necesita más experimentación para estudiar el efecto de técnicas centradas en código versus dirigidas por modelos en la mantenibilidad a largo plazo de la definición de la infraestructura de la nube en un proceso DevOps.

RQ3. ¿Cuál herramienta IaC se percibe como más fácil de usar, ARGON o Ansible?

Se encontró evidencia empírica para afirmar que los participantes percibieron que ARGON es más fácil de usar que Ansible para definir la infraestructura de la nube. Este resultado puede deberse a que los participantes percibieron que el aprendizaje y el uso de un enfoque dirigido por modelos (ARGON) fue fácil. En consecuencia, todos los experimentos son estadísticamente significativos en términos de PEOU para rechazar las hipótesis nulas H_{30} . El metaanálisis también confirmó que el coeficiente d de Cohen para la variable PEOU tiene una importancia práctica, con un tamaño de efecto medio para la muestra UPV2 y grande para las muestras UPV1 y UPV3. En general, los resultados sugieren que el uso de modelos para definir la infraestructura de la nube mejoró la facilidad de uso percibida por los participantes de la herramienta IaC. De hecho,

los participantes podían definir correctamente los recursos de infraestructura con ARGON sin una amplia capacitación en modelado.

RQ4. ¿Cuál herramienta IaC se percibe como más útil, ARGON o Ansible?

Se encontró evidencia empírica para afirmar que los participantes percibieron que ARGON es más útil que Ansible para especificar la infraestructura de la nube. Este resultado podría deberse a que los participantes percibieron que ARGON fue eficaz para lograr su objetivo, es decir, definir los recursos de infraestructura. Por lo tanto, todos los experimentos son estadísticamente significativos en términos de PU para rechazar las hipótesis nulas H_{40} . El metaanálisis también confirmó que el coeficiente d de Cohen para la variable PU tiene una importancia práctica, con un tamaño de efecto medio para las muestras UPV1 y UPV3 y grande para la muestra UPV2. En general, los resultados sugieren que ARGON mitiga la complejidad de usar lenguajes de scripting y, por lo tanto, los participantes percibieron que ARGON es útil para definir los recursos de infraestructura de la nube.

RQ5. ¿Cuál herramienta IaC tienen la mayor intención de uso en el futuro, ARGON o Ansible?

Se encontró evidencia empírica para afirmar que los participantes tienen la intención de usar ARGON en el futuro. Este resultado puede deberse a que los participantes completaron las tareas experimentales en poco tiempo y se sintieron cómodos al usar ARGON para especificar la infraestructura de la nube. Como resultado, todos los experimentos son estadísticamente significativos en términos de ITU para rechazar las hipótesis nulas H_{50} . Además, el metaanálisis también confirmó que el coeficiente d de Cohen para la variable ITU tiene una importancia práctica, con un tamaño de efecto medio para las muestras UPV1 y UPV3 y grande para la muestra UPV2.

8.4 Amenazas a la validez

En esta sección, se utilizan las recomendaciones de (Wohlin *et al.*, 2012) para discutir los problemas que podrían haber amenazado la validez de la familia de experimentos.

8.4.1 Validez de la conclusión

Las amenazas a la validez de la conclusión se refieren a los problemas que afectan la capacidad de sacar la conclusión correcta (Wohlin *et al.*, 2012). Para lograr la confiabilidad de la implementación de los tratamientos, se proporcionó a los participantes una máquina virtual configurada con todas las herramientas necesarias para ejecutar la tarea experimental. Es importante mencionar que

todas las máquinas virtuales se configuraron con las mismas herramientas y espacio de trabajo. Además, se utilizó el diseño cruzado AB/BA para mitigar los problemas relacionados con el tamaño de muestras pequeñas y para aumentar la sensibilidad de cada experimento. Con respecto al poder estadístico, se utilizó la directriz propuesta por (Vegas *et al.*, 2016) donde el tamaño del efecto de los tratamientos solo debe medirse si el período, la secuencia o cualquier variable de bloqueo no tienen relación y no hay carryover. En este estudio, todos los experimentos cumplen con estos requisitos y, por lo tanto, se obtuvo un metaanálisis con resultados estadísticamente significativos. Además, para disminuir la amenaza de recopilación de datos, se aplicó los mismos procedimientos de extracción de datos a cada experimento y cada variable dependiente se calculó de manera consistente.

8.4.2 Validez interna

Las amenazas a la validez interna son influencias que pueden afectar a la variable independiente con respecto a la causalidad (Wohlin *et al.*, 2012). Para abordar las amenazas de validez interna, se debe tener en cuenta los factores que intervienen en el diseño cruzado, tales como el período, la secuencia, el *carryover* y los sujetos (participantes).

El período se confunde con el objeto experimental. En este caso, existe dos objetos experimentales: O1—MODAFIN y O2—ARGON. Cada objeto experimental consta de requisitos que los participantes deben utilizar para especificar los recursos de infraestructura de la nube. Los resultados afirman que no hay evidencia empírica de que uno de los períodos mejore los resultados experimentales más que otro.

La secuencia especifica el orden en que se aplicarán los tratamientos. En este caso, hay dos secuencias: S1 (ARGON-Ansible) y S2 (Ansible-ARGON). En general, no hay suficiente evidencia empírica para afirmar que una de las secuencias es mejor que otra, aunque, en la muestra UPV2, S2 parece ser mejor que S1. En este caso, el resultado obtenido al usar ARGON podría haber motivado a los participantes a mejorar el resultado al usar Ansible.

El *carryover* ocurre cuando se administra un tratamiento antes de que el efecto de otro tratamiento administrado previamente haya retrocedido por completo (Vegas *et al.*, 2016). Además, la interacción entre el tratamiento y el período se confunde intrínsecamente con el carryover y con el efecto de la secuencia y, en consecuencia, es imposible distinguir cuál de los tres está ocurriendo (Vegas *et al.*, 2016). Sin embargo, el período y la secuencia no tienen significación estadística en todos los experimentos y, por lo tanto, no hay carryover. Como resultado, los participantes no se vieron afectados por el carryover. Además, el

efecto de aprendizaje se mitigó mediante el uso de dos objetos experimentales para cada experimento.

Por otro lado, no hubo diferencias en la experiencia de los participantes ya que ninguno de ellos tenía conocimientos previos en computación en la nube. Además, se evitó el intercambio de información mediante el uso de diferentes objetos experimentales y monitoreando a los participantes durante los experimentos. Finalmente, para evaluar la comprensibilidad de los materiales, se realizó un estudio piloto para descubrir errores y corregirlos.

8.4.3 Validez del constructo

La validez del constructo se refiere a generalizar el resultado del experimento teniendo en cuenta el diseño del experimento, así como los problemas relacionados con el comportamiento de los participantes y los experimentadores (Wohlin *et al.*, 2012).

Las medidas utilizadas para obtener las variables cualitativas y cuantitativas pueden influir en la validez del constructo. Esto se mitigó mediante el uso de medidas que se aplican comúnmente en otros estudios empíricos de ingeniería de software. Por un lado, dado que se utilizó variables basadas en el rendimiento —como la efectividad y la eficiencia— para evaluar la definición de los recursos de infraestructura de la nube, se utilizó un conjunto de ocho requisitos con los que evaluar los dos tratamientos. En este escenario, un requisito se define correctamente si la infraestructura descrita satisface todo lo solicitado en ese requisito, independientemente del tratamiento (ANSIBLE y Ansible) utilizado. Por otro lado, se utilizó un cuestionario para medir las variables basadas en la percepción en términos de facilidad de uso percibida, utilidad percibida e intención de uso de cada tratamiento. Estos constructos se utilizan ampliamente para medir la percepción del participante y se basan en el modelo de aceptación de tecnología (Davis, 1989) (*Technology Acceptance Model*, TAM). La fiabilidad del cuestionario se evaluó mediante la prueba Alfa de Cronbach.

