

UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. Ingeniería de Sistemas Informáticos



TRABAJO FIN DE GRADO

Grado en Ingeniería del Software

**Servicios en la Nube con Microsoft Azure:
Desarrollo y Operación de una aplicación
Android con *DevOps***

Curso 2015-2016

Autor:

Alejandro Mora Rodríguez

UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. Ingeniería de Sistemas Informáticos



TRABAJO FIN DE GRADO

Grado en Ingeniería del Software

**Servicios en la Nube con Microsoft Azure:
Desarrollo y Operación de una aplicación
Android con *DevOps***

Curso 2015-2016

Autor:

Alejandro Mora Rodríguez

Directores:

Jessica Díaz Fernández

Jennifer Pérez Benedí

Agradecimientos

Estudiar y trabajar a la vez no es fácil y si tienes una familia menos todavía. El tiempo es el que es y al final se lo he acabado robando a ellos. Sé que mi familia, en el fondo, entiende que lo hago por ellos. Es por tanto de ley agradecer primero y con todo mi corazón a mi mujer, Verónica, por su apoyo y comprensión en estos años que, debido a que me he volcado en mí mismo y mis proyectos personales, no le he prestado la suficiente atención. Espero también poder dar la vuelta a la situación lo antes posible para poder disfrutar de mi hijo Marco en esos **años que ya no vuelven antes que sea demasiado tarde...**

Mención especial a mis dos tutoras, Jessica y Jennifer. Mil gracias por haberme dado la oportunidad de hacer este proyecto que por sus características tecnológicas y metodológicas, todas ellas últimas tendencias, estoy seguro en breve me dará la oportunidad de dar un salto profesional. Como espero seguiremos en contacto os iré contando. Y sobre todo mil gracias por todo el apoyo, consejos, revisiones, enseñanzas y, en definitiva, tiempo que me habéis dedicado y que, lo que más siento, sé que también se lo estáis robando a vuestras familias.

Gracias a todos,

Resumen

Una de las últimas tendencias en el área de las TIC (Tecnologías de la Información y Comunicación) es la adopción del paradigma de Computación en la Nube (*Cloud Computing*). *Cloud Computing* aboga por el uso de recursos computacionales (redes, servidores, almacenamiento, así como aplicaciones y servicios software) de terceros prestados como servicios, evitando el despliegue y mantenimiento de una infraestructura propia. Es por ello, que las empresas están adoptando la nube como la herramienta perfecta para seguir el ritmo de innovación empresarial.

Este Proyecto Fin de Grado (PFG) realiza un estudio exhaustivo del paradigma *Cloud Computing* con el objetivo de poner en práctica “**servicios avanzados para la construcción de aplicaciones en la nube**” utilizando una de las plataformas de desarrollo en la nube existentes en el mercado, concretamente Microsoft Azure. Dichos servicios avanzados han sido configurados para desarrollar y desplegar en la nube una aplicación piloto (caso de estudio) que cubre el desarrollo y despliegue de aplicaciones software en la nube y la gestión del ciclo de vida de estas aplicaciones mediante el enfoque DevOps. Este nuevo movimiento técnico y cultural complementa de alguna forma las metodologías ágiles incluyendo no sólo el proceso de desarrollo sino también el de operación, promoviendo buenas prácticas como la automatización, el control de versiones, y la integración y despliegue continuos. Para ello, este PFG utiliza también los servicios en la nube que Visual Studio Team Services (VSTS) proporciona para la gestión del ciclo de vida de la aplicación.

En aras a cumplir estos objetivos este PFG se ha llevado a cabo un caso de estudio que consiste en el desarrollo y despliegue de una aplicación Android de visualización de fotografías con *backend* en la nube para el almacenamiento y procesado de dichas fotografías. La aplicación está compuesta por tanto de un cliente Android (*frontend*) y un servicio API REST Java (*backend*), que fueron desarrollados, gestionados y desplegados en menos de 4 meses utilizando Microsoft Azure y VSTS.

Abstract

One of the latest trends in the area of ICT (Information and Communication Technology) is the adoption of the Cloud Computing paradigm. Cloud Computing advocates the use of computer resources (networks, servers, storage, as well as applications and software services) provided as third-party services, avoiding the deployment and maintenance of its own infrastructure. This is the reason why companies are adopting the cloud as the perfect tool to keep pace of business innovation.

This Degree Final Project (DFP) performs a comprehensive study of Cloud Computing paradigm **in order to implement “advanced services for building cloud applications”** using one of the cloud development platforms existing in the market, namely Microsoft Azure. These advanced services have been used and set up to develop and deploy a cloud application pilot (case study) that covers the development and deployment of software applications in the cloud and their lifecycle management of these applications through the DevOps approach. This new technical and cultural movement somehow complements agile methodologies including not only the development process but also the operation, promoting good practices such as automation, versioning, and integration and continuous deployment. For this, this DFP also uses cloud services that Visual Studio Team Services (VSTS) provides for managing the lifecycle of the application.

In order to meet these objectives, this DFP developed a case study that consists in developing and deploying an Android application for showing photos with backend in the cloud for storage and processing of such photographs. The application consists of both an Android client (frontend) and a Java REST API service (backend), which were developed, managed and deployed in less than 4 months using Microsoft Azure and VSTS.

Tabla de contenido

1	Introducción	1
1.1	Motivación	3
1.2	Objetivos	4
1.3	Estructura del documento	5
2	Conceptos Previos	6
2.1	Cloud Computing	6
2.1.1	Definición	8
2.1.2	Modelos de servicio	10
2.1.3	Modelos de despliegue	12
2.1.4	Características esenciales	14
2.2	Patrones y Tácticas de Diseño	16
2.2.1	Estilo arquitectónico REST	16
2.2.2	Patrón particionado	19
2.2.3	Patrón vista materializada	20
2.2.4	Táctica manejo de excepciones	22
2.3	Microsoft Azure	22
2.3.1	Azure PowerShell	23
2.3.2	Administrador de Recursos (<i>Resource Manager</i>)	27
2.3.3	Servicio de Almacenamiento	28
2.3.4	Plan del Servicio de Aplicaciones (<i>Service Plan App</i>)	34
2.3.5	Servicio de Aplicaciones: API app	38
2.3.6	Servicio de automatización	45
2.4	Android	52
2.4.1	Android Studio 2.0	54
2.4.2	Estructura de un proyecto en Android Studio 2.0	55
2.5	Desarrollo Ágil de Software	59
2.6	DevOps	64
2.6.1	Automatización Total	66
2.6.2	Control de versiones	66

2.6.3	Integración y Despliegue Continuo	67
3	Caso de Estudio	68
3.1	Descripción	68
3.2	Stakeholders	69
3.3	Diseño Conceptual.....	70
3.4	Arquitectura.....	72
3.4.1	Requisitos del sistema	73
3.4.2	Decisiones de diseño.....	79
3.4.3	Vistas Arquitectónicas	87
4	Desarrollo de una aplicación Android utilizando servicios de Microsoft Azure	95
4.1	Servicio API REST Java (<i>backend</i>).....	95
4.1.1	Construcción: Eclipse y Maven (<i>pom.xml</i>)	95
4.1.2	Interfaz API: JAX-RS con Jersey	97
4.1.3	Documentación API: Swagger	102
4.2	Cliente REST Android (<i>frontend</i>)	105
4.2.1	Construcción: Android Studio y Gradle (<i>build.gradle</i>).....	105
4.2.2	Interactuación con API REST: Retrofit	107
4.2.3	Interfaz gráfica de usuario.....	110
5	Administración del ciclo de vida de la aplicación con Visual Studio Team Services....	114
5.1	Configuración inicial.....	114
5.1.1	Alta de cuenta de Microsoft y activación de suscripción a Microsoft Azure 114	
5.1.2	Alta de la cuenta VSTS y creación del proyecto.....	115
5.1.3	Concesión de permisos de acceso al repositorio de versiones en VSTS.....	118
5.1.4	Concesión de permisos de acceso al proyecto en VSTS	121
5.1.5	Instalación de un agente de compilación privado de VSTS en local.....	122
5.1.6	Alta del servicio de punto de conexión a Microsoft Azure desde VSTS.....	124
5.1.7	Creación de grupos recursos en Microsoft Azure	127
5.2	Proceso diario de desarrollo.....	130
5.2.1	Construcción manual o automática de aplicaciones desde VSTS	130
5.2.2	Creación recursos de grupo de recursos en Microsoft Azure desde VSTS....	133
5.2.3	Construcción y copia de un API en Microsoft Azure desde VSTS	136

5.2.4	Construcción aplicación Android desde VSTS	139
5.2.5	Ejecución de test de carga Apache JMeter en VSTS	142
6	Conclusiones y Trabajos Futuros	147
6.1	Conclusiones.....	147
6.2	Trabajos Futuros.....	148
	Bibliografía.....	151
	Glosario de términos	155

Lista de figuras

Fig. 1 Cuadrante mágico de Gartner para PaaS para el año 2015 [3]	2
Fig. 2 Evolución de la arquitectura de las aplicaciones [5]	6
Fig. 3 Responsabilidades de usuario y proveedor servicios en modelos servicio <i>Cloud</i> [8] ..	10
Fig. 4 Definición de nube según NIST [9].....	14
Fig. 5 Interacción entre cliente y servidor mediante API <i>Web</i> [11]	17
Fig. 6 Restricciones del estilo arquitectónico REST	17
Fig. 7 Particionar almacenamientos para mejorar escalabilidad [13]	20
Fig. 8 Patrón vista materializada [13]	21
Fig. 9 Aspectos fundamentales de ALM [16]	22
Fig. 10 Servicios IaaS, PaaS de Azure	23
Fig. 11 Instalación Azure Resource Manager para PowerShell	24
Fig. 12 Conexión interactiva a Microsoft Azure Resource Manager con Azure PowerShell...24	24
Fig. 13 Gestión Active Directory desde el portal clásico de Azure.....	25
Fig. 14 Usuarios en el Active Directory del portal clásico de Azure	25
Fig. 15 Administradores de la suscripción desde el portal clásico de Microsoft Azure	26
Fig. 16 Obtención credenciales de cuenta de afiliación con Azure PowerShel.....	26
Fig. 17 Conexión no interactiva a Azure Resource Manager con Azure PowerShell	27
Fig. 18 Administración de la cuenta de almacenamiento en Microsoft Azure	29
Fig. 19 Contenedor de una cuenta de almacenamiento [19]	29
Fig. 20 Componentes de un servicio tabla [19]	30
Fig. 21 Componentes de un servicio cola [19]	31
Fig. 22 Precio almacenamiento en blobs en Europa Occidental [20]	33
Fig. 23 Precio acceso a los blobs en Europa Occidental [20]	33

Fig. 24 Precio acceso y almacenamiento en tablas en Europa Occidental [20]	34
Fig. 25 Crear plan de servicio de aplicaciones desde la plataforma	35
Fig. 26 Administración de plan de servicio de aplicaciones en Azure	35
Fig. 27 Planes de tarifa del servicio de aplicaciones Microsoft Azure	36
Fig. 28 Escalar verticalmente el plan de servicio de aplicaciones	37
Fig. 29 Escalar horizontalmente el plan del servicio de aplicaciones.....	38
Fig. 30 Crear aplicación API desde la plataforma	38
Fig. 31 Administración de aplicación de API en Azure.....	39
Fig. 32 Configuración de la aplicación API.....	40
Fig. 33 Configuración API: credenciales de implementación.....	41
Fig. 34 Configuración API: espacio de implementación (<i>staying slot</i>).....	41
Fig. 35 Configuración API: creación de prototipo API en Java	42
Fig. 36 Herramientas API: explorador de recursos Azure en JSON.....	43
Fig. 37 Herramientas API: explorador de recursos de Azure en PowerShell	43
Fig. 38 Herramientas API: interfaz Kudu de depuración de aplicaciones web CMD	44
Fig. 39 Herramientas API: interfaz Kudu de depuración de aplicaciones <i>Web</i> PowerShell ..	44
Fig. 40 Intercambio de aplicación API entre producción y slot.....	45
Fig. 41 Crear cuenta de automatización desde plataforma.....	46
Fig. 42 Administración de cuenta de automatización en Azure	47
Fig. 43 Plan de tarifa de la cuenta de automatización Azure	47
Fig. 44 <i>Runbook</i> de tipo <i>Script</i> PowerShell	48
Fig. 45 Programaciones de un <i>runbook</i>	49
Fig. 46 Activos de una cuenta de automatización.....	49
Fig. 47 Certificados de cuenta de automatización	50
Fig. 48 Credenciales de cuenta de automatización	51