Finalmente, para evitar la preocupación por la evaluación, los participantes no fueron calificados según los resultados que obtuvieron. Sin embargo, los participantes ganaron un punto extra por participar en el experimento. Además, los participantes no estaban al tanto de las hipótesis experimentales para prevenir el sesgo en los tratamientos realizados.

8.4.4 Validez externa

Las amenazas a la validez externa son condiciones que limitan nuestra capacidad de generalizar los resultados de nuestro experimento a la práctica industrial (Wohlin *et al.*, 2012). La primera preocupación fue seleccionar grupos de

participantes que sean representativos de la población objetivo, es decir, desarrolladores de software y personal de operaciones. Los participantes elegidos eran estudiantes que no conocían la computación en la nube o las herramientas de IaC. En este caso, los estudiantes son la próxima generación de profesionales del software y, por lo tanto, están relativamente cerca de la población objetivo (Kitchenham *et al.*, 2002). En consecuencia, los estudiantes fueron seleccionados con el perfil de ingenieros de software noveles.

El tamaño y la complejidad de los objetos experimentales es una amenaza que podría afectar la validez externa. En este escenario, se utilizó una tarea experimental con una complejidad moderada porque el experimento requiere que los participantes completen la tarea asignada en un tiempo limitado (sesiones de 2 horas). Sin embargo, se considera que la tarea realizada por los participantes tiene una complejidad media-alta y, por lo tanto, puede considerarse una tarea representativa en un entorno práctico. Por otro lado, se seleccionó la herramienta Ansible como tratamiento de control porque es una herramienta IaC ampliamente utilizada en la industria (Geerling, 2015).

8.5 Conclusiones

En esta familia de experimentos, se consiguió evidencia empírica sobre cómo la herramienta ARGON puede ayudar a los ingenieros de software noveles a especificar los recursos de infraestructura en comparación con Ansible, una herramienta con un lenguaje de scripting ampliamente utilizada.

Esta evidencia empírica es una contribución al conjunto de conocimientos de IaC y MDE, porque proporciona datos reales sobre qué enfoque (dirigido por modelos o centrado en el código) es más adecuado bajo ciertas condiciones para apoyar la definición de la infraestructura de la nube. En particular, se encontró evidencia que respalda la afirmación de que un enfoque dirigido modelos (ARGON) es más efectivo que una técnica centrada en el código (Ansible) para especificar la infraestructura de la nube. Además, considerando el tiempo necesario para completar la tarea experimental, ARGON demostró ser más eficiente que Ansible. Es importante mencionar que los factores fijos como el período, la secuencia y el carryover no fueron estadísticamente significativos y, por lo tanto, estos factores no tuvieron influencia en los resultados del experimento. Por otro lado, se encontró evidencia que respalda la afirmación de que los participantes percibieron que ARGON es más fácil de usar y más útil que Ansible para especificar los recursos de infraestructura. Además, los participantes expresaron su intención de usar ARGON en el futuro. Sin embargo, se necesitan más repeticiones para confirmar o refutar estos resultados.

Los hallazgos también tienen implicaciones prácticas. Se descubrió que el tiempo de modelado requerido para especificar los recursos de infraestructura con ARGON es menor que el tiempo necesario para escribir un script para Ansible. En este contexto, ARGON también demostró ser más efectivo que Ansible para definir los recursos de infraestructura. En consecuencia, ARGON es una herramienta de modelado de infraestructura prometedora, que puede mejorar el tiempo en las tareas de aprovisionamiento de la infraestructura.

Es importante tener en cuenta que la familia de experimentos presenta resultados preliminares sobre la efectividad de ARGON (y de su lenguaje ArgonML) para modelar la infraestructura. Aunque los resultados son prometedores, estos resultados deben interpretarse con precaución, ya que solo son válidos dentro del contexto establecido en esta familia de experimentos. Es necesario verificar si los resultados se mantienen al utilizar objetos experimentales más complejos o si los profesionales con experiencia en la computación en la nube están involucrados en la experimentación. No obstante, este estudio tiene valor como la primera familia de experimentos utilizados para evaluar la efectividad de las herramientas para apoyar el enfoque de IaC.

Capítulo 9

Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones obtenidas como resultado del diseño y la investigación del artefacto MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*) en un contexto de aplicación. Además, se presentan los proyectos previstos a realizar como próximos pasos derivados de esta tesis.

En la Sección 9.1 se presentan las conclusiones del trabajo de investigación mediante las respuestas a las preguntas de investigación que guiaron esta tesis y el cumplimiento de los objetivos de investigación.

En la Sección 9.2 se presentan los trabajos futuros que se derivan de esta tesis como oportunidades de investigación.

9.1 Conclusiones

En esta tesis se ha presentado una solución a la diversidad de los lenguajes de scripting —de las herramientas IaC (*Infrastructure as Code*)— junto con la heterogeneidad del tipo de infraestructura que ofrece cada proveedor de servicios IaaS (*Infrastructure as a Service*) respecto al aprovisionamiento de infraestructura en la nube. Primero, se presentó MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*) para soportar IaC mediante la *abstracción* y *automatización* del aprovisionamiento de infraestructura utilizando la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering*, MDE). Luego, se desarrolló el lenguaje ArgonML (*ARGON Modeling Language*) para *abstraer* la especificación de los recursos de infraestructura y la herramienta ARGON (*An infrastruCTure modelinG tool for clOud provisioniNg*) para *automatizar* el aprovisionamiento de infraestructura propuesto por MoCIP. Finalmente, se realizaron experimentos controlados para evaluar el rendimiento y la percepción de los usuarios al utilizar MoCIP y la herramienta ARGON.

Por otro lado, se utilizó la metodología *Design Science* como soporte conceptual al procedimiento que se empleó para realizar esta tesis. Primero, se diseñó el artefacto MoCIP y la herramienta ARGON con su lenguaje ArgonML como instrumentos MDE. Luego, se investigó la interacción de MoCIP y ARGON en el contexto del aprovisionamiento de infraestructura en la nube. En consecuencia, en este capítulo se presentan las respuestas a las preguntas de investigación (*Research Question*, RQ) en donde se especifica el cumplimiento de los objetivos de investigación respecto al diseño y la investigación de MoCIP — y su herramienta ARGON— en un contexto de aplicación.

9.1.1 *Objetivo de conocimiento sobre IaC y MDE*

Un objetivo de conocimiento describe los fenómenos a estudiar (Wieringa, 2014). En este caso, el objetivo es *conocer los trabajos de investigación que realizan el aprovisionamiento de infraestructura en la nube*. Sin embargo, para delimitar el alcance del objetivo se plantea la siguiente pregunta de investigación.

- **RQ1.** *¿Cuáles son los trabajos de investigación relacionados con el aprovisionamiento de infraestructura en la nube que utilizan IaC y MDE?*

La respuesta a la pregunta de investigación presenta una revisión de los trabajos de investigación relacionados con el aprovisionamiento de infraestructura en la nube que utilizan la Infraestructura como Código (IaC) y la Ingeniería de Software Dirigida por Modelos (MDE). Luego de realizar una revisión de la literatura científica se seleccionó quince trabajos relacionados que investigan IaC y MDE.

El principal problema que abordan todos los trabajos relacionados con IaC y MDE es la diversidad de los lenguajes de scripting que utilizan las herramientas IaC. Por otro lado, tres trabajos abordan la heterogeneidad de las tecnologías entre los proveedores de servicios en la nube y la interoperabilidad entre las soluciones de nube existentes. En cambio, una publicación afronta la integración de las diferentes fases del ciclo de vida de los servicios en la nube, mientras que otra publicación aborda el escalado automático de servicios entre múltiples plataformas en la nube. Finalmente, un trabajo de investigación enfrenta la falta de metodologías para obtener ventajas de las funciones como servicio en la nube.