Fig. 49 Variables de cuenta de automatización	51
Fig. 50 Programaciones de una cuenta de automatización	52
Fig. 51 Evolución cuota de mercado SO móviles entre 2009 y 2015 en el mundo [25]	53
Fig. 52 Mapa distribución mundial iOS y Android [25]	53
Fig. 53 Porcentaje de uso distintas versiones Android en 2016 [26]	54
Fig. 54 UI Android 2.0.....	54
Fig. 55 Proyecto Android " <i>FrontEnd</i> " y módulo " app ".....	55
Fig. 56 Estructura de una aplicación Android	58
Fig. 57 Manifiesto por el Desarrollo Ágil de Software [29].....	59
Fig. 58 Principios del Manifiesto Ágil [30]	60
Fig. 59 Proceso de desarrollo ágil de software. Metodología SCRUM [32].....	63
Fig. 60 <i>Board</i> de Scrum de VSTS.....	63
Fig. 61 Desarrollo y Operaciones por separado [33]	64
Fig. 62 Desarrollo y Operaciones integrados [33].....	64
Fig. 63 Eliminación grietas en el ciclo de vida de desarrollo [35]	65
Fig. 64 DevOps <i>workflow</i> [36].....	65
Fig. 65 Diseño conceptual del caso de estudio.....	71
Fig. 66 Desarrollo ágil de sw a través de VSTS: definición HU	74
Fig. 67 <i>Backlog Agile</i> desde donde se gestionan las HU en la pestaña WORK de VSTS.....	74
Fig. 68 <i>Board Agile</i> en VSTS en la pestaña WORK de VSTS.....	75
Fig. 69 <i>Iterations/Sprints Agile</i> en el panel de control de VSTS	75
Fig. 70 Interacción típica de la aplicación entre cliente REST y servicio RESTful.....	80
Fig. 71 Almacenamiento sin particionar.....	82
Fig. 72 Partición vertical	82
Fig. 73 Partición horizontal	83

Fig. 74 Vista materializada.....	84
Fig. 75 Trazabilidad entre decisiones de diseño y requisitos.....	86
Fig. 76 Vista de los servicios Azure.....	90
Fig. 77 Vista de flujo de trabajo	94
Fig. 78 pom.xml	97
Fig. 79 Extracto del API con las anotaciones JAX-RS	99
Fig. 80 Extracto del modelo con las anotaciones JAX-RS.....	100
Fig. 81 Peticiones HTTP REST implementadas en el caso de estudio	101
Fig. 82 Extracto de código del API con anotaciones Swagger	102
Fig. 83 Documentación de API generada con Swagger UI	103
Fig. 84 Prueba de operación de API con Swagger UI.....	104
Fig. 85 <i>build.gradle</i>	107
Fig. 86 Definición interfaz Retrofit	108
Fig. 87 Creación servicio Retrofit	109
Fig. 88 Llamada síncrona en Retrofit	109
Fig. 89 Llamada asíncrona en Retrofit.....	110
Fig. 90 Creación aplicación Android partiendo de la plantilla Master/Detail Flow	111
Fig. 91 Visualización aplicación Android en modo multipanel.....	112
Fig. 92 Visualización de la aplicación Android en modo único panel	113
Fig. 93 Visualización mensaje error si el cliente REST no puede acceder al API RES.....	113
Fig. 94 Configuración entornos desarrollo, test y producción de la aplicación desde VSTS	114
Fig. 95 Sitio web de activación de suscripciones de <i>Azure</i>	115
Fig. 96 Sitio <i>Web</i> registro VSTS.....	116
Fig. 97 Sitio <i>Web</i> creación cuenta VSTS.....	116
Fig. 98 Sitio <i>Web</i> opciones nueva cuenta VSTS.....	117

Fig. 99 Panel principal de la cuenta VSTS: creación de nuevo proyecto VSTS	117
Fig. 100 Botón de acceso al panel de configuración de VSTS	118
Fig. 101 Botón para añadir usuarios al sistema de control de versiones en VSTS	118
Fig. 102 Ventana para añadir usuario al sistema de control de versiones en VSTS	119
Fig. 103 Ajuste de los permisos de acceso al sistema de control de versiones en VSTS ...	119
Fig. 104 Correo confirmación que llega a usuario dado de alta al repositorio en VSTS.....	120
Fig. 105 Ventana de alta de miembros del equipo en un proyecto en VSTS.....	121
Fig. 106 Ventana de alta de usuario como administrador del equipo en VSTS	122
Fig. 107 Botón de acceso al panel de configuración de VSTS	122
Fig. 108 Botón de descarga agente de compilación en pestaña Agent Pools VSTS	123
Fig. 109 Contenido del fichero "agent.zip".....	123
Fig. 110 Instalación agente de compilación.....	123
Fig. 111 Agentes de compilación de VSTS.....	124
Fig. 112 Ejecución de agente de compilación privado en consola de comandos Windows	124
Fig. 113 Botón de acceso al panel de configuración de VSTS	124
Fig. 114 Ventana de servicios de puntos de conexión de VSTS.....	125
Fig. 115 Alta de punto de conexión a subscripción Azure en VSTS.....	125
Fig. 116 Resultado ejecución <i>script "SNPCreacion.ps1"</i> en consola PowerShell	126
Fig. 117 Servicios puntos de conexión en la pestaña Services del Control Panel de VSTS	126
Fig. 118 Definición de la compilación que instala los entornos de trabajo en Azure	127
Fig. 119 Extracto <i>script PowerShell "Deploy-Azure.ps1"</i>	128
Fig. 120 Grupos de recursos para los entornos producción, desarrollo y test en Azure	129
Fig. 121 Proceso diario de desarrollo	130
Fig. 122 Activación CI con cambios en la rama master del repositorio en VSTS	131
Fig. 123 Selección del agente de compilación por defecto en VSTS	131

Fig. 124 Encolar construcción aplicación con agente hospedado	132
Fig. 125 Consola VSTS con el resultado de la construcción de la aplicación	132
Fig. 126 Definición de la compilación que crea un grupo de recursos Azure desde VSTS .	133
Fig. 127 Extracto <i>script JSON</i> “ <i>apimedtrans.json</i> ”.....	134
Fig. 128 Variables de la compilación despliegue recursos de grupo recursos Azure	135
Fig. 129 Aspecto y grupo recursos creado desde VSTS en plataforma Microsoft Azure	136
Fig. 130 Definición compilación despliegue API: 1er. paso compilación con Maven	137
Fig. 131 Definición copia API con <i>cURL</i>	138
Fig. 132Variables para la copia API	138
Fig. 133 API desplegado en Azure desde VSTS.....	139
Fig. 134 Definición compilación construcción Android: 1er. paso compilación Gradle.....	140
Fig. 135 Definición compilación construcción Android: 2º paso firma	141
Fig. 136 Android construido con agente privado e instalado en AVD desde VSTS.....	142
Fig. 137 Configuración test de carga con JMeter: Thread Group	143
Fig. 138 Configuración test de carga con JMeter: HTTP Resquest	143
Fig. 139 Alta del test de carga Apache JMeter desde la pestaña TEST de VSTS	144
Fig. 140 Ejecución del test de caga Apache JMeter desde la pestaña TEST de VSTS	144
Fig. 141 Resultados del test de carga desde la pestaña TEST de VSTS	145
Fig. 142 Representación gráfica resultados test carga en pestaña TEST de VSTS	146
Fig. 143 Patrón Colas [13].....	148
Fig. 144 Patrón Caché [13].....	149
Fig. 145 Patrón Reintentar [13].....	150
Fig. 146 Aspectos fundamentales de ALM [16]	155
Fig. 147 Descripción Cuadrante Mágico de Gartner [48]	158

1 Introducción

Una de las últimas tendencias en el área de las TIC (Tecnologías de la Información y Comunicación) es la adopción del paradigma de Computación en la Nube (*Cloud Computing*) que aboga por el uso de recursos computacionales (redes, servidores, almacenamiento, así como aplicaciones y servicios software) de terceros prestados como servicios, evitando el despliegue y mantenimiento de una infraestructura y cuyos ventajas más destacables son [1]:

- **Coste:** aunque la migración de la información a la nube supone una inversión importante, ya que hay que pagar al proveedor *Cloud* que proporciona sus servidores e infraestructuras, se evita el gasto en *hardware* y *software* propio, renovación de licencias y personal de soporte técnico. En cualquier caso, solo se paga por los servicios que se utilizan y durante el tiempo que se utilizan, es decir, lo que se conoce como modelo *pay-as-you-go* (pago por uso).
- **Ubicuidad:** se puede acceder a la información de forma inmediata desde cualquier lugar del mundo con un dispositivo que posea conexión a Internet.
- **Disponibilidad:** el proveedor *Cloud* garantiza que el sistema está operativo la mayor parte del tiempo.
- **Rendimiento y Escalabilidad:** para evitar que baje el rendimiento ante un eventual aumento de carga, se pueden aumentar los recursos disponibles durante el tiempo necesario.

Sin embargo, también hay que considerar algunas desventajas como, por ejemplo, problemas relacionados con la seguridad y el cumplimiento de las normativas de protección y privacidad de datos [2]:

- **Seguridad:** dada la sensibilidad de los datos que podrían estar en la nube, hay que tratar de minimizar el riesgo del mal uso de esta información. Por ello, se hace necesario, por ejemplo, tener un cifrado robusto de la información.
- **Cumplimiento de la normativa de datos:** el modelo de *Cloud Computing* hace posible que tanto los proveedores de servicios como los datos almacenados en la nube se encuentren ubicados en cualquier punto del planeta. El cliente que contrata servicios de *Cloud Computing* es el responsable del tratamiento de los datos por lo que la normativa aplicable al cliente y al prestador del servicio es la legislación española sobre protección de datos (Ley Orgánica 15/1999, de 13 de diciembre y Reglamento de desarrollo –RLOPD– aprobado por R.D. 1720/2007). La localización de los datos tiene importancia porque las garantías exigibles para su protección son distintas según los

países en que se encuentren. En este sentido, los países del Espacio Económico Europeo ofrecen garantías suficientes y no se considera legalmente que exista una transferencia internacional de datos.

En cualquier caso, y teniendo en cuenta que el término *Cloud Computing* se utilizó por primera vez hace ya casi veinte años, es durante la última década cuando esta tendencia de servicios en la nube ha ido creciendo a nivel mundial. Entre los proveedores que sirven este tipo de servicios en la nube es posible destacar Microsoft, Google, IBM o Amazon. En concreto, Microsoft Azure ofrece un conjunto de servicios avanzados para el desarrollo y despliegue de aplicaciones en la nube abstrayendo detalles de la infraestructura de despliegue que se conocen de forma genérica como *Platform as a Service* (PaaS). Además, según la consultora Gartner, Microsoft Azure por tercer año consecutivo es líder en su Cuadrante Mágico para empresas en PaaS (ver Fig. 1)

Figure 1. Magic Quadrant for Enterprise Application Platform as a Service, Worldwide



Fig. 1 Cuadrante mágico de Gartner para PaaS para el año 2015 [3]

1.1 Motivación

Las empresas de reconocido éxito están buscando constantemente nuevas oportunidades de negocio, pugnando por ser competitivas y no quedar fuera del mercado. Hoy en día, las empresas tienen que ser ágiles para poder responder a las nuevas necesidades del mercado, con la mayor rapidez y la menor inversión posible, es decir, tienen que ser capaces de crecer o decrecer en función de la demanda sin perjudicar a la relación coste-beneficio.

Además, en España parece que la dificultad para encontrar trabajo está motivando a emprendedores a materializar sus ideas en un negocio. La idea de crear una empresa propia ha dejado de ser una poco común para convertirse en una salida alternativa. Si ese negocio necesita de las TIC o incluso reside en la *Web*, los servicios proporcionados por los proveedores *Cloud* reducen considerablemente la barrera de entrada que podría suponer el proveerse de los recursos computacionales necesarios. *Cloud* y su modelo *pay-as-you-go* permite optimizar en este sentido los recursos: se paga por los servicios que se usen y por el tiempo que se usen, los cuales pueden ser aumentados o disminuidos de forma rápida, fácil e ilimitada desde cualquier dispositivo que tenga conexión a Internet y sin requerir la intervención del proveedor de los mismos.

Por tanto, se disponen de todos los ingredientes necesarios para el caldo de cultivo de las *startups*, empresas con poco capital, ágiles y basadas en la tecnología. De forma que, para cualquier persona formada en tecnologías informáticas buscando empleo, donde no lo hay, no hay nada que pueda ser más motivante que estudiar cómo sacar provecho de los servicios que ofrece la nube y montar su propia *startup*. Entre los servicios existentes en la nube, no solo son de interés los servicios avanzados para el desarrollo y despliegue de aplicaciones software, sino también los servicios para gestión del ciclo de vida de las aplicaciones, es decir, *Application Lifecycle Management* (ALM). Estos servicios ALM permiten llevar a cabo un proceso de desarrollo ágil (*Agile Software Development*) y complementarlo con las buenas prácticas promovidas por la nueva tendencia DevOps. Entre las buenas prácticas DevOps se encuentra la integración continua (compilación, ejecución de pruebas y despliegue automáticos) la cual favorece el principal principio de las metodologías ágiles, es decir, entregas de *software* funcional muy frecuentes.

1.2 Objetivos

El objetivo principal de este Proyecto Fin de Grado (PFG) es estudiar y poner en práctica “servicios avanzados para la construcción de aplicaciones en la nube” utilizando la PaaS Microsoft Azure. Dichos servicios avanzados se configurarán para definir una solución PaaS que cubra el desarrollo de aplicaciones software y la gestión del ciclo de vida de las aplicaciones. Además, este PFG quiere verificar que la solución PaaS satisface los siguientes atributos: *time to market (TTM)*, coste, interoperabilidad y escalabilidad.

Para la consecución de este objetivo general, se plantean los siguientes objetivos específicos:

OBJ1. Crear una aplicación móvil, *frontend* y *backend*, que utilice un conjunto de servicios ofertados por la plataforma *Cloud* Microsoft Azure. El desarrollo de la aplicación debe adecuarse a los patrones *Cloud* apropiados, con el objetivo de poder verificar la mejora de los atributos de calidad que se desean cumplir y que por orden de importancia son:

- **Time to market**: el desarrollo de la aplicación no debe superar los 6 meses de duración que es el margen que se ha establecido para la entrega del PFG.
- **Coste**: no debe superar los 80 Euros mensuales disponibles en la suscripción de Microsoft Azure disponible para la realización de este PFG.
- **Interoperabilidad**: deben poder comunicarse sistemas heterogéneos.
- **Escalabilidad**: debe gestionar convenientemente incrementos de carga.

OBJ2. Administrar el ciclo de vida de la aplicación con la suite de herramientas *Application Lifecycle Management* (ALM) de Microsoft Visual Studio Team Services (VSTS). El PFG aplicará una gestión ágil y las buenas prácticas (automatización del proceso de desarrollo y despliegue, control de versiones, integración continua) promovidas por DevOps [4].

1.3 Estructura del documento

El resto del documento está organizado en las siguientes capítulos:

- **Capítulo 2: Conceptos Previos**

Este capítulo proporciona los conocimientos necesarios para entender el PFG. Este capítulo describe el paradigma *Cloud Computing*, la PaaS Microsoft Azure, Android como sistema operativo para dispositivos móviles, las metodologías de desarrollo de software ágiles, el enfoque DevOps, y por último el entorno de desarrollo Visual Studio Team Services (VSTS).

- **Capítulo 3: Caso de Estudio**

Este capítulo describe el caso de estudio y la arquitectura que se han diseñado para poner en práctica “servicios avanzados para la construcción de aplicaciones en la nube” de la PaaS Microsoft Azure y validar la consecución de los objetivos del PFG. Respecto a la arquitectura, se enumeran los principales requisitos funcionales y arquitectónicos que debe cumplir el sistema, se definen las principales decisiones de diseño adoptadas de entre las recomendadas para la nube y se muestran las vistas arquitectónicas más representativas para describir el sistema creado. Finalmente, se han descrito los servicios que se han utilizado para el desarrollo de la aplicación. Estos son los siguientes: servicio de almacenamiento, servicio de automatización, servicio de aplicaciones (API) y plan de servicio de aplicaciones.

- **Capítulo 4: Desarrollo de una aplicación Android utilizando los servicios de Microsoft Azure**

Este capítulo detalla los dos principales componentes de la aplicación, es decir, el servicio API REST Java (*backend*) y el cliente REST Android (*fronten*) que se han desarrollado.

- **Capítulo 5: Administración del ciclo de vida de la aplicación con Visual Studio Team Services**

Este capítulo incluye una serie de guías de asistencia a la utilización de VSTS para llevar a cabo la gestión del ciclo de vida de la aplicación. En todas ellas se ha tomado como ejemplo el caso de estudio.

- **Capítulo 6: Conclusiones y Trabajos Futuros**

Este capítulo resume las principales contribuciones del PFG y analiza las futuras investigaciones.

2 Conceptos Previos

Este capítulo introduce los conceptos y conocimientos de relevancia para este PFG en lo que respecta al paradigma *Cloud Computing* y la solución *Cloud Microsoft Azure*. Además, este capítulo describe las características del sistema operativo Android, ya que el caso de estudio diseñado para poner en práctica “servicios avanzados para la construcción de aplicaciones en la nube” de la PaaS Microsoft Azure consiste en una aplicación para dispositivos móviles Android. Finalmente, se describen las metodologías de desarrollo ágil de software y DevOps que se han adoptado para el desarrollo de este PFG, así como el entorno de desarrollo que se ha utilizado para su soporte, Visual Studio Team Services.

2.1 Cloud Computing

En los últimos 50 años [5], la arquitectura software en su evolución ha ido pasando, principalmente, por las diferentes etapas que se muestran en la Fig. 2 y que se van a describir a continuación.

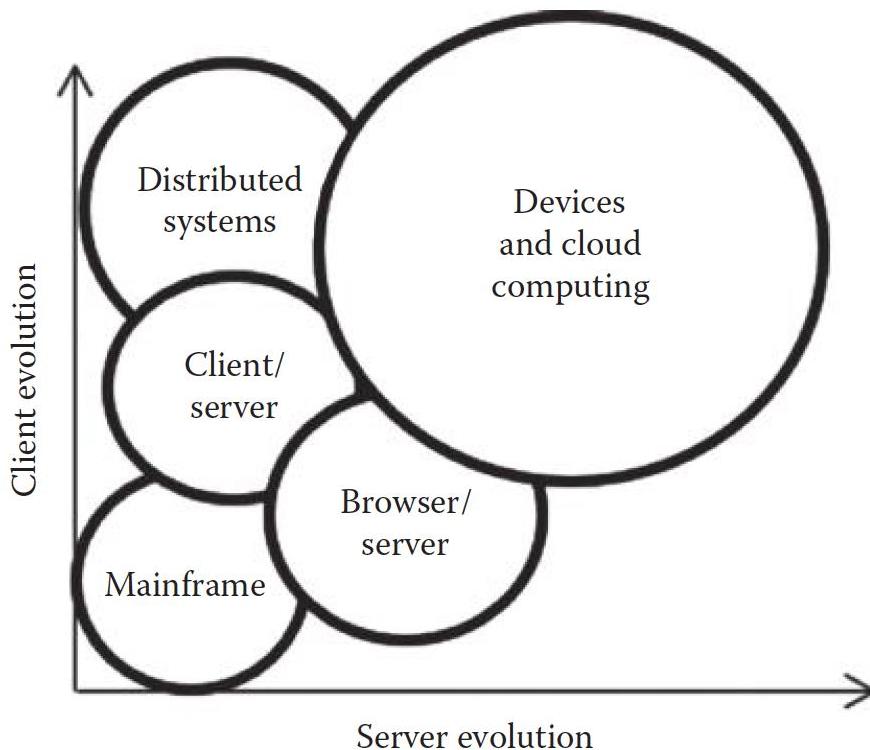


Fig. 2 Evolución de la arquitectura de las aplicaciones [5]

En la etapa *Mainframe*¹, las personas se concentraban en la optimización de los limitados recursos de los sistemas *host*² debido a que sus terminales no tenían apenas capacidad de proceso, es decir, todas las tareas computacionales eran llevadas a cabo en los *hosts*. Los administradores de los *mainframe* eran los guardianes de todos los datos en todos los sistemas y nada podía ser hecho sin contar con ellos, lo que era un cuello de botella.

Cuando se expandió el uso de ordenadores personales, como no pararon de crecer sus capacidades de procesamiento y almacenamiento, el trabajo era efectuado en la propia máquina cliente. Los sistemas distribuidos impulsaron esta idea al extremo y eliminaron la necesidad de servidores centralizados. A priori era una ventaja pero, por otra parte, el desarrollo, mantenimiento y gestión de estos sistemas es extremadamente complejo y caro.

Posteriormente, cuando los navegadores (*browsers*) e Internet maduraron, los *browsers* reemplazaron a los programas de escritorio y la computación, y el almacenamiento se volvió a centralizar llevándose a cabo en el lado servidor. Siendo su mayor problema el garantizar la seguridad

En 1999, Marc Benioff fundó Salesforce e introdujo el concepto de SaaS (*Software As A Service*), que permitía a las empresas no tener que hacer enormes inversiones para construir y mantener infraestructuras de sistemas y, a la vez, no tener que distribuir software a los clientes. Sin embargo, los servicios proporcionados por Salesforce se limitaban a un reducido número de clientes. El reto estaba en ampliar la idea de Benioff y conseguir que un mayor número de servicios fuesen accesibles por un mayor número de usuarios. Llevar a cabo esta idea no era fácil, ni estaba al alcance de muchas compañías, ya que hay que construir una plataforma que proporcione potencia computacional casi ilimitada, almacenamiento eficiente y fiable, rendimiento en red sin precedentes, seguridad, y tarifas lo suficientemente atractivas como para atraer usuarios y que pasen de utilizar software a consumir servicios.

En 2006, Amazon con EC2 (*Elastic Computing Platform*) se convirtió en pionero en *Cloud Computing*, concretamente, en el primer proveedor de una plataforma *Cloud* abierta. Pronto, muchas compañías comenzaron a migrar sus sistemas a EC2 y Amazon se convierte en sinónimo de *Cloud Computing*.

¹ *Mainframe* (Computadora central) es una computadora grande, potente y costosa, usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos; por ejemplo, para el procesamiento de transacciones bancarias.

² *Host* es el término usado en informática para referirse a las computadoras conectadas a una red, que proveen y utilizan servicios de ella. Los usuarios deben utilizar anfitriones para tener acceso a la red.

En 2007, la tecnología creada por Apple³ también influye, aunque no lo parezca, al desarrollo de *Cloud Computing* al inundar el mercado de iPhones y iPads. Las funciones que antes se hacían con ordenadores personales, como agenda, compartir ficheros, vídeos o fotografías, almacenar y escuchar música, correo electrónico, etc., ahora se hacen mediante dispositivos móviles desde cualquier lugar y en cualquier momento. Surgen problemas tales como, por ejemplo, ¿Cómo dar acceso seguro? ¿Cómo mantener los datos sincronizados? Es a partir de este momento, cuando los departamentos de TI de las empresas se ven en la necesidad de adoptar *Cloud Computing* para tratar de dar solución a esta necesidad y a los problemas derivados.

En 2008, Microsoft anunció su participación en el negocio de las plataformas *Cloud* con Microsoft Azure. El compromiso de la compañía con la plataforma *Cloud* es total, ya que ha comprometido enormes recursos para proporcionar una plataforma de primer nivel desarrollando tecnologías *Cloud*, construyendo enormes centros de datos, e incluso transfiriendo el núcleo de su negocio, tal como Office, a la plataforma *Cloud*.

En cualquier caso, en la adopción de *Cloud Computing* también existe oposición. Cuando Internet se estaba alzando como una gran innovación tecnológica existían ciertos sectores que la rechazaban ondeando la bandera de la seguridad. Eso mismo está sucediendo ahora con *Cloud Computing* [6] aludiendo que no es una tecnología madura. Los diagramas *Hype Cycle* de Gartner tratan de mostrar las tendencias tecnológicas del mercado y en qué punto de madurez se encuentra cierta tecnología, en base a la cual, las empresas pueden decidir si esperar o adelantarse y arriesgarse para tomar una posible posición ventajosa en el mercado. La realidad es que estos diagramas muestran que *Cloud Computing* está solo empezando. Pero cada vez más empresas y usuarios individuales se están dando cuenta de las ventajas de *Cloud Computing*, lo que está favoreciendo que la adopción de este paradigma vaya en aumento, especialmente en el caso de *startups* y pequeñas y medianas empresas. Las grandes empresas están tardando en adoptar este paradigma debido a la complejidad que conlleva migrar años de arquitecturas heredadas, infraestructuras, desafíos organizacionales, etc. En cualquier caso, se estima que para el 2020 alcanzará una escala de US\$240 billones. Todo ello apunta a que es el momento de moverse a *Cloud Computing* [7].

2.1.1 Definición

En un centro de proceso de datos (CPD)⁴ hay que gestionar la compra e instalación de hardware, virtualización, la instalación del sistema operativo y cualquier otra aplicación que se

³ <https://www.apple.com/es/>

⁴ CPD es una sala que aglutina los grandes sistemas informáticos y equipos de telecomunicaciones de una gran empresa, y que suelen estar protegidos por altas medidas de seguridad.

requiera, configurar la red, configurar el *firewall* y configurar almacenamientos para los datos, además del futuro mantenimiento de todo ello. En esencia, una gestión completa de los servicios *software* y *hardware* de una organización. A este modelo de gestión se le conoce como *on-premises*. Este modelo requiere mucho tiempo y dinero, independientemente que luego se use o no. La única ventaja es que se puede elegir el software y hardware [8].

Por otra parte, *Cloud Computing* proporciona una alternativa moderna al tradicional CPD *on-premises* en la que un proveedor público de *Cloud* es el único responsable de la compra y mantenimiento de *hardware*, lo que se conoce como el modelo *off-premises*. Por lo general, los entornos *Cloud* permiten a los usuarios una forma fácil de gestionar los recursos de computación, almacenamiento, red y aplicaciones mediante un portal *online*. Al cliente se le aplica el modelo de *pay-as-you-go* o *pay-per-use*, es decir, que va a pagar en función del nivel de uso que necesite. Si el cliente necesitase más recursos, estos pueden ser provisionados de forma fácil y elástica con un esfuerzo mínimo y sin interacción del proveedor. Es decir, permite alquilar el acceso a recursos *hardware* y *software* que de otra forma hubiese sido caro adquirir con el único inconveniente que estamos limitados a la oferta del proveedor [8].

Como **ventajas** del *Cloud Computing* se puede destacar:

- Nula inversión en adquisición y en mantenimiento de infraestructuras.
- Posibilidad de aumentar o disminuir el consumo de los recursos *hardware* o *software* inmediatamente y, en algunos casos, automáticamente.
- Pago en función de la demanda y, por tanto, permitiendo un control más eficiente de los gastos.
- Disfrutar de los procedimientos de seguridad, disponibilidad y rendimiento más avanzados de proveedores con experiencia y conocimientos en este tipo de servicios.
- Acceso a los recursos desde cualquier punto geográfico.