Con respecto a las soluciones propuestas, diez publicaciones proponen utilizar un lenguaje de dominio específico (*Domain-Specific Language*, DSL) para modelar el despliegue de aplicaciones en la nube. En cambio, dos trabajos relacionados utilizan perfiles UML para modelar, analizar e implementar los sistemas en la nube. Por otro lado, una publicación plantea la integración de la gestión de la configuración y la implementación de los servicios en la nube, mientras que otra publicación propone un marco (*framework*) para escalar las aplicaciones en la nube. Finalmente, dos trabajos proponen un mapeo y la traducción desde los recursos de la nube definidos con el estándar TOSCA hacia herramientas de la comunidad DevOps.

En cuanto a las técnicas MDE, doce trabajos relacionados utilizan modelos para representar en un alto nivel de *abstracción* elementos, tales como máquinas virtuales, contenedores, aplicaciones y configuraciones. En cambio, dos publicaciones utilizan modelos en tiempo de ejecución (*model@run-time*) para representar con modelos los sistemas en ejecución. Por otro lado, siete publicaciones utilizan transformaciones de modelo a texto para convertir modelos del estándar TOSCA en scripts para herramientas IaC. Finalmente, cuatro trabajos relacionados utilizan y adaptan la arquitectura dirigida por modelos (*Model-Driven Architecture*, MDA) para definir en diferentes niveles de abstracción elementos, tales como máquinas virtuales, contenedores y aplicaciones que serán desplegadas en la nube.

En conclusión, los trabajos relacionados con IaC y MDE han centrado sus esfuerzos en mejorar el despliegue de aplicaciones en la nube. Esto ocasionó una brecha que cubrir en el aprovisionamiento de infraestructura en nube. No se ha encontrado una solución para *abstraer* la complejidad de especificar los recursos de infraestructura de diversos proveedores de servicios IaaS, así como no se ha encontrado una solución para *automatizar* el aprovisionamiento de la infraestructura en la nube. Además, no se han encontrado estudios empíricos que evalúen el rendimiento de una persona al definir un script IaC y tampoco se evalúa la percepción de los usuarios después de utilizar una herramienta IaC.

9.1.2 *Objetivo de diseño del artefacto MoCIP*

El objetivo es diseñar *MoCIP: A Model-Driven Approach to Cloud Infrastructure Provisioning*. El objetivo plantea un conjunto de desafíos que se derivan en la siguiente pregunta de investigación:

- **RQ2.** *¿Cómo diseñar el artefacto MoCIP utilizando IaC y MDE para realizar el aprovisionamiento de infraestructura en la nube?*

Como respuesta a la pregunta de investigación se propone MoCIP, un enfoque dirigido por modelos para el aprovisionamiento de infraestructura en la nube. MoCIP soporta la Infraestructura como Código (IaC) mediante la Ingeniería de Software Dirigida por Modelos (MDE).

MoCIP utiliza los principios de *abstracción* y *automatización* de MDE. En este caso, se desarrollaron dos instrumentos MDE que ayudan a responder la pregunta de investigación y a la consecución de su objetivo.

En primer lugar, se desarrolló el lenguaje específico de dominio ArgonML para *abstraer* las capacidades de la computación en la nube, tales como cómputo, almacenamiento, redes y elasticidad. El lenguaje ArgonML permite modelar los recursos de infraestructura de la nube.

En segundo lugar, se desarrolló la herramienta ARGON para *automatizar* el aprovisionamiento de infraestructura en la nube. Por un lado, ARGON realiza transformaciones de modelo a modelo para generar modelos que representan la infraestructura de diferentes proveedores de servicios IaaS. Por otro lado, ARGON realiza transformaciones de modelo a texto para generar scripts IaC con la información subyacente del proveedor de servicios IaaS y de la herramienta de aprovisionamiento de infraestructura.

El objetivo de diseñar un artefacto se ha cumplido en su totalidad. A lo largo de este proyecto de doctorado, se ha presentado MoCIP para soportar IaC en el aprovisionamiento de infraestructura en la nube. Además, se utiliza MDE para abstraer y automatizar el aprovisionamiento de infraestructura en la nube con la herramienta ARGON y su lenguaje ArgonML.

9.1.3 *Objetivo de conocimiento sobre la investigación de MoCIP*

El objetivo es *conocer la percepción de los usuarios al utilizar MoCIP en un contexto de aplicación*. Para lograr el objetivo se plantea la siguiente pregunta de investigación.

- **RQ3.** *¿Cuál es la percepción de los usuarios al utilizar MoCIP en su contexto de aplicación?*

La respuesta a la pregunta de investigación manifiesta la percepción de los usuarios al utilizar MoCIP en el aprovisionamiento de infraestructura en la nube.

Sin embargo, el artefacto MoCIP tiene instrumentos MDE para abstraer y automatizar el aprovisionamiento de infraestructura. En consecuencia, para alcanzar el objetivo de conocimiento, se plantean objetivos de predicción. En este caso, un objetivo de predicción intenta predecir cómo será la interacción de un artefacto (o su instrumento) con el contexto de un problema.

En primer lugar, se plantea un objetivo para *predecir la percepción de los usuarios al utilizar MoCIP*. En este caso, el objetivo de predicción se deriva en preguntas de investigación, que a su vez se derivan de la pregunta RQ3 y que motivaron el diseño y la ejecución de dos experimentos controlados.

RQ3.1. *¿MoCIP se percibe como fácil de usar?*

Se encontró evidencia empírica para afirmar que los participantes de los experimentos percibieron que MoCIP es fácil de usar para realizar el aprovisionamiento de infraestructura en la nube. Este resultado puede deberse a que los participantes percibieron que el aprendizaje y el uso de MoCIP —un enfoque dirigido por modelos— fue fácil.

RQ3.2. *¿MoCIP se percibe como útil?*

Se encontró evidencia empírica para afirmar que los participantes de los experimentos percibieron que MoCIP es útil para realizar el aprovisionamiento de infraestructura en la nube. Este resultado podría deberse a que los participantes percibieron que MoCIP fue eficaz para lograr su objetivo, es decir, aprovisionar la infraestructura en la nube.

RQ3.3. *¿Existe la intención de usar MoCIP en el futuro?*

Se encontró evidencia empírica para afirmar que los participantes de los experimentos tienen la intención de usar MoCIP en el futuro. Este resultado puede deberse a que los participantes completaron la tarea experimental y se sintieron cómodos al usar MoCIP para aprovisionar la infraestructura en la nube.

En segundo lugar, se plantea un objetivo para *predecir el rendimiento y la percepción de los usuarios al comparar ARGON versus Ansible*. Por un lado, ARGON es una herramienta IaC dirigida por modelos que proporciona el lenguaje ArgonML para modelar los recursos de la infraestructura. Por otro lado, Ansible es una herramienta IaC centrada en el código que es ampliamente utilizada en la industria. Es importante mencionar que ARGON y Ansible son herramientas IaC para el aprovisionamiento de infraestructura. En este caso, el objetivo de predicción se deriva en un conjunto de preguntas de investigación, que a su vez se derivan de la pregunta RQ3 y que motivaron el diseño y ejecución de una familia de tres experimentos controlados.