Como **desventajas** del *Cloud Computing* se puede destacar:

- Percepción de inseguridad ya que los datos y lógica de negocio están fuera de la empresa.
- Dificultad para integrar los recursos *Cloud* con los sistemas *on-premises*.
- Disponibilidad sujeta a paradas por mantenimiento programadas por el proveedor y no por el cliente.
- Dos posibles puntos de fallo externos a la infraestructura: proveedor de servicios *Cloud* y el proveedor de Internet.

En lo que respecta a la definición formal de NIST, *Cloud Computing* es un modelo que permite el acceso universal, adecuado y bajo demanda a un conjunto de recursos de cómputo configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente provistos y puestos a disposición del cliente con un mínimo esfuerzo de gestión y de interacción con el proveedor del servicio. Además, el modelo *Cloud Computing* cumple 5 características esenciales y dispone de 3 modelos de servicio y 4 modelos de despliegue.

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models [9].

2.1.2 Modelos de servicio

En función del grado de visión y control al que el usuario del servicio tiene acceso, se distinguen 3 modelos de servicio: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) y *Software as a Service* (SaaS). La Fig. 3 muestra estos modelos de servicios en función del grado de gestión del usuario y el proveedor en la nube, desde una completa gestión del usuario (*on-premises*) a una gestión total del proveedor (SaaS).

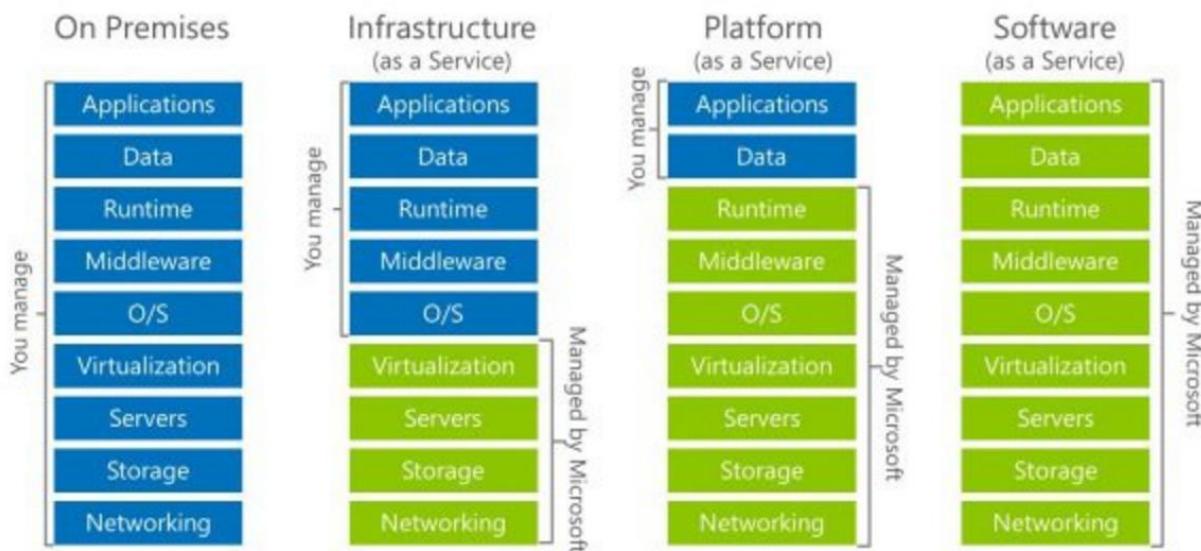


Fig. 3 Responsabilidades de usuario y proveedor servicios en modelos servicio *Cloud* [8]

2.1.2.1 *Infrastructure as a Service (IaaS)*

El uso de un modelo IaaS implica que los clientes renuncian a usar sus propios equipos físicos y usan los recursos virtuales que le proporciona el proveedor de servicios *Cloud*. El cliente no gestiona ni controla la infraestructura subyacente (servidores, *routers*, etc.), pero tiene control sobre los sistemas operativos, *middleware*⁵, política de almacenamiento y aplicaciones desplegadas.

The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls) [9].

2.1.2.2 *Platform as a Service (PaaS)*

El modelo de PaaS está construido sobre IaaS. Los proveedores PaaS ofrecen acceso a un entorno de programación y ejecución que se sustenta sobre una infraestructura escalable de componentes *hardware* y de *middleware*, sobre la que el cliente no tiene ningún conocimiento ni control. Una PaaS permite que los clientes desarrollen y ejecuten sus propias aplicaciones sobre un entorno ajeno al cliente, ofrecido por el proveedor del servicio. Por lo tanto, el cliente ni gestiona ni controla la infraestructura (ni servidores, ni sistemas operativos, ni almacenamiento, ni ningún tipo de elementos de red o seguridad, etc.), pero tiene control de las aplicaciones desplegadas y de la configuración del entorno de ejecución de las mismas.

The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.³ The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment [9].

⁵ *Middleware* o lógica de intercambio de información entre aplicaciones es un *software* que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, *hardware* y/o sistemas operativos.

2.1.2.3 Software as a Service (SaaS)

El modelo SaaS está construido sobre una PaaS. Los proveedores SaaS ofrecen a los usuarios acceso a un conjunto de aplicaciones específicas que son ejecutadas en las infraestructuras del proveedor y controladas por dicho proveedor. El cliente como máximo puede modificar algunos parámetros de configuración de la aplicación.

Office 365 es buen ejemplo de oferta SaaS. Los subscriptores pagan una tasa de suscripción anual o mensual y de esta forma obtienen Outlook, OneDrive y el resto de la Suite Microsoft Office. Los subscriptores siempre tienen disponible la última versión. Comparado con instalar y actualizar Office cada año, esto es menos caro y requiere menos esfuerzo para mantenerse a la última. Otros ejemplos populares de SaaS son Google Docs, Salesforce, Dropbox, Gmail.

The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

2.1.3 Modelos de despliegue

Dependiendo de las necesidades de cada compañía los servicios se pueden obtener mediante un proveedor externo (nube pública), una infraestructura exclusiva para la compañía (nube privada) o de una comunidad específica (nube comunitaria), o hacer uso de las anteriores (nube híbrida).

2.1.3.1 Private Cloud

La nube privada o *Private Cloud* consiste en una infraestructura *Cloud* implantada para una sola empresa, que se puede gestionar de forma interna o por un proveedor externo.

Las nubes privadas permiten utilizar soluciones de seguridad avanzada, una disponibilidad y tolerancia a los fallos que no son posibles en la nube pública. Sin embargo, puesto que se trata de soluciones independientes, crear una nube privada sigue exigiendo una inversión significativa y, por consiguiente, no ofrece los beneficios de coste a corto plazo que ofrece la nube pública.

The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

2.1.3.2 Public Cloud

La nube pública o *Public Cloud* hace referencia al modelo estándar de *Cloud Computing*, en donde el proveedor de estos servicios pone a disposición de cualquier usuario en Internet su infraestructura, es decir, permite al usuario el uso de su software o hardware en forma libre o mediante un pago en función del uso de los mismos.

The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

2.1.3.3 Community Cloud

La infraestructura de la nube comunitaria o *Community Cloud* es compartida por varias organizaciones que comparten preocupaciones similares acerca de algún tema específico como, por ejemplo, seguridad, investigación, etc. Puede ser administrada por la organización o por un tercero y puede existir dentro o fuera de la misma organización. Una nube comunitaria es similar a una nube pública excepto que su acceso está limitado a una comunidad específica de consumidores.

The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

2.1.3.4 Hybrid Cloud

La nube híbrida o *Hybrid Cloud* combina los modelos de nube privada y pública, las cuales permanecen como entidades distintas pero están enlazadas. Un ejemplo típico sería una organización que puede almacenar datos sensibles de los clientes de la organización en una aplicación de la nube privada pero interconectar dicha aplicación a otra aplicación de la nube pública como SaaS.

The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

2.1.4 Características esenciales

Entre las características esenciales que debe cumplir modelo *Cloud* se encuentran la reutilización de recursos, el autoprovisionamiento bajo demanda, el acceso a través de internet, la elasticidad y el servicio medible. Todas ellas soportadas por un modelo de servicio y de despliegue (ver Fig. 4). A continuación se describen en qué consisten cada una de estas características.



Fig. 4 Definición de nube según NIST [9]

2.1.4.1 Autoprovisionamiento bajo demanda

El usuario puede provisionarse de recursos del proveedor unilateralmente sin que el proveedor actúe de ninguna forma en dicha transacción.

A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider [9].

2.1.4.2 Accesibilidad a través de Internet

Los recursos están disponibles en la red y son accesibles mediante mecanismos estándar y desde plataformas heterogéneas (por ejemplo: ordenadores, teléfonos móviles o tabletas).

Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations) [9].

2.1.4.3 Reutilización de recursos

Los recursos son puestos a disposición de los consumidores siguiendo un modelo de multipropiedad, asignándose y reasignándose dispositivos físicos o lógicos atendiendo a la demanda de dichos consumidores. En este sentido el usuario no tiene un estricto control del lugar exacto en el que se encuentra su información, aunque sí debe poder especificar un ámbito mínimo de actuación (por ejemplo: un país, un estado o un centro de proceso de datos concreto).

The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth [9].

2.1.4.4 Elasticidad

El proceso de asignación dinámica de recursos (para crecer o decrecer y durante cualquier cantidad de tiempo) ha de ser llevado a cabo rápidamente de manera que sea un proceso totalmente transparente al usuario, que debería acabar con la sensación de que los recursos de que dispone son ilimitados.

Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time [9].

2.1.4.5 Servicio medible

Para poder garantizar los puntos anteriores, los sistemas *Cloud* deben poseer alguna forma de medir si los recursos de que dispone un usuario son suficientes para la actividad que están llevando a cabo o si, por el contrario, hay que proveerle de más recursos. Su uso debe ser medido, monitorizado y reportado para ser lo más transparente posible.

Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service [9].

2.2 Patrones y Tácticas de Diseño

Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. Además, los patrones se descubren en la práctica, no se inventan [10].

Por otra parte, las tácticas para el diseño de arquitecturas de software son decisiones de diseño que influyen en el control de la respuesta de un atributo de calidad. A diferencia de los patrones de diseño, las tácticas son soluciones menos detalladas y están enfocadas a atributos de calidad específicos. Las tácticas se aplican en conjunto con los patrones durante el diseño de la arquitectura de un sistema [10].

A continuación, de entre los cientos de patrones y tácticas arquitectónicas existentes, se van a desarrollar únicamente los patrones y tácticas que son relevantes en este PFG.

2.2.1 Estilo arquitectónico REST

Al construir una solución arquitectónica para la nube debemos tener presente que todo puede fallar y fallará. A nuestro favor tenemos que la infraestructura *Cloud* está diseñada sobre todo para la alta disponibilidad, la escalabilidad y es tolerante al particionado⁶ por naturaleza.

⁶ Tolerancia al particionado o *partition tolerance* es la garantía que el sistema sigue funcionando a pesar de que haya sido partido por fallo de red.

Teniendo esto presente, el secreto para no fracasar en la nube es utilizar APIs REST (*Representational State Transfer*) [5].

Los APIs exponen un conjunto de datos y funciones que facilitan las interacciones entre programas de ordenador y les permite intercambiar información (ver Fig. 5). Las API *Web* son utilizadas por los programas cliente para comunicarse con los servicios *Web* [11].

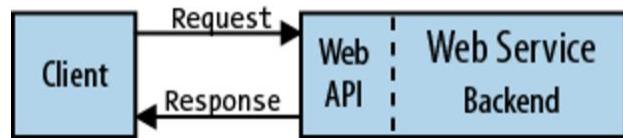


Fig. 5 Interacción entre cliente y servidor mediante API Web [11]

Un API REST es un API *Web* que respeta el estilo arquitectural REST. Un servicio *Web* es RESTful si tiene un API REST. Un API REST consta de un conjunto de recursos interconectados. A este conjunto de recursos se lo conoce como el modelo de recursos del API REST. Si las API REST están bien diseñadas, pueden atraer clientes para utilizar sus servicios *Web*.

El término REST se originó en el año 2000, en una tesis doctoral sobre la *Web* escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP [11]. La motivación de esta tesis fue capturar los principios arquitectónicos de la *Web* que la han hecho tener tanto éxito y que la han permitido crecer tanto y en tan poco tiempo. A este conjunto de principios arquitectónicos le llamo REST.

Por tanto, REST es un estilo de arquitectura *software* para sistemas hipermedia distribuidos como la *Web*, siendo las restricciones que lo rigen las siguientes (ver Fig. 6):



Fig. 6 Restricciones del estilo arquitectónico REST

- **Cliente-servidor:** separar lo que es competencia del cliente y el servidor es clave. La *Web* se basa en el sistema cliente-servidor. Se deben implementar y desplegar independientemente, en cualquier lenguaje o tecnología, siempre y cuando respeten el interfaz uniforme.
- **Interfaz uniforme:** las interacciones entre los componentes *Web* (clientes, servidores, etc.) dependen de la uniformidad de sus interfaces. Si alguno de los componentes no lo respeta, el sistema de comunicación de la *Web* se rompe. Los componentes *Web* interoperan consistentemente si cumplen las siguientes restricciones:
 - **Identificación de recursos:** cada recurso de la *Web* debe ser accesible por un identificador único, como una URI (*Uniform Resource Identifier*).
 - **Manipulación de recursos a través de representaciones:** los clientes manipulan representaciones de recursos. El mismo recurso se puede representar en diferentes formas para diferentes clientes. Por ejemplo, un documento se puede representar como HTML para un navegador *Web* y como JSON⁷ para un programa. La idea clave es que la representación es una forma de interactuar con el recurso pero no es el recurso en sí. De esta forma, se puede representar el recurso de diferentes formas sin cambiar su identificador.
 - **Mensajes auto-descriptivos:** el estado que se desea para un recurso se puede transmitir dentro del mensaje de petición del cliente. El estado actual del recurso se puede representar dentro del mensaje de respuesta devuelto por el servidor. Por ejemplo, se puede utilizar un mensaje de petición para transferir una representación que sugiera una actualización (un estado nuevo) de una página gestionada por un servidor. Es cosa del servidor aceptar o denegar la petición del cliente. Los mensajes auto-descriptivos pueden incluir metadatos para transmitir detalles adicionales relacionados con el estado del recurso, el formato de representación, tamaño, etc. Un mensaje HTTP proporciona cabeceras para organizar los diversos tipos de metadatos en campos uniformes. Además, define un conjunto pequeño de operaciones siendo las más importantes POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD en bases de datos (*Create, Read, Update, Delete*) que se requieren para la persistencia de datos.
 - **HATEOAS (*Hypermedia As the Engine of Application State*):** la representación del estado de un recurso debe poder incluir enlaces a recursos relacionados.

⁷ JSON (*JavaScript Object Notation*) es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

- **Sistema de capas:** posibilita introducir transparentemente intermediarios tales como *proxies* y *gateways* entre los clientes y los servidores siempre que hagan uso del interface uniforme de la *Web*. De esta forma, estos intermediarios podrán interceptar la comunicación entre cliente y servidor si es necesario, generalmente para reforzar la seguridad, balanceo de carga, sistemas de cache, etc.
- **Cacheable:** las respuestas a una petición deben poder ser cacheables. En este caso, el cliente puede acceder a una caché y reutilizar la respuesta más tarde si se hace una petición equivalente. En general, la caché se encuentra en el cliente aunque nada impide que este en el servidor también. La ventaja de añadir esta restricción es que se evitarán determinadas peticiones al servidor, mejorando así la eficiencia y escalabilidad. Se va a reducir el tiempo medio de espera de una serie de interacciones. La desventaja de esta restricción es que puede inducir a un mal funcionamiento de una aplicación si los datos obtenidos del caché difieren de los que se hubiesen obtenido realizando la petición directamente al servidor.
- **Sin estado:** un servidor *Web* no debe memorizar el estado de sus aplicaciones cliente. Como consecuencia, cada cliente debe incluir toda la información contextual que considere relevante en cada interacción con el servidor *Web*. Esta restricción mejora la escalabilidad. La desventaja de esta restricción es que puede empeorar el funcionamiento de la red porque incrementa el tráfico de datos repetidos al enviar una serie de peticiones. Esto ocurre porque los datos no pueden quedarse almacenados en el servidor identificando un contexto determinado de comunicación. Además poniendo el estado de la aplicación en el lado del cliente, se reduce el control para obtener un comportamiento consistente en la aplicación. La aplicación se hace muy dependiente de una correcta implementación de la semántica en las distintas versiones del cliente.
- **Código bajo demanda:** está restricción es opcional, consiste en permitir a los clientes descargar y ejecutar código en forma de *applets*, *plug-ins* y *scripts*. Esto simplifica el lado del cliente porque reduce el número de funcionalidades que tiene que tener implementadas al crearse. Las funcionalidades se pueden descargar posteriormente aumentando así la extensibilidad del sistema. Su principal desventaja es que tiende a establecer un acoplamiento de tecnologías entre los servidores *Web* y sus clientes, ya que estos tienen que ser capaces de entender y ejecutar el código que se bajan del servidor.

2.2.2 Patrón particionado

Es fácil escalar la capa *Web* de una aplicación en la nube simplemente añadiendo o quitando servidores *Web*. Sin embargo, si todos estos servidores *Web* están atacando al mismo almacén de datos, el cuello de botella se traslada de los servidores al almacén de datos y la capa de datos es la más difícil de escalar [12].

Para conseguir una capa de datos escalable se utiliza el particionado de datos. Las particiones son siempre atendidas por un solo servidor de particiones donde, cada servidor de particiones, puede atender una o más particiones. Puesto que una partición siempre la atiende un único servidor de particiones y cada servidor de particiones puede atender una o varias particiones, la eficiencia de las peticiones que se atienden se correlaciona con el estado del servidor. Es posible que los servidores que tengan mucho tráfico en sus particiones no puedan mantener un alto rendimiento.

Además, es mejor hacer el particionado al principio porque una vez que la aplicación está en producción puede ser muy difícil. En este sentido, ha de plantearse particionar los datos si se cree que se va a tener un gran volumen (cantidad de datos), velocidad (tasa de crecimiento de los datos) o variedad (diferentes tipos de datos) y en función de esto existen diferentes estrategias de partición (ver Fig. 7):

- **Particionado vertical:** consiste en partir una tabla en columnas. Un conjunto de columnas va a un almacén de datos y otro conjunto va otro.
- **Particionado horizontal:** consiste en partir una tabla en filas. Un conjunto de filas va a un almacén de datos y otro conjunto de filas va a otro diferente.
- **Particionado híbrido:** consiste en combinar los particionados verticales y horizontales.

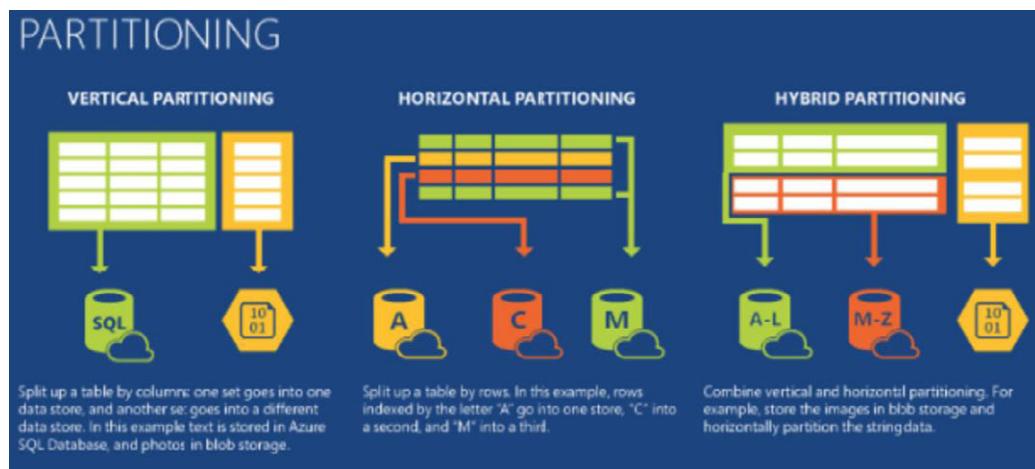


Fig. 7 Particionar almacenamientos para mejorar escalabilidad [13]

2.2.3 Patrón vista materializada

Cuando se almacenan datos, no se suele tener en cuenta cómo se leen los datos. Se suelen guardar en un formato próximo al de los datos. Sin embargo, esto tiene un efecto negativo en las consultas de estos datos [14].

Para favorecer consultas eficientes, la solución más común es generar por adelantado una vista que materialice los datos en un formato más parecido al del conjunto de resultados. El patrón vista materializada (ver Fig. 8) consiste en generar vistas prefabricadas de datos en entornos donde la fuente de datos no está en formato adecuado para las consultas y generar dicha consulta es complicado o el rendimiento de la consulta es pobre debido a la naturaleza de los datos.

Estas vistas materializadas, solo contienen los datos que necesita una consulta, permitiendo a las aplicaciones obtener rápidamente la información que necesitan. Un punto clave es que, en general, estas vistas y los datos que contienen pueden ser regenerados desde los almacenes de fuentes de datos. Una vista materializada nunca es actualizada directamente por una aplicación, de esta forma se convierte en una cache especializada. Cuando la fuente de datos de las vistas cambia, la vista debe ser actualizar para incluir la nueva información. Esto puede hacerse automáticamente bajo una apropiada planificación o cuando el sistema detecte cambios en los datos originales. En otro caso, será necesario regenerar la vista manualmente.

Estas vistas no son recomendables cuando la fuente de datos es fácil de consultar, se necesita una consistencia alta (las vistas no siempre son consistentes con los datos), la fuente de datos cambia muy rápidamente (la sobrecarga de crear estas vistas puede empeorar el rendimiento).

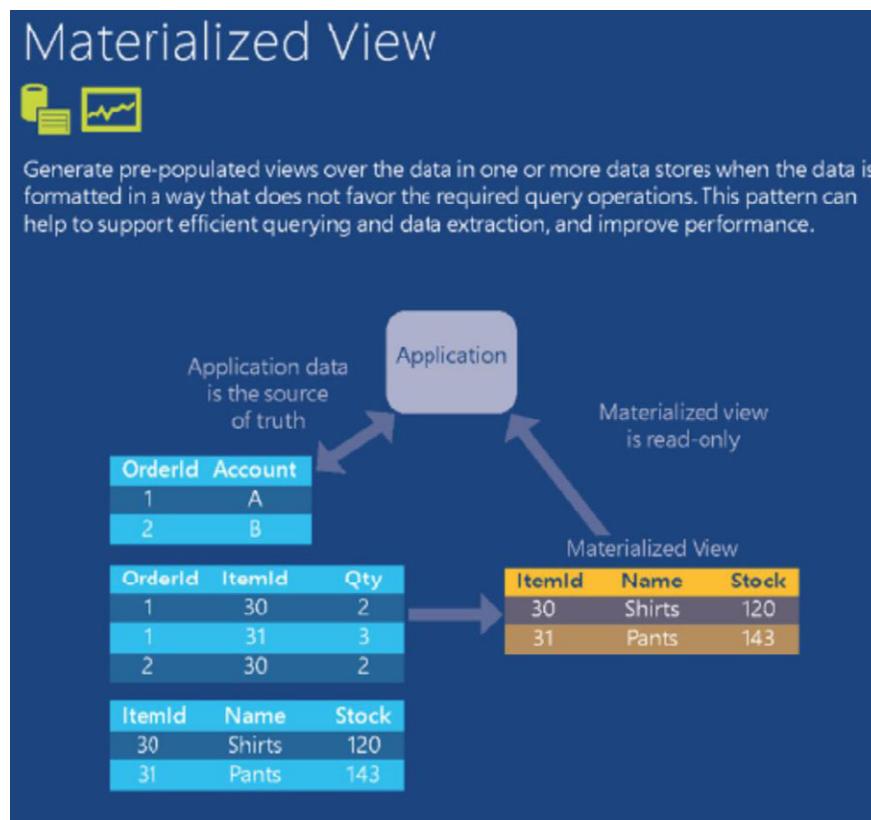


Fig. 8 Patrón vista materializada [13]

2.2.4 Táctica manejo de excepciones

Una vez se detecta una excepción, el sistema debería gestionarla de alguna manera. Lo más fácil sería que el programa dejase de funcionar lo cual es la peor opción desde el punto de vista de la disponibilidad, resistencia, usabilidad y testeabilidad. En este sentido, existen otras alternativas que dependen del entorno de programación y donde lo más sencillo suele ser devolver un simple código de error.

2.3 Microsoft Azure

Microsoft Azure es una plataforma de servicios de Microsoft en una nube pública. Es decir, cualquiera con una conexión a Internet puede entrar al portal⁸ de Microsoft Azure y disponer de los servicios ofertados para crear sus aplicaciones [15].

Microsoft Azure es también una plataforma en la nube que, integrada con Visual Studio Team Services⁹ (VSTS), nos permite realizar *Application Lifecycle Management* (ALM), es decir, administrar el ciclo de vida de nuestra aplicación. ALM divide el ciclo de vida en tres áreas distintas (ver Fig. 9): gobierno (aborda las decisiones y gestión de proyecto de una aplicación), desarrollo (proceso iterativo de crear la aplicación) y operaciones (trabajo requerido para ejecutar y gestionar la aplicación). Es decir, engloba no solo el ciclo de desarrollo del software (gestión de requisitos, arquitectura del software, programación, pruebas, mantenimiento), sino también gestión de proyecto y gestión de versiones.

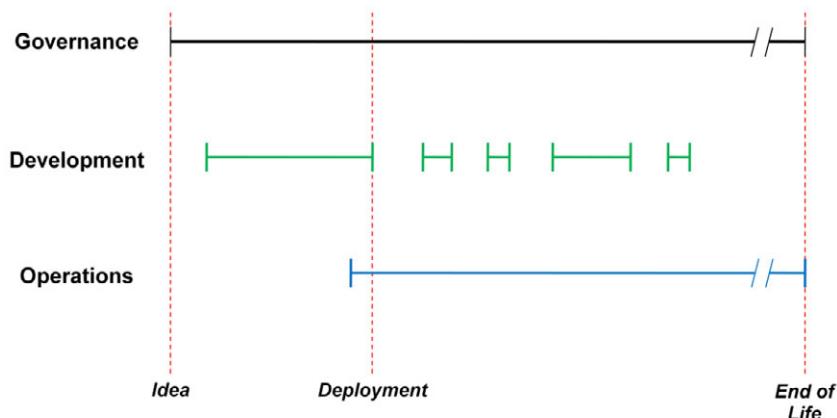


Fig. 9 Aspectos fundamentales de ALM [16]

⁸ <https://portal.azure.com/>

⁹ <https://www.visualstudio.com/>

Microsoft Azure proporciona servicios IaaS y PaaS permitiéndonos crear y entregar nuestro propio software como servicio (ver Fig. 10). Es una plataforma compatible con casi todo tipo de tecnologías. Soporta Oracle, Linux, PHP, Node.js, Android, MySQL, iOS, Git, etc.