RQ3.4. *¿Cuál herramienta IaC es más efectiva al definir la infraestructura de la nube, ARGON o Ansible?*

Se encontró evidencia empírica para afirmar que ARGON es más efectivo que Ansible para especificar los recursos de infraestructura, es decir, la exactitud en la definición de los recursos de infraestructura de la nube fue superior con ARGON. En general, los resultados muestran que, aunque Ansible fue diseñado para hacer que los entornos de infraestructura sean accesibles para cualquier persona con conocimientos básicos en técnicas y estructuras de codificación, el enfoque dirigido por modelos utilizado por ARGON ayudó a los participantes a obtener un modelo de infraestructura de nube más correcto.

RQ3.5. *¿Cuál herramienta IaC es más eficiente al definir la infraestructura de la nube, ARGON o Ansible?*

Se encontró evidencia empírica para afirmar que ARGON es más eficiente que Ansible para especificar recursos de infraestructura. En este caso, la eficiencia es la relación entre la efectividad y el tiempo empleado para definir la infraestructura de la nube. En consecuencia, este resultado puede deberse a que la definición de los recursos de la infraestructura fue más efectiva (correcta) y en menor tiempo al utilizar ARGON. En general, los resultados de los experimentos son alentadores y sugieren que es útil aplicar un enfoque dirigido por modelos (ARGON) en la definición de la infraestructura.

RQ3.6. *¿Cuál herramienta IaC se percibe como más fácil de usar, ARGON o Ansible?*

Se encontró evidencia empírica para afirmar que los participantes percibieron que ARGON es más fácil de usar que Ansible para definir la infraestructura de la nube. Este resultado puede deberse a que los participantes percibieron que el aprendizaje y el uso de un enfoque dirigido por modelos (ARGON) fue fácil. En general, los resultados sugieren que el uso de modelos para definir la infraestructura de la nube mejoró la facilidad de uso percibida por los participantes.

RQ3.7. *¿Cuál herramienta IaC se percibe como más útil, ARGON o Ansible?*

Se encontró evidencia empírica para afirmar que los participantes percibieron que ARGON es más útil que Ansible para especificar la infraestructura de la nube. Este resultado podría deberse a que los participantes percibieron que ARGON fue eficaz para lograr su objetivo, es decir, definir los recursos de infraestructura. En general, los resultados sugieren que ARGON mitiga la complejidad de usar lenguajes de scripting y, por lo tanto, los participantes percibieron que ARGON es útil para definir la infraestructura de la nube.

RQ3.8. *¿Cuál herramienta IaC tienen la mayor intención de uso en el futuro, ARGON o Ansible?*

Se encontró evidencia empírica para afirmar que los participantes tienen la intención de usar ARGON en el futuro. Este resultado puede deberse a que los participantes completaron las tareas experimentales en poco tiempo y se sintieron cómodos al usar ARGON para especificar la infraestructura de la nube.

En conclusión, el objetivo de conocer la percepción de los usuarios al utilizar MoCIP en un contexto de aplicación se ha cumplido en su totalidad.

En primer lugar, se logró el objetivo de predecir la percepción de los usuarios al utilizar MoCIP mediante el resultado de dos experimentos. En este caso, los experimentos respaldan la afirmación de que los participantes percibieron que MoCIP es fácil de usar y útil para realizar el aprovisionamiento de la infraestructura en la nube. Además, los participantes tienen la intención de usar MoCIP en el futuro.

En segundo lugar, se logró el objetivo de predecir el rendimiento y la percepción de los usuarios al comparar ARGON versus Ansible mediante una familia de experimentos. Por un lado, se obtuvo evidencia empírica para afirmar que ARGON es más efectivo y eficiente que Ansible respecto a definir la infraestructura de la nube. Por otro lado, se obtuvo evidencia empírica para afirmar que los participantes percibieron que ARGON es más fácil de usar y útil que Ansible para definir la infraestructura de la nube. Además, los participantes tienen la intención de utilizar ARGON en el futuro.

9.2 Trabajos futuros

Los resultados de la investigación presentados en esta tesis han abierto una serie de trabajos futuros en diferentes áreas.

9.2.1 Ampliar el soporte de MoCIP

En esta tesis, se presentó MoCIP como un soporte dirigido por modelos para el aprovisionamiento de infraestructura en la nube. En este caso, MoCIP se centra en la computación en la nube para soportar una *infraestructura* flexible y escalable con un acceso a sus recursos en tiempo real. Sin embargo, nuevas tecnologías como el Internet de las cosas (*Internet of Things*, IoT) y la computación cercana a las fuentes de datos necesitan soporte respecto al aprovisionamiento de sus recursos de infraestructura. En consecuencia, se propone ampliar el soporte a la computación de borde (*Edge Computing*) para gestionar el aprovisionamiento de los recursos de infraestructura que procesan los datos de dispositivos inteligentes IoT, tales como sensores, actuadores, entre otros. Además, se plantea soportar

el aprovisionamiento de la infraestructura cercana a la fuente de los datos con el fin de mitigar problemas, tales como un bajo ancho de banda y una mínima transmisión de datos y, por lo tanto, reducir la latencia de las conexiones a la computación en la nube.

9.2.2 Ampliar el soporte de las herramientas MDE

En cuanto a las herramientas de soporte MDE, se abordará el cálculo del coste de la infraestructura —a ser aprovisionada en un proveedor de servicios IaaS— en base a los modelos de infraestructura generados por ARGON. Los proveedores públicos de servicios IaaS ofrecen el sistema de pago por el uso de sus recursos de infraestructura. En consecuencia, a partir de un modelo infraestructura se propone realizar el cálculo del coste de la infraestructura para diferentes proveedores de servicios IaaS con el fin de obtener la mejor opción y reducir o minimizar los costes.

Por otro lado, se desea ampliar la herramienta ARGON para que soporte la tecnología de contenedores. En este caso, los proveedores de servicios IaaS ofrecen recursos de infraestructura optimizados para utilizar contenedores, tales como Docker (Hykes, 2013). En consecuencia, se propone abstraer y automatizar el proceso de aprovisionamiento de contenedores en la nube.

9.2.3 Mantenibilidad de los metamodelos de infraestructura

El lenguaje ArgonML utiliza metamodelos de infraestructura para definir su lenguaje de modelado. Por lo tanto, los metamodelos permiten describir todos los modelos que ArgonML puede representar. En cambio, ARGON utiliza los metamodelos de infraestructura para generar modelos —mediante transformaciones de modelo a modelo— que representan la infraestructura de diferentes proveedores de servicios IaaS. En consecuencia, los metamodelos son la *pedra angular* de ARGON y ArgonML.

Debido a que cada metamodelo de infraestructura *abstrae* las capacidades de la nube —tales como cómputo, almacenamiento, redes y elasticidad— de cada proveedor de servicios IaaS, el cambio de un servicio de infraestructura de un proveedor de la nube tiene un impacto en su respectivo metamodelo. Por lo tanto, se propone investigar la extensibilidad de los metamodelos de infraestructura con el fin de facilitar su mantenibilidad.

9.2.4 Pruebas a los modelos de infraestructura

MoCIP y su herramienta ARGON utilizan modelos para representar en un alto nivel de abstracción la infraestructura de diversos proveedores de servicios IaaS. En este caso, a partir de los modelos se genera scripts IaC para crear la

infraestructura en la nube. En este caso, se propone realizar la validación de los modelos de infraestructura con el fin de encontrar y solucionar algún problema respecto a la creación y/o actualización de la infraestructura, así como solucionar los problemas con la idempotencia de los scripts IaC con fin de ejecutar varias veces cada script IaC y obtener *siempre* el resultado deseado. En consecuencia, los recursos de infraestructura de un modelo pueden ser validados en tiempo de diseño con el fin de obtener un mensaje de error adecuado para el dominio.

9.2.5 Implantación de MoCIP

El proyecto de doctorado utilizó la metodología *Design Science* (Wieringa, 2014) para guiar el diseño y la investigación del artefacto MoCIP en un contexto de aplicación. Primero, se investigó los fenómenos que hay que mejorar en el contexto del aprovisionamiento de infraestructura. Luego, se presentó el diseño del artefacto MoCIP para tratar el aprovisionamiento de infraestructura en la nube. Finalmente, se realizó la validación empírica para evaluar si MoCIP mejoró el aprovisionamiento de infraestructura en la nube.

Por lo tanto, se propone la implantación de MoCIP en un contexto industrial práctico con empresas para tratar el problema del aprovisionamiento de infraestructura. En este caso, se plantea que expertos en computación en la nube y en aprovisionamiento de infraestructura utilicen MoCIP y su herramienta ARGON en un proyecto de infraestructura en un contexto industrial práctico. Luego, se planifica evaluar si la utilización de MoCIP y su herramienta ARGON han sido exitoso en el aprovisionamiento de infraestructura en el contexto industrial con empresas.

9.2.6 Experimentos controlados

Con el fin de evaluar MoCIP y su herramienta ARGON en el contexto del proceso de aprovisionamiento de infraestructura, se realizaron experimentos controlados. Por un lado, se evaluó la percepción de los participantes respecto a la utilización de MoCIP en el aprovisionamiento de infraestructura. Por otro lado, se evaluó el rendimiento y la percepción de los participantes respecto a la comparación de ARGON versus Ansible en la definición de la infraestructura.

Sin embargo, es necesario realizar más réplicas de los experimentos teniendo como sujetos a profesionales en computación en la nube de la industria y otros sectores que permitan validar la solución desde diferentes puntos de vista de usuarios a través de casos de estudio más complejos.

Finalmente, se buscará evaluar la usabilidad/experiencia del usuario en cuanto a la herramienta ARGON y su lenguaje ArgonML con el fin de mejorar el modelado de los recursos de infraestructura.

Bibliografía

- Abrahão, S., Insfran, E., Carsí, J. A., & Genero, M. (2011). Evaluating requirements modeling methods based on user perceptions: A family of experiments. *Information Sciences*, 181(16), pp. 3356–3378. <https://doi.org/10.1016/j.ins.2011.04.005>
- Acerbis, R., Bongio, A., Brambilla, M., Tisi, M., Ceri, S., & Tosetti, E. (2007). Developing eBusiness Solutions with a Model Driven Approach: The Case of Acer EMEA. *Web Engineering*, (Mdd), pp. 539–544. https://doi.org/10.1007/978-3-540-73597-7_51
- Adams, B., & McIntosh, S. (2016). Modern Release Engineering in a Nutshell -- Why Researchers Should Care. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 78–90. <https://doi.org/10.1109/saner.2016.108>
- Aiftimiei, C., Costantini, A., Bucchi, R., Italiano, A., Michelotto, D., Panella, M., Pergolesi, M., Saletta, M., Traldi, S., Vistoli, C., Zizzi, G., & Salomoni, D. (2017). Cloud Environment Automation: From infrastructure deployment to application monitoring. *Journal of Physics: Conference Series*, 898(8). <https://doi.org/10.1088/1742-6596/898/8/082016>
- Alipour, H., & Liu, Y. (2018). Model Driven Deployment of Auto-Scaling Services on Multiple Clouds. *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 93–96. <https://doi.org/10.1109/ICSA-C.2018.00033>
- Amazon. (2006). Amazon Web Services. Retrieved November 7, 2019, from https://en.wikipedia.org/wiki/Amazon_Web_Services
- Amazon Web Services. (2011). CloudFormation. Retrieved August 29, 2019, from <https://aws.amazon.com/cloudformation/>
- Apache Software Foundation. (2009a). Deltacloud. Retrieved October 21, 2019, from <http://deltacloud.apache.org/>
- Apache Software Foundation. (2009b). Libcloud. Retrieved October 21, 2019, from <http://libcloud.apache.org/>
- Apache Software Foundation. (2011). jclouds. Retrieved October 21, 2019, from <http://jclouds.apache.org/>
- Artac, M., Borovsak, T., Di Nitto, E., Guerriero, M., Perez-Palacin, D., & Tamburri, D. A. (2018). Infrastructure-as-Code for Data-Intensive Architectures: A Model-Driven Development Approach. *2018 IEEE International Conference on Software Architecture (ICSA)*, pp. 156–165. <https://doi.org/10.1109/ICSA.2018.00025>
- Artač, M., Borovšak, T., Di Nitto, E., Guerriero, M., & Tamburri, D. A. (2016). Model-Driven continuous deployment for quality devops. *2016 International Workshop on Quality-Aware DevOps (QUDOS)*, pp. 40–41. <https://doi.org/10.1145/2945408.2945417>

- Artac, M., Borovssak, T., Di Nitto, E., Guerriero, M., & Tamburri, D. A. (2017). DevOps: Introducing infrastructure-as-code. *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 497–498. <https://doi.org/10.1109/ICSE-C.2017.162>
- Barker, W. C. (2003). Guideline for Identifying an Information System as a National Security System. *NIST Special Publication*.
- Baset, S., Suneja, S., Bila, N., Tuncer, O., & Isci, C. (2017). Usable declarative configuration specification and validation for applications, systems, and cloud. *Middleware 2017 - Proceedings of the 2017 International Middleware Conference (Industrial Track)*, (Cv1), pp. 29–32. <https://doi.org/10.1145/3154448.3154453>
- Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). The goal question metric approach. In *Encyclopaedia of Software Engineering*. <https://doi.org/10.1.1.104.8626>
- Basili, V. R., Shull, F., & Lanubile, F. (1999). Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering*, 25(4), pp. 456–473.
- Baum, M., Das, R., & Swan, E. (2003). Splunk. Retrieved September 13, 2019, from <https://www.splunk.com/>
- Beary, W. (2012). fog - The Ruby cloud services library. Retrieved October 21, 2019, from <http://fog.io/>
- Bhattacharjee, A., Barve, Y., Gokhale, A., & Kuroda, T. (2018). CloudCAMP: Automating the deployment and management of cloud services. *2018 IEEE International Conference on Services Computing (SCC)*, pp. 237–240. <https://doi.org/10.1109/SCC.2018.00038>
- Brabra, H., Mtibaa, A., Gaaloul, W., Benatallah, B., & Gargouri, F. (2019). Model-Driven Orchestration for Cloud Resources. *2019 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 422–429. <https://doi.org/10.1109/cloud.2019.00074>
- Brambilla, M., Cabot, J., & Wimmer, M. (2017). Model-Driven Software Engineering in Practice. In *Synthesis Lectures on Software Engineering* (Second Edition, Volume 1). <https://doi.org/10.2200/S00441ED1V01Y201208SWE001>
- Brikman, Y. (2017). *Terraform: Up and Running* (First Edit). O'Reilly Media.
- Buyya, R., Broberg, J., & Gościński, A. (2011). *Cloud computing : principles and paradigms*. Wiley.
- Canonical. (2011). Juju. Retrieved October 23, 2019, from <https://jaas.ai/>
- Carver, J. C. (2010). Towards Reporting Guidelines for Experimental Replications : A Proposal. *1st International Workshop on Replication in Empirical Software Engineering Research, RESEER*, pp. 2–5. <https://doi.org/10.1109/3CA.2010.5533863>
- Casale, G., Artac, M., van den Heuvel, W.-J., van Hoorn, A., Jakovits, P., Leymann, F., Long, M., Papanikolaou, V., Presenza, D., Russo, A., Srirama, S. N., Tamburri, D. A., Wurster, M., & Zhu, L. (2019). RADON: rational decomposition and orchestration for serverless computing. In *SICS Software-Intensive Cyber-Physical Systems*, pp. 1–11. <https://doi.org/10.1007/s00450-019-00413-w>
- Chef. (2009). Chef. Retrieved September 25, 2019, from <https://www.chef.io/>
- Chen, W., Liang, C., Wan, Y., Gao, C., Wu, G., Wei, J., & Huang, T. (2016). MORE: A model-