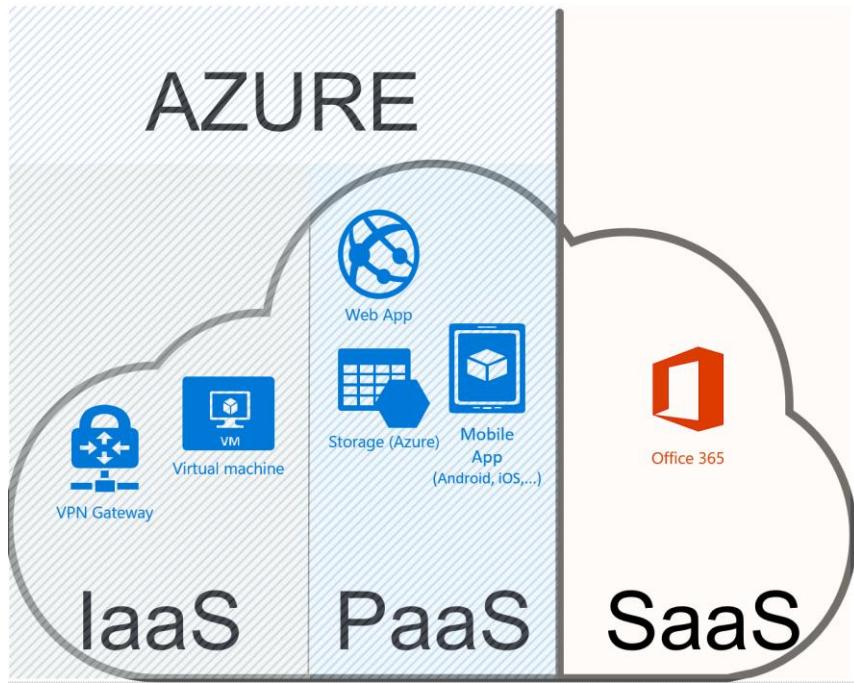


Fig. 10 Servicios IaaS, PaaS de Azure

Microsoft también es el único proveedor reconocido por Gartner como líder en seis Cuadrantes Mágicos relacionados con servicios en la nube [3]. Dado su reconocimiento, Microsoft Azure ha sido la plataforma *Cloud* elegida para desarrollar este PFG. En las siguientes secciones se van a detallar servicios y aplicaciones que nos ofrece la plataforma.

2.3.1 Azure PowerShell

Microsoft Azure nos proporciona Azure PowerShell [17]. Azure PowerShell es un conjunto de módulos que proporcionan cmdlets (*scripts* en lenguaje PowerShell) para gestionar Microsoft Azure con Windows PowerShell. Estos cmdlets se pueden utilizar para las mismas tareas que se pueden efectuar desde el portal de Microsoft Azure. De esta forma, se pueden crear *scripts* y automatizar todo lo que respecta a la gestión de Microsoft Azure. Para instalar Azure PowerShell y poder utilizarlo hay que seguir los siguientes pasos:

- Desde un SO Windows, se descarga e instala Azure PowerShell desde el sitio de Microsoft ¹⁰.
- Se abre una ventana de PowerShell con permisos de administrador, se abre un menú contextual del programa y se selecciona la opción "*Ejecutar como administrador*".
- Se instalan los módulos de Microsoft Azure Resource Manager (Microsoft Azure RM) (ver Fig. 11).

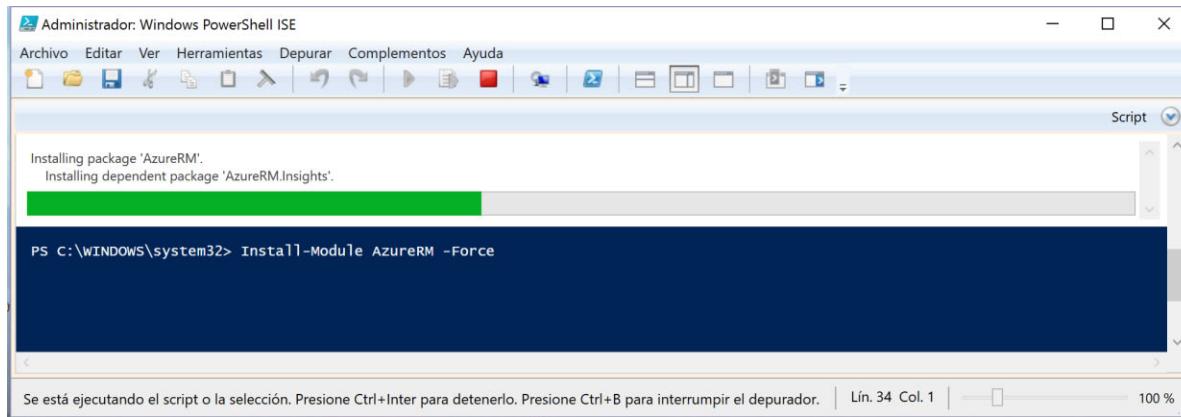


Fig. 11 Instalación Azure Resource Manager para PowerShell

- Se conecta a Microsoft Azure RM (ver Fig. 12) en forma interactiva (sale una ventana de conexión).

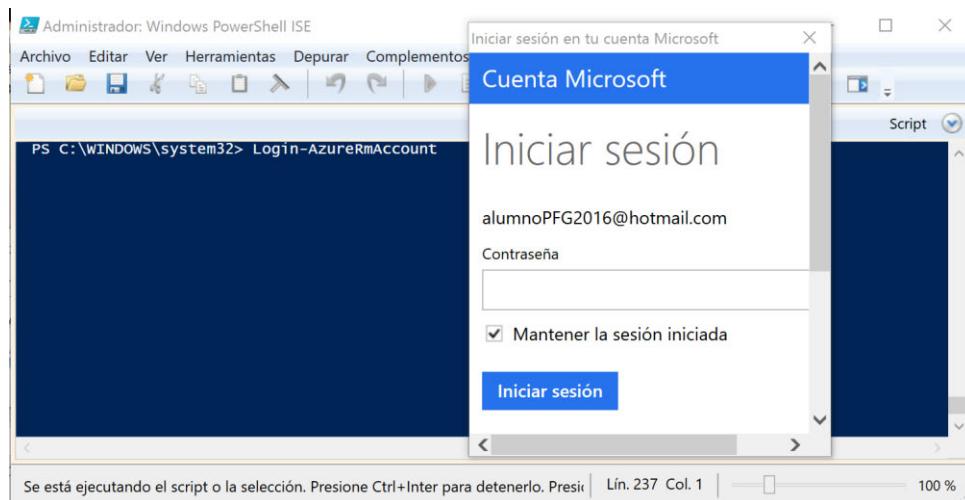


Fig. 12 Conexión interactiva a Microsoft Azure Resource Manager con Azure PowerShell

¹⁰ <https://azure.microsoft.com/es-es/documentation/articles/powershell-install-configure/>

- Si se dispone de una cuenta de una afiliación (organización, centro educativo) es posible conectarse de forma no interactiva (sin necesidad de conectarse). Una cuenta de afiliación es una cuenta de un usuario gestionada desde su organización o centro educativo, y definida en el Azure Active Directory (servicio de directorio de Microsoft que facilita el trabajo con recursos de red en una forma unificada) para su organización o centro educativo. Si no se tiene una cuenta de este tipo, y se está utilizando una cuenta Microsoft para conectarse en la suscripción de Microsoft Azure, se puede crear una siguiendo los siguientes pasos:

1. Se conecta en el portal clásico de Microsoft Azure¹¹ y se accede a Active Directory (ver Fig. 13).

NAME	STATUS	ROLE	SUBSCRIPTION	DATACENTER REGION	COUNTRY OR REGIO...
Default Directory	Active	Global Administrator	Shared by all Default Direct...	Europe, United States	Spain

Fig. 13 Gestión Active Directory desde el portal clásico de Azure

2. Se añade un usuario al directorio (ver Fig. 14). Durante su creación, se suministrará una dirección email para el usuario y un *password* temporal. Esta información es necesaria en el paso 5.

DISPLAY NAME	USER NAME	SOURCED FROM
Alejandro	alexmorarodriguez@alumnoPFG2016hotmail.onmicrosoft...	Microsoft Azure Active Directory
Alejandro Mora	alex@alumnoPFG2016hotmail.onmicrosoft.com	Microsoft Azure Active Directory
alexmorarodriguez@gmail.com	alexmorarodriguez@gmail.com	Microsoft account
alumno PFG2016	alumnoPFG2016@hotmail.com	Microsoft account
j.perezbenedi@outlook.es	j.perezbenedi@outlook.es	Microsoft account
oscar3377	oscar3377@hotmail.com	Microsoft account
verorodriguezblanco@gmail.com	verorodriguezblanco@gmail.com	Microsoft account
yessicadiaz@hotmail.com	yessicadiaz@hotmail.com	Microsoft account

Fig. 14 Usuarios en el Active Directory del portal clásico de Azure

¹¹ <https://manage.windowsazure.com/>

3. Opcionalmente, desde el portal clásico de Microsoft Azure, se selecciona “*Settings*” y después “*Administrators*”. Desde aquí, se añade al nuevo usuario como administrador (ver Fig. 15). Esto permitirá a la cuenta de afiliación gestionar su suscripción de Microsoft Azure.

NAME	SUBSCRIPTION	SUBSCRIPTION ID	ROLE
alumnoPFG2016@hotmail.com	Azure Pass	6a200011-67f5-419a-8692-baf4e8957fa1	Service administrator
alexmorarodriguez@alumnoPFG2016hotmail.com	Azure Pass	6a200011-67f5-419a-8692-baf4e8957fa1	Co-administrator

Fig. 15 Administradores de la suscripción desde el portal clásico de Microsoft Azure

4. Finalmente, se desconecta del portal clásico de Microsoft Azure y se vuelve a conectar utilizando la cuenta de afiliación. Si ésta es la primera vez que se conecta con esta cuenta, se le pedirá que cambie la contraseña.
5. Ahora ya se puede conectar de forma no interactiva:
- Se puede obtener los credenciales de una cuenta de afiliación y guardarlos en una variable (ver Fig. 16).

```
PS C:\WINDOWS\system32> $cred=Get-Credential
cmdlet Get-Credential en la posición 1 de la canal
Proporcione valores para los parámetros siguientes
```

Proporcione valores para los parámetros siguientes:

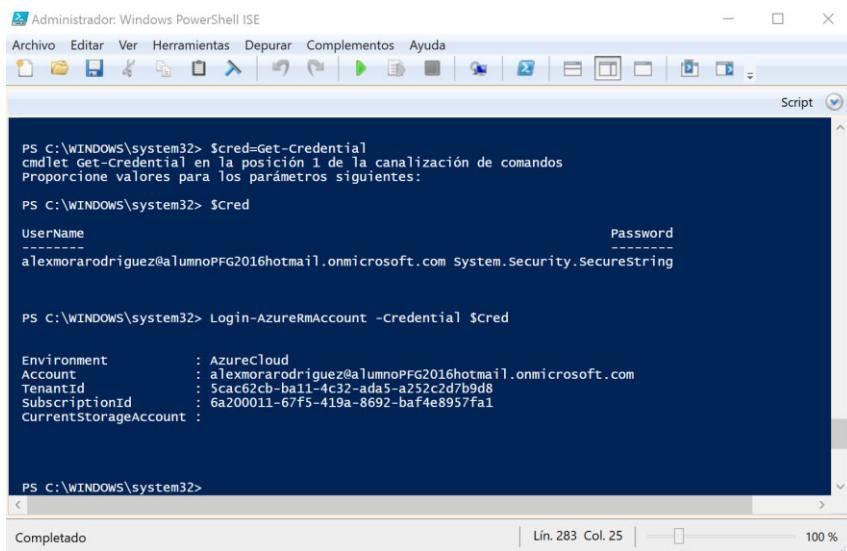
Usuario:

Contraseña:

Aceptar Cancelar

Fig. 16 Obtención credenciales de cuenta de afiliación con Azure PowerShell

- b. Se conecta a Azure RM (ver Fig. 17) en forma no interactiva (sin conexión).



```

Administrator: Windows PowerShell ISE
Archivo Editar Ver Herramientas Depurar Complementos Ayuda
Script Script

PS C:\WINDOWS\system32> $cred=Get-Credential
cmdlet Get-Credential en la posición 1 de la canalización de comandos
Proporcione valores para los parámetros siguientes:
PS C:\WINDOWS\system32> $cred
UserName _____ Password _____
----- alexmorarodriguez@alumnoPFG2016hotmail.onmicrosoft.com system.security.secureString

PS C:\WINDOWS\system32> Login-AzureRmAccount -credential $cred
Environment      : AzureCloud
Account          : alexmorarodriguez@alumnoPFG2016hotmail.onmicrosoft.com
TenantId         : 5cac62cb-ba11-4c32-ad45-a252c2d7b9d8
SubscriptionId   : 6a200011-67f5-419a-8692-baf4e8957fa1
CurrentStorageAccount : 

PS C:\WINDOWS\system32>

```

Fig. 17 Conexión no interactiva a Azure Resource Manager con Azure PowerShell

2.3.2 Administrador de Recursos (*Resource Manager*)

El administrador de recursos de Microsoft Azure permite trabajar con los recursos de una aplicación como un grupo. De esta forma, se pueden implementar, administrar y supervisar todos los recursos de una solución en forma única, repetible y coordinada. Por ejemplo, se pueden crear grupos para definir diferentes entornos como desarrollo, test y producción.