- driven operation service for cloud-based IT systems. *2016 IEEE International Conference on Services Computing (SCC)*, pp. 633–640. <https://doi.org/10.1109/SCC.2016.88>
- Ciolkowski, M., Shull, F., & Biffi, S. (2002). A family of experiments to investigate the influence of context on the effect of inspection techniques. In *Proceedings of the Sixth International Conference on Empirical Assessment in Software Engineering (EASE)*, Keele, UK, pp. 48–60.
- Cloudify. (2012). Cloudify. Retrieved October 21, 2019, from <https://cloudify.co/>
- Davey, A., & Savla, J. (2009). Statistical power analysis with missing data: A structural equation modeling approach. In *Statistical Power Analysis with Missing Data: A Structural Equation Modeling Approach*. <https://doi.org/10.4324/9780203866955>
- Davis, F. D. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, pp. 319–340. <https://doi.org/10.2307/249008>
- Davis, F. D., Bagozzi, R. P., & Warshaw, P. R. (1989). User Acceptance of Computer Technology: A Comparison of Two Theoretical Models. *Management Science*, *35*(8), pp. 982–1003. <https://doi.org/10.1287/mnsc.35.8.982>
- DeHaan, M. (2012). Ansible. Retrieved January 25, 2019, from Sponsored by Red Hat website: <https://www.ansible.com/>
- Dolstra, E., Vermaas, R., & Levy, S. (2013). Charon: Declarative provisioning and deployment. *2013 1st International Workshop on Release Engineering, RELENG 2013 - Proceedings*, pp. 17–20. <https://doi.org/10.1109/RELENG.2013.6607691>
- Eclipse Foundation. (2006a). Acceleo. Retrieved August 29, 2019, from <https://www.eclipse.org/acceleo/>
- Eclipse Foundation. (2006b). Graphical Modeling Framework. Retrieved September 4, 2019, from <https://www.eclipse.org/gmf-tooling/>
- Eclipse Foundation. (2012). Emfatic. Retrieved September 4, 2019, from <https://www.eclipse.org/emfatic/>
- Ellis, P. D. (2012). The Essential Guide to Effect Sizes. In *The Essential Guide to Effect Sizes*. <https://doi.org/10.1017/cbo9780511761676>
- Erl, T., Puttini, R., & Mahmood, Z. (2013). *Cloud Computing: Concepts, Technology & Architecture* (First Edition).
- Evans, C. (2001). YAML. Retrieved September 3, 2019, from <https://yaml.org/>
- Ferry, N., & Rossini, A. (2018). CloudMF: Model-Driven Management of Multi-Cloud Applications. *ACM Transactions on Internet Technology (TOIT)*, *18*(2), pp. 16–24. <https://doi.org/10.1145/3125621>
- Gartner. (2018). DevOps. Retrieved September 13, 2019, from <https://www.gartner.com/it-glossary/devops>
- Geerling, J. (2015). *Ansible for DevOps: Server and configuration management for humans*. Retrieved from <https://www.amazon.es/Ansible-DevOps-configuration-management-English-ebook/dp/B016G55NOU>
- Genero, M., Cruz-Lemus, J., & Piattini, M. (2014). Métodos de investigación en ingeniería del software. In *Editorial RA-MA*.

- Glaser, F. (2016). Domain Model Optimized Deployment and Execution of Cloud Applications with TOSCA. In *System Analysis and Modeling. Technology-Specific Aspects of Models. SAM 2016. Lecture Notes in Computer Science* (Vol. 9959). <https://doi.org/10.1007/978-3-319-46613-2>
- Gómez, O. S., Juristo, N., & Vegas, S. (2014). Understanding replication of experiments in software engineering: A classification. *Information and Software Technology*. <https://doi.org/10.1016/j.infsof.2014.04.004>
- Google Inc. (2012). Google Compute Engine. Retrieved November 7, 2019, from https://en.wikipedia.org/wiki/Google_Compute_Engine
- Gouw, S. de, Lienhardt, M., Mauro, J., Nobakht, B., & Zavattaro, G. (2015). On the Integration of Automatic Deployment into the ABS Modeling Language. *Service Oriented and Cloud Computing. ESOC 2015. Lecture Notes in Computer Science, Volume 9306*. https://doi.org/10.1007/978-3-319-24072-5_4
- Hanappi, O., Hummer, W., & Dustdar, S. (2016). Asserting reliable convergence for configuration management scripts. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA, 02-04-Nov, pp. 328–343*. <https://doi.org/10.1145/2983990.2984000>
- HashiCorp. (2017). Terraform. Retrieved August 29, 2019, from <https://www.terraform.io/>
- HashiCorp Inc. (2012). Packer. Retrieved October 22, 2019, from <https://www.packer.io/>
- Hashimoto, M., & Bender, J. (2010). Vagrant. Retrieved October 22, 2019, from <https://www.vagrantup.com/>
- Hintsch, J., Gorling, C., & Turowski, K. (2016). Modularization of Software as a Service Products: A Case Study of the Configuration Management Tool Puppet. *Proceedings - 2015 3rd International Conference on Enterprise Systems, ES 2015, 184–191*. <https://doi.org/10.1109/ES.2015.25>
- Hochstein, L., & Moser, R. (2017). *Ansible: Up and Running* (2nd Edition). Retrieved from <http://shop.oreilly.com/product/0636920065500.do>
- Höst, M., Regnell, B., & Wohlin, C. (2000). Using students as subjects - a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering, 5*(3), pp. 201–214. <https://doi.org/10.1023/A:1026586415054>
- Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. In *Continuous Delivery* (First Edition). <https://doi.org/10.1007/s13398-014-0173-7.2>
- Hummer, W., Rosenberg, F., Oliveira, F., & Eilam, T. (2013). *Automated testing of chef automation scripts*. 1–2. <https://doi.org/10.1145/2541614.2541632>
- Hykes, S. (2013). Docker. Retrieved October 22, 2019, from <https://www.docker.com/>
- Ikeshita, K., Ishikawa, F., & Honiden, S. (2017). *Test Suite Reduction in Idempotence Testing of Infrastructure as Code. 10375*, pp. 98–115. <https://doi.org/10.1007/978-3-319-61467-0>
- ISO/IEC. (1991). International Standard ISO/IEC 9126-4. *Information Technology -- Software Product Evaluation -- Quality Characteristics and Guidelines for Their Use*.
- ISO/IEC, & IEEE. (2010). ISO/IEC/IEEE 24765:2010 Systems and software engineering —