- **Recurso:** es una instancia de servicio en Microsoft Azure. La mayoría de los servicios en Microsoft Azure se pueden representar como un recurso.
- **Grupo de Recursos:** es una agrupación lógica de recursos. Las reglas más importantes a tener en cuenta con los grupos de recursos son:
 - Todos los recursos del grupo deben compartir el mismo ciclo de vida. Se implementarán, actualizarán y eliminarán de forma conjunta.
 - Cada recurso solo puede existir en un grupo de recursos.
 - Se puede agregar o quitar un recurso de un grupo de recursos en cualquier momento.
 - Se puede mover un recurso de un grupo de recursos a otro.
 - Un grupo de recursos puede contener recursos que residen en diferentes regiones.

- **Plantilla de grupo de recursos:** es un fichero JSON que permite declarativamente describir un conjunto de recursos. Estos recursos pueden después ser añadidos a un grupo de recursos nuevo o a un ya existente.

2.3.3 Servicio de Almacenamiento

Microsoft Azure proporciona un servicio de almacenamiento persistente, escalable y redundante a través de las cuentas de almacenamiento. Microsoft se encarga de todas las copias de seguridad (*backups*) y mantenimientos. Una suscripción puede tener hasta 50 cuentas de almacenamiento de hasta 500 TB cada una [18].

2.3.3.1 Cuenta de Almacenamiento (*Storage Account*)

Una cuenta de almacenamiento estándar de Microsoft Azure proporciona acceso a los servicios de almacenamiento de Microsoft Azure. Como ya se ha comentado, cada cuenta puede contener hasta 500 TB en total [19]. Entre los servicios de almacenamiento que ofrece se encuentran los *blobs*, las tablas y las colas los cuales van a ser detallados en las siguientes subsecciones.

Para crear una cuenta de almacenamiento desde la plataforma (ver Fig. 25) necesitamos seleccionar un nombre para la cuenta, un modelo de implementación (administrador de recursos/clásica), un tipo de cuenta (uso general/almacenamiento de *blobs*), un rendimiento (estándar/*premium*), un modo de replicación (ver Sección 2.3.3.2), una suscripción, un grupo de recursos y una ubicación (Oeste de Europa, Norte de Estados Unidos, etc.) del centro de datos donde se desea este alojada la información. **Si se elige el tipo de cuenta "almacenamiento de *blobs*", se debe seleccionar también el nivel de acceso (estático/dinámico) para determinar la frecuencia con que se accede a los datos y optimizar costos.** En este sentido, el nivel de acceso estático es ideal para los datos a los que se accede frecuentemente y el nivel de acceso dinámico esta optimizado para almacenar a menor precio los datos a los que se accede con menos frecuencia, como las copias de seguridad.

Una vez creada la cuenta de almacenamiento, se puede administrar desde la plataforma Microsoft Azure (ver Fig. 26) y gestionar los distintos tipos servicios de almacenamiento (ver 2.3.3).

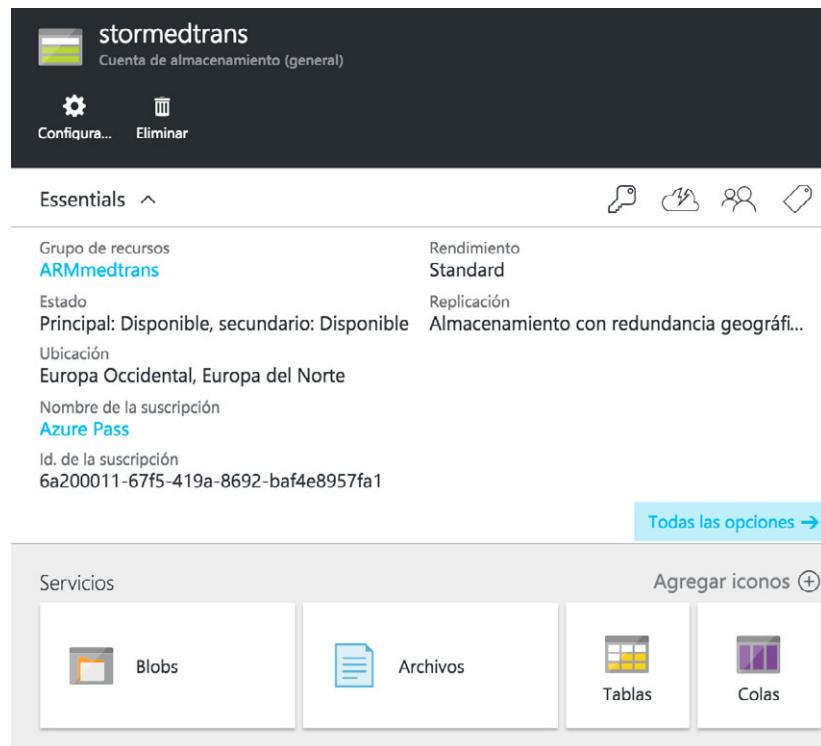


Fig. 18 Administración de la cuenta de almacenamiento en Microsoft Azure

2.3.3.1.1 *Blobs*

Los *blobs* proporcionan una forma de almacenar grandes cantidades de datos binarios no estructurados tales como video, audio, imágenes, etc. [19]. Un servicio de *blob* (ver Fig. 19) está compuesto por:

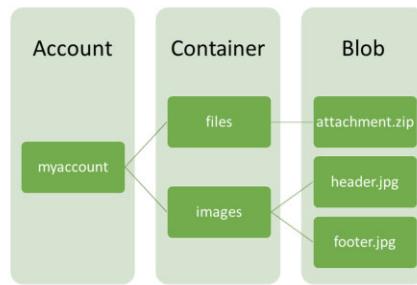


Fig. 19 Contenedor de una cuenta de almacenamiento [19]

- Formato URL: para acceder a una servicio contenedor de una cuenta de almacenamiento se utiliza el siguiente formato:

http://<storage account>.container.core.windows.net/<container>

- Cuenta de almacenamiento: necesario para acceder al contenedor.
- Contenedor: es una colección de *blobs*.
- *Blob*: es un archivo de cualquier tipo. Pueden tener un tamaño de hasta 1 TB.

2.3.3.1.2 Tablas (*Tables*)

El servicio de almacenamiento de tablas de Microsoft Azure se basa en un almacén de claves/atributos NoSQL de Microsoft que, a diferencia de las bases de datos relacionales tradicionales, no tiene esquema. El acceso a los datos es más rápido y su coste muy inferior al del SQL tradicional para volúmenes de datos similares [19].

Un servicio de tabla (ver Fig. 20) está compuesto por:

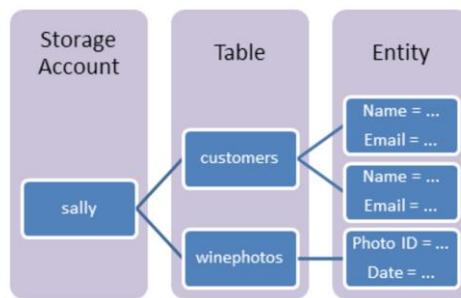


Fig. 20 Componentes de un servicio tabla [19]

- Formato URL: para acceder a una servicio tabla de una cuenta de almacenamiento se utiliza el siguiente formato:

http://<storage account>.table.core.windows.net/<table>

- Cuenta de almacenamiento: necesaria para acceder a la tabla.
- Tabla: es una colección de entidades dentro de una cuenta. Las entidades no están obligadas a seguir ningún esquema, lo que significa que una única tabla puede contener entidades con diferentes conjuntos de propiedades.
- Entidad: es un conjunto de propiedades, similar a una fila de una base de datos. Puede tener un tamaño máximo de 1MB.
- Propiedades: es un par nombre-valor. Cada entidad puede incluir hasta 252 propiedades para almacenar datos. Cada entidad tiene 3 propiedades de sistema que especifican una *partition key*, una *row key* y un *timestamp*. Las entidades con el mismo *partition key* pueden ser consultadas más rápidamente e insertadas/actualizadas en operaciones atómicas. El *row key* de una entidad es un identificador único dentro de un *partition key*.

2.3.3.1.3 Colas (Queues)

El servicio de almacenamiento de colas es un servicio para almacenar un gran número de mensajes que pueden ser accedidos desde cualquier punto del mundo mediante llamadas autenticadas utilizando HTTP o HTTPS. Un único mensaje de cola puede tener un tamaño de hasta 64 KB. La cola puede tener millones de mensajes hasta completar el límite de la capacidad de la cuenta de almacenamiento [19].

Un servicio de cola (ver Fig. 21) está compuesto por:

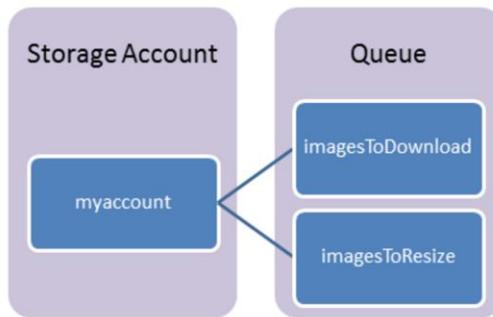


Fig. 21 Componentes de un servicio cola [19]

- Formato URL: para acceder a una servicio cola en una cuenta de almacenamiento se utiliza el siguiente formato:

`http://[account].queue.core.windows.net/<queue>`

- Cola: es un conjunto de mensajes.
- Mensaje: un mensaje en cualquier formato. Puede tener un tamaño máximo de 64 KB.

2.3.3.2 Modo de Replicación de Datos

Los datos están siempre replicándose para asegurar persistencia y alta disponibilidad. Como mínimo están almacenados por triplicado. En cualquier caso, cuando se crea una cuenta de almacenamiento, se debe seleccionar una de las siguientes opciones de replicación:

- **Almacenamiento con redundancia local (LRS):** mantiene tres copias de los datos. LRS se replica tres veces dentro de una única instalación de una sola región. LRS protege los datos frente a errores comunes del *hardware*, pero no frente a errores de una única instalación.
- **Almacenamiento con redundancia de zona (ZRS):** mantiene tres copias de los datos. ZRS se replica tres veces entre dos o tres instalaciones, ya sea dentro de una sola región o entre dos regiones, proporcionando mayor persistencia que LRS. ZRS garantiza la persistencia de sus datos dentro de una sola región.

- **Almacenamiento con redundancia geográfica (GRS):** mantiene seis copias de los datos. Con GRS, los datos se replican tres veces dentro de la región primaria, y se replican también tres veces en una región secundaria a cientos de kilómetros de distancia de la región primaria, proporcionando el nivel más alto de persistencia. En caso de que se produzca un error en la región primaria, se comutará a la región secundaria. GRS garantiza la durabilidad de los datos en dos regiones independientes.
- **Almacenamiento con redundancia geográfica con acceso de lectura (RA-GRS):** es igual que GRS pero proporciona acceso de lectura a dichos datos en la ubicación secundaria desde la ubicación principal o la secundaria.

2.3.3.3 Tarifas de la Cuentas de Almacenamiento

Las cuentas de almacenamiento conllevan una serie de costes tanto de almacenamiento como de acceso. Estos costes dependen principalmente del tipo de almacenamiento, es decir, *blob*, tabla, cola, etc. y de la ubicación del CPD donde reside la información. En función del tipo de almacenamiento se tienen en cuenta otros factores, como vamos a ver a continuación.

Las tarifas de almacenamiento y acceso aplicadas a los *blobs* dependen del tipo de redundancia (LRS/GRS/RA-GRS) y del nivel de acceso (estático/dinámico) que se definieron cuando se creó la cuenta de almacenamiento. En la Fig. 22 se pueden consultar las tarifas de almacenamiento para *blobs* en Europa Occidental que, además, depende de la cantidad de datos almacenada. En y Fig. 23 se pueden consultar las tarifas de acceso para blobs en Europa Occidental que, además, depende del tipo de acceso.

Blobs en bloques

El almacenamiento de blobs en bloques se usa para streaming y almacenamiento de documentos, videos, imágenes, copias de seguridad y otros datos de texto no estructurado o binarios. A continuación se indican los precios para cuentas de almacenamiento de blobs dedicadas con niveles de acceso frecuente y esporádico. Los blobs en anexos se miden también como blobs en bloques

[Más información ▶](#)

Precios de almacenamiento

Estos son los costos de almacenar datos en blobs en bloques. Los precios que se indican a continuación son los cargos mensuales por GB de datos almacenado. Estos precios pueden variar según el nivel de acceso del almacenamiento de blobs en bloques (frecuente o esporádico), la opción de redundancia que elija y la cantidad de datos que almacene.