- Vocabulary. In *ISO/IEC/IEEE 24765*.
- Jiang, Y., & Adams, B. (2015). Co-evolution of infrastructure and source code: An empirical study. *IEEE International Working Conference on Mining Software Repositories, 2015-Augus*, pp. 45–55. <https://doi.org/10.1109/MSR.2015.12>
- Johann, S. (2017). Kief Morris on Infrastructure as Code. *IEEE Software, 34*(1), pp. 117–120. <https://doi.org/10.1109/MS.2017.13>
- Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming, 72*(1–2), pp. 31–39. <https://doi.org/10.1016/j.scico.2007.08.002>
- Juristo, N., & Gómez, O. S. (2012). *Replication of Software Engineering Experiments*. https://doi.org/10.1007/978-3-642-25231-0_2
- Karvinen, T., & Li, S. (2017). Investigating survivability of configuration management tools in unreliable and hostile networks. *2017 3rd International Conference on Information Management, ICIM 2017*, 327–331. <https://doi.org/10.1109/INFOMAN.2017.7950402>
- Kathiravelu, P., & Veiga, L. (2016). SENDIM for incremental development of cloud networks: Simulation, emulation and deployment integration middleware. *Proceedings - 2016 IEEE International Conference on Cloud Engineering, IC2E 2016*, pp. 143–146. <https://doi.org/10.1109/IC2E.2016.22>
- Kawaguchi, K. (2011). Jenkins. Retrieved September 3, 2019, from <https://jenkins.io/>
- Keckskemeti, G., Kertesz, A., & Marosi, A. C. (2016). Towards a Methodology to Form Microservices from Monolithic Ones. *Proceedings of the Euro-Par 2016: Parallel Processing Workshops*. <https://doi.org/10.1007/978-3-319-58943-5>
- Kitchenham, B. A., Pfleger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K., & Rosenberg, J. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering, 28*(8), pp. 721–734. <https://doi.org/10.1109/TSE.2002.1027796>
- Kolovos, D. S., García-Domínguez, A., Rose, L. M., & Paige, R. F. (2015). Eugenia: towards disciplined and automated development of GMF-based graphical model editors. *Software & Systems Modeling, 16*(1), 229–255. <https://doi.org/10.1007/s10270-015-0455-3>
- Kotonya, G., & Sommerville, I. (1998). *Requirements Engineering: Processes and Techniques*. John Wiley & Sons Inc.
- Kuehl, R. O. (2000). *Design of experiments: statistical principles of research design and analysis* (2nd Edition). Pacific Grove CA: Duxbury/Thomson Learning.
- Lwakatara, L. E., Kuvaja, P., & Oivo, M. (2015). Dimensions of devOps. *Lecture Notes in Business Information Processing, 212*, 212–217. https://doi.org/10.1007/978-3-319-18612-2_19
- Mann, A., Brown, A., Stahnke, M., & Kersten, N. (2018). 2018 State of DevOps Report. In *Puppetlabs*. [https://doi.org/10.1016/S0022-3913\(12\)00047-9](https://doi.org/10.1016/S0022-3913(12)00047-9)
- Marrone, S., & Nardone, R. (2015). Automatic resource allocation for high availability cloud services. *Procedia Computer Science, 52*(1), pp. 980–987. <https://doi.org/10.1016/j.procs.2015.05.176>

- Maxwell, K. (2002). Applied Statistics for Software Managers. In *Web Engineering*. Prentice Hall.
- Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. In *Recommendations of the National Institute of Standards and Technology*. <https://doi.org/10.1016/b978-0-12-804018-8.15003-x>
- Microsoft. (2010). Microsoft Azure. Retrieved November 7, 2019, from https://en.wikipedia.org/wiki/Microsoft_Azure
- Miglierina, M. (2015). Application deployment and management in the cloud. *Proceedings - 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014*, (Section IV), pp. 422–428. <https://doi.org/10.1109/SYNASC.2014.63>
- Miglierina, M., & Tamburri, D. A. (2017). Towards Omnia: A monitoring factory for quality-Aware DevOps. *ICPE 2017 - Companion of the 2017 ACM/SPEC International Conference on Performance Engineering*, pp. 145–150. <https://doi.org/10.1145/3053600.3053629>
- Moody, D. L. (2003). The method evaluation model: a theoretical model for validating information systems design methods. *Proc. of ECIS '03*, pp. 1327–1336. <https://doi.org/10.1.1.108.3682>
- Morris, K. (2016). *Infrastructure As Code: Managing Servers in the Cloud* (First Edition). O'Reilly Media.
- Nitto, E. Di, Matthews, P., Petcu, D., & Solberg, A. (2017). Model-Driven Development and Operation of Multi-Cloud Applications. In E. Di Nitto, P. Matthews, D. Petcu, & A. Solberg (Editors.), *Springer International Publishing*. <https://doi.org/10.1007/978-3-319-46031-4>
- OASIS. (2013). Topology and Orchestration Specification for Cloud Applications (TOSCA). Retrieved October 23, 2019, from https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
- OpenStack Foundation. (2010). OpenStack. Retrieved October 23, 2019, from <https://www.openstack.org/>
- OpenStack Foundation. (2013). OpenStack Heat. Retrieved August 29, 2019, from <https://wiki.openstack.org/wiki/Heat>
- Parnin, C., Helms, E., Atlee, C., Boughton, H., Ghattas, M., Glover, A., Holman, J., Micco, J., Murphy, B., Savor, T., Stumm, M., Whitaker, S., & Williams, L. (2017). The Top 10 Adages in Continuous Deployment. *IEEE Software*, 34(3), pp. 86–95. <https://doi.org/10.1109/MS.2017.86>
- Preston-Werner, T., Wanstrath, C., & Hyett, P. (2008). GitHub. Retrieved September 3, 2019, from <https://github.com/>
- Puppet Labs. (2005). Puppet. Retrieved September 25, 2019, from <https://puppet.com/>
- Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, pp. 65–77. <https://doi.org/10.1016/j.infsof.2018.12.004>
- Riungu-kalliosaari, L., Makinen, S., Lwakatara, L. E., Tiihonen, J., & Mannisto, T. (2016). DevOps Adoption Benefits and Challenges in Practice: A Case Study. *17th International Conference on Product-Focused Software Process Improvement, PROFES 2016, 3009*, pp. 590–597.

- Rossini, A. (2015). Cloud Application Modelling and Execution Language (CAMEL) and the PaaS Workflow. *2015 European Conference on Service-Oriented and Cloud Computing (E/SOCC)*, 567, pp. 437–439. <https://doi.org/10.1007/978-3-319-33313-7>
- Sandobalín, J. (2017). A Model-Driven Approach to Continuous Delivery of Cloud Resources. *Proceedings - 15th International Conference on Service-Oriented Computing, ICSOC, 10797 LNCS*, pp. 346–351. https://doi.org/10.1007/978-3-319-91764-1_29
- Sandobalín, J., Insfrán, E., & Abrahão, S. (2017a). An Infrastructure Modelling Tool for Cloud Provisioning. *Proceedings - IEEE 14th International Conference on Services Computing, SCC*, pp. 354–361. <https://doi.org/10.1109/SCC.2017.52>
- Sandobalín, J., Insfrán, E., & Abrahão, S. (2017b). End-to-End Automation in Cloud Infrastructure Provisioning. *Proceedings - 26th International Conference on Information Systems Development, ISD*. Cyprus.
- Sandobalín, J., Insfrán, E., & Abrahão, S. (2018). An Infrastructure Modeling Approach for Multi-Cloud Provisioning. *Proceedings - 27th International Conference on Information Systems Development, ISD 2018*. Lund, Sweden.
- Santos, A., Gomez, O. S., & Juristo, N. (2018). Analyzing Families of Experiments in SE: a Systematic Mapping Study. *IEEE Transactions on Software Engineering*, (July), pp. 1–18. <https://doi.org/10.1109/TSE.2018.2864633>
- Scheuner, J., Cito, J., Leitner, P., & Gall, H. (2015). Cloud WorkBench: Benchmarking IaaS Providers Based on Infrastructure-as-Code. *2015 International World Wide Web Conference Committee (IW3C2)*, pp. 239–242. <https://doi.org/10.1145/2740908.2742833>
- Scheuner, J., Leitner, P., Cito, J., & Gall, H. (2014). Cloud work bench - Infrastructure-as-code based cloud benchmarking. *2014 Cloud Computing Technology and Science (CloudCom)*, pp. 246–253. <https://doi.org/10.1109/CloudCom.2014.98>
- Schmidt, D. C. (2006). Model-Driven Engineering. *Computer*, (February), pp. 25–31.
- Senapathi, M., Buchan, J., & Osman, H. (2018). DevOps Capabilities, Practices, and Challenges: Insights from a Case Study. *22nd International Conference on Evaluation and Assessment in Software Engineering*, pp. 57–67. <https://doi.org/10.1145/3210459.3210465>
- Sharma, T., Fragkoulis, M., & Spinellis, D. (2016). Does your configuration code smell? *Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016*, pp. 189–200. <https://doi.org/10.1145/2901739.2901761>
- Singh, N. K., Thakur, S., Chaurasiya, H., & Nagdev, H. (2015). Automated provisioning of application in IAAS cloud using Ansible configuration management. *Proceedings on 2015 1st International Conference on Next Generation Computing Technologies, NGCT 2015*, pp. 81–85. <https://doi.org/10.1109/NGCT.2015.7375087>
- Sonatype. (2002). Maven. Retrieved September 2, 2019, from <https://maven.apache.org/>
- Sonatype. (2008). Nexus. Retrieved September 3, 2019, from <https://www.sonatype.com/nexus-repository-sonatype>
- Spinellis, D. (2012). Don't install software by hand. *IEEE Software*, 29(4), pp. 86–87. <https://doi.org/10.1109/MS.2012.85>
- Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008). EMF: eclipse modeling