	LRS		GRS		RA-GRS	
	ACCESO ESPORÁDICO	ACCESO FRECUENTE	ACCESO ESPORÁDICO	ACCESO FRECUENTE	ACCESO ESPORÁDICO	ACCESO FRECUENTE
Primeros 100 TB/mes	€0,0126	€0,0202	€0,0253	€0,0405	€0,0316	€0,0514
Siguientes 900 TB / mes	€0,0126	€0,0195	€0,0253	€0,0391	€0,0316	€0,0496
Siguientes 4.000 TB / mes	€0,0126	€0,0188	€0,0253	€0,0376	€0,0316	€0,0478
Más de 5.000 TB/mes	Ponerse en contacto con nosotros					

Fig. 22 Precio almacenamiento en blobs en Europa Occidental [20]

Precios de acceso

Estos son los costos de las operaciones HTTP en datos de blobs en bloques, junto con el costo de recuperar datos de blobs en bloques o escribir datos en blobs en bloques. Estos precios pueden variar según el nivel de acceso del almacenamiento de blobs en bloques (frecuente o esporádico) y la opción de redundancia que elija.

	LRS		GRS Y RA-GRS	
	ACCESO ESPORÁDICO	ACCESO FRECUENTE	ACCESO ESPORÁDICO	ACCESO FRECUENTE
Operaciones PUT Blob/Block, List, Create Container (en decenas de millar)	€0,0843	€0,0455	€0,1687	€0,0911
Todas las demás operaciones, excepto Delete, que es gratis (en decenas de millar)	€0,0084	€0,0034	€0,0084	€0,0034
Recuperación de datos (por GB)	€0,0084	Gratis	€0,0084	Gratis
Escrutina de datos (por GB)	€0,0021	Gratis	€0,0042	Gratis
Transferencia de datos de replicación geográfica (por GB)	N/D	N/D	€0,0169	€0,0169

Fig. 23 Precio acceso a los blobs en Europa Occidental [20]

Las tarifas de almacenamiento y acceso aplicadas a las tablas dependen del tipo de redundancia (LRS/GRS/RA-GRS) que se definió cuando se creó la cuenta de almacenamiento. En la Fig. 24 se pueden consultar las tarifas de almacenamiento para tablas en Europa Occidental que, además, depende de la cantidad de datos almacenada. También en la Fig. 24 se pueden consultar las tarifas de acceso para tablas en Europa Occidental que, además, depende del número de accesos efectuados.

Tablas

Las tablas ofrecen almacenamiento NoSQL para datos no estructurados y semiestructurados, ideal para aplicaciones web, libretas de direcciones y otros datos de usuario.

[Más información ▶](#)

Precios de almacenamiento

CAPACIDAD DE ALMACENAMIENTO	LRS	GRS	RA-GRS
Primer TB / mes	€0,059 por GB	€0,0801 por GB	€0,1012 por GB
Siguientes 49 TB (1 hasta 50 TB)/ mes	€0,0548 por GB	€0,0675 por GB	€0,0843 por GB
Siguientes 450 TB (50 hasta 500 TB)/ mes	€0,0506 por GB	€0,059 por GB	€0,0759 por GB
Siguientes 500 TB (500 hasta 1.000 TB)/ mes	€0,0464 por GB	€0,0548 por GB	€0,0675 por GB
Siguientes 4.000 TB (1.000 hasta 5.000 TB)/ mes	€0,0379 por GB	€0,0506 por GB	€0,0632 por GB
Más de 5.000 TB/mes	Ponerse en contacto con nosotros	Ponerse en contacto con nosotros	Ponerse en contacto con nosotros

Precios de acceso

Cobramos €0,003 por 100.000 transacciones para tablas. Las transacciones incluyen las operaciones de lectura y escritura en el almacenamiento.

Fig. 24 Precio acceso y almacenamiento en tablas en Europa Occidental [20]

2.3.4 Plan del Servicio de Aplicaciones (*Service Plan App*)

Un plan del servicio de aplicaciones representa un conjunto de características y capacidades que se pueden compartir entre múltiples aplicaciones del servicio de aplicaciones de Microsoft Azure, incluidas las aplicaciones *Web*, aplicaciones móviles, aplicaciones lógicas o aplicaciones de API. Las aplicaciones de una misma suscripción y ubicación geográfica pueden compartir un plan. Una aplicación solo se puede asociar a un único plan en un momento determinado.

Tanto las aplicaciones como los planes se incluyen en un grupo de recursos. Un grupo de recursos sirve como límite del ciclo de vida de cada uno de los recursos que contiene. Los grupos de recursos permiten administrar todos los componentes de una aplicación conjuntamente. Se pueden tener varios planes de servicio de aplicaciones en un solo grupo de recursos, lo que es útil para tener entornos diferentes de desarrollo, producción, testing, etc. o regiones diferentes en un mismo grupo de recursos.

Para crear un plan del servicio de aplicaciones desde la plataforma necesitamos seleccionar un nombre para el plan de servicio de aplicaciones, una suscripción, un grupo de recursos, una ubicación (Oeste de Europa, Norte de Estados Unidos, etc.) del CPD donde se desea este alojada la información y un plan de tarifa (ver Fig. 25).

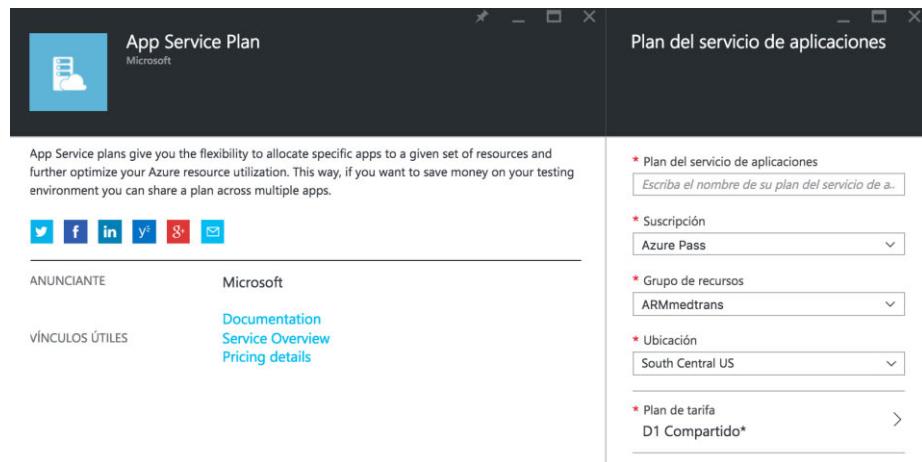


Fig. 25 Crear plan de servicio de aplicaciones desde la plataforma

Una vez creado el plan del servicio de aplicaciones, se puede administrar desde la plataforma Microsoft Azure (ver Fig. 26).

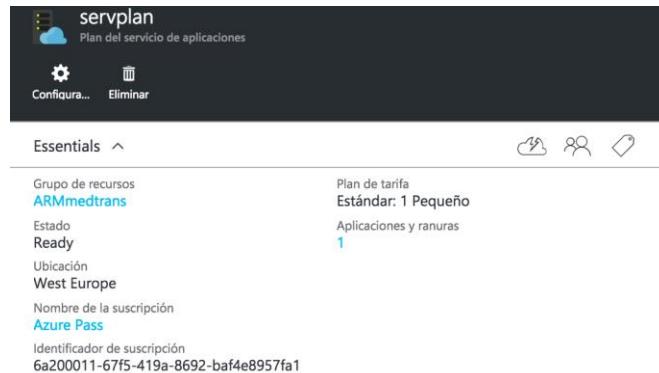


Fig. 26 Administración de plan de servicio de aplicaciones en Azure

2.3.4.1 Planes de Tarifa

Los planes de servicio de aplicaciones admiten 5 planes de tarifa (Gratis, Compartido, Básico, Estándar y Premium), donde cada uno de ellos tiene sus propias funcionalidades y capacidad. Como es obvio, en función del plan el precio varía y cuesta más cuantas más características ofrece (ver Fig. 27).

F1 Free	D1 Shared*	B1 Basic	B2 Basic	B3 Basic
- Shared infrastructure	- Shared infrastructure	1 Core	2 Core	4 Core
1 GB Storage	1 GB Storage Custom domains	1.75 GB RAM 50 GB Storage 5 SNI, 1 IP Custom domains Up to 10 instances Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability	3.5 GB RAM 10 GB Storage 5 SNI, 1 IP Custom domains / SSL Up to 10 instances Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability	7 GB RAM 10 GB Storage Custom domains Up to 3 instances Manual scale
0.00 EUR/MONTH (ESTIMATED)	8.16 EUR/MONTH (ESTIMATED, *PER APP)	47.06 EUR/MONTH (ESTIMATED)	94.11 EUR/MONTH (ESTIMATED)	188.22 EUR/MONTH (ESTIMATED)

S1 Standard	S2 Standard	S3 Standard	P1 Premium	P2 Premium	P3 Premium
1 Core	2 Core	4 Core	1 Core	2 Core	4 Core
1.75 GB RAM 50 GB Storage 5 SNI, 1 IP Custom domains Up to 10 instances Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability	3.5 GB RAM 50 GB Storage 5 SNI, 1 IP Custom domains / SSL Up to 10 instances Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability	7 GB RAM 50 GB Storage 5 SNI, 1 IP Custom domains / SSL Up to 10 instances Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability	1.75 GB RAM BizTalk Services 250 GB Storage Up to 20 instances * Subject to availability 20 slots Web app staging 5 slots daily Backup Traffic Manager Geo availability	3.5 GB RAM BizTalk Services 250 GB Storage Up to 20 instances * Subject to availability 20 slots Web app staging 50 times daily Backup Traffic Manager Geo availability	7 GB RAM BizTalk Services 250 GB Storage Up to 20 instances * Subject to availability 20 slots Web app staging 50 times daily Backup Traffic Manager Geo availability
62.74 EUR/MONTH (ESTIMATED)	125.48 EUR/MONTH (ESTIMATED)	250.97 EUR/MONTH (ESTIMATED)	207.05 EUR/MONTH (ESTIMATED)	414.09 EUR/MONTH (ESTIMATED)	828.19 EUR/MONTH (ESTIMATED)

Fig. 27 Planes de tarifa del servicio de aplicaciones Microsoft Azure

2.3.4.2 Escalar verticalmente (*scale up/scale down*)

Una operación de escalado vertical es equivalente a mover el sitio *Web* a un servidor físico más potente. Se debe considerar hacer un escalado vertical cuando el sitio está alcanzando alguna cuota, lo cual indica que se está quedando pequeño lo que teníamos.

El escalado vertical se llevar a cabo seleccionando manualmente el nuevo plan de tarifa deseado de los disponibles (ver Fig. 28).

Plan	Características	Costo (EUR/MES)
P1 Premium	1 Núcleo, 1.75 GB de RAM, Servicios de BizTalk, 250 GB Almacenamiento, Hasta 20 instancias * Sujeto a disponibilidad, 20 espacios Almacenamiento provis., 50 veces al día Copia de seguridad, Administrador de trá., Disponibilidad de geóreas	207.05
P2 Premium	2 Núcleo, 3.5 GB de RAM, Servicios de BizTalk, 250 GB Almacenamiento, Hasta 20 instancias * Sujeto a disponibilidad, 20 espacios Almacenamiento provis., 50 veces al día Copia de seguridad, Administrador de trá., Disponibilidad de geóreas	414.09
P3 Premium	4 Núcleo, 7 GB de RAM, Servicios de BizTalk, 250 GB Almacenamiento, Hasta 20 instancias * Sujeto a disponibilidad, 20 espacios Almacenamiento provis., 50 veces al día Copia de seguridad, Administrador de trá., Disponibilidad de geóreas	828.19
S1 Estándar	1 Núcleo, 1.75 GB de RAM, 50 GB Almacenamiento, 5 SNI, 1 IP, Dominios personalizados, Hasta 10 instancias Escalado automático, Diario Copia de seguridad, 5 espacios Almacenamiento provis., Administrador de trá., Disponibilidad de geóreas	62.74
S2 Estándar	2 Núcleo, 3.5 GB de RAM, 50 GB Almacenamiento, 5 SNI, 1 IP, Dominios personalizados, Hasta 10 instancias Escalado automático, Diario Copia de seguridad, 5 espacios Almacenamiento provis., Administrador de trá., Disponibilidad de geóreas	125.48
S3 Estándar	4 Núcleo, 7 GB de RAM, 50 GB Almacenamiento, 5 SNI, 1 IP, Dominios personalizados, Hasta 10 instancias Escalado automático, Diario Copia de seguridad, 5 espacios Almacenamiento provis., Administrador de trá., Disponibilidad de geóreas	250.97

Fig. 28 Escalar verticalmente el plan de servicio de aplicaciones

2.3.4.3 Escalar Horizontalmente (*scale in/scale out*)

Un operación de escalado horizontal es equivalente a crear múltiples copias del sitio *Web* y añadir un balanceador de carga para distribuir la demanda entre ellos. De la gestión y configuración de dicho balanceador de carga se encarga Microsoft Azure.

El escalado horizontal se puede llevar a cabo modificando manualmente el número de instancias de su servicio o bien establecer parámetros para que esta cantidad se escale automáticamente en función de la demanda (ver Fig. 29).

En cualquier caso, antes de escalar el número de instancias (escalado horizontal) hay que estudiar si seleccionando otro plan de tarifa con más núcleos y memoria (escalado vertical) es posible encontrar un mejor rendimiento para el mismo número de instancias y de forma más económica.