- framework. In *Engineering*. Retrieved from <http://portal.acm.org/citation.cfm?id=1197540>
- Templeton, G. F. (2011). A two-step approach for transforming continuous variables to normal: Implications and recommendations for IS research. *Communications of the Association for Information Systems*. <https://doi.org/10.17705/1CAIS.02804>
- Vegas, S., Apa, C., & Juristo, N. (2016). Crossover Designs in Software Engineering Experiments: Benefits and Perils. *IEEE Transactions on Software Engineering*, 42(2), pp. 120–135. <https://doi.org/10.1109/TSE.2015.2467378>
- Virmani, M. (2015). Understanding DevOps & bridging the gap from continuous integration to continuous delivery. *5th International Conference on Innovative Computing Technology, INTECH 2015*, (Intech), 78–82. <https://doi.org/10.1109/INTECH.2015.7173368>
- Volanja. (2013). Ansible_spec. Retrieved September 4, 2019, from https://github.com/volanja/ansible_spec
- Weerasiri, D., Barukh, M., Benatallah, B., & Cao, J. (2016). A Model-Driven Framework for Interoperable Cloud Resources Management. In *Service-Oriented Computing, ICSOC 2016. Lecture Notes in Computer Science* (Vol. 9936). <https://doi.org/10.1007/978-3-319-46295-0>
- Weiss, A., Guha, A., & Brun, Y. (2017). Tortoise: Interactive system configuration repair. *32nd IEEE/ACM International Conference on Automated Software Engineering*, pp. 625–636. <https://doi.org/10.1109/ASE.2017.8115673>
- Wettinger, J., Andrikopoulos, V., & Leymann, F. (2015). Automated capturing and systematic usage of DevOps knowledge for cloud applications. *Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015*, 60–65. <https://doi.org/10.1109/IC2E.2015.23>
- Wettinger, J., Behrendt, M., Binz, T., Breitenbücher, U., Breiter, G., Leymann, F., Moser, S., Schwertle, I., & Spatzier, T. (2013). Integrating configuration management with model-driven cloud management based on TOSCA. *2013 International Conference on Cloud Computing and Services Science (CLOSER)*, pp. 437–446.
- Wettinger, J., Breitenbücher, U., Kopp, O., & Leymann, F. (2015). Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel. *Future Generation Computer Systems, Volume 56*, pp. 317–332. <https://doi.org/http://dx.doi.org/10.1016/j.future.2015.07.017>
- Wettinger, J., Breitenbucher, U., & Leymann, F. (2014). Standards-based DevOps automation and integration using TOSCA. *Proceedings - IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC*, 59–68. <https://doi.org/10.1109/UCC.2014.14>
- Wettinger, J., Breitenbücher, U., & Leymann, F. (2014). DevOpSlang - Bridging the gap between development and operations. *The European Conference on Service-Oriented and Cloud Computing (ESOCC)*, 8745, 108–122. https://doi.org/10.1007/978-3-662-44879-3_8
- Wieringa, R. (2014). Design Science Methodology for Information Systems and Software Engineering. In *Springer Berlin Heidelberg*. <https://doi.org/10.1145/1810295.1810446>
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. In *Experimentation in Software Engineering*. <https://doi.org/10.1007/978-3-642-29044-2>

Resumen

DevOps (*Development & Operations*) es un movimiento que fomenta la colaboración entre los desarrolladores y el personal de operaciones a través de un conjunto de principios, prácticas y herramientas para optimizar el tiempo de entrega del software. En particular, la práctica del *despliegue continuo* de software es una gran fuente de problemas y genera mucha atención cuando los artefactos de software se entregan tarde o cuando un defecto crítico llega a producción. Al mismo tiempo, la práctica del despliegue continuo es la frontera entre los desarrolladores y el personal de operaciones en el ciclo de entrega del software. En consecuencia, se recomienda iniciar la implantación de DevOps con la práctica del despliegue de software. Para enfrentar este desafío, los profesionales e investigadores están utilizando la Infraestructura como Código (*Infrastructure as Code*, IaC), que es un enfoque para la automatización de la infraestructura basado en prácticas de desarrollo de software. El objetivo de IaC es definir en un script todas las instrucciones para crear, actualizar y ejecutar recursos de infraestructura. En este escenario, la automatización del aprovisionamiento de la infraestructura acelera la práctica del despliegue continuo en el ciclo de entrega del software.

La computación en la nube se ha convertido en el principal modelo de pago por uso utilizado por profesionales e investigadores para conseguir servicios en la nube en un corto período de tiempo. En este escenario, IaC y la computación en la nube están promoviendo algunos cambios en la industria, por ejemplo, los equipos de operaciones pasan todo su tiempo trabajando en software, en lugar de configurar servidores y conectar cables de red. Existe una variedad de herramientas IaC que utilizan scripts para definir y ejecutar los recursos de infraestructura en diferentes proveedores de servicios IaaS (*Infrastructure as a Service*). Sin embargo, la diversidad de lenguajes de scripting de las herramientas IaC junto con la heterogeneidad del tipo de infraestructura que ofrece cada proveedor de servicios IaaS, han ocasionado que utilizar scripts IaC para el aprovisionamiento de infraestructura sea una actividad lenta y propensa a errores.

El objetivo del proyecto de doctorado es proponer una solución a la diversidad de los lenguajes de scripting —de las herramientas IaC— y a la heterogeneidad del tipo de infraestructura que ofrece cada proveedor de servicios IaaS respecto al aprovisionamiento de infraestructura en la nube. Para afrontar estos desafíos se propone MoCIP (*A Model-Driven Approach to Cloud Infrastructure Provisioning*). MoCIP es un enfoque dirigido por modelos para el aprovisionamiento de la infraestructura en la nube que soporta IaC mediante la Ingeniería de Software Dirigida por Modelos (*Model-Driven Engineering*, MDE). MoCIP utiliza los dos principios fundamentales de MDE: *abstracción* y *automatización*. En primer lugar, se desarrolló el lenguaje específico de dominio ArgonML para *abstraer* las capacidades de la computación en la nube, tales como cómputo, elasticidad, almacenamiento y redes. El lenguaje ArgonML permite modelar los recursos de infraestructura de la nube. En segundo lugar, se desarrolló la herramienta ARGON para *automatizar* el aprovisionamiento de infraestructura en la nube. Además, con el fin de investigar la interacción de MoCIP en un contexto de aplicación se diseñaron y ejecutaron un conjunto de experimentos controlados.

Tesis Doctoral

© Julio César Sandobalín Guamán
Valencia, España
MMXV-MMXX



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA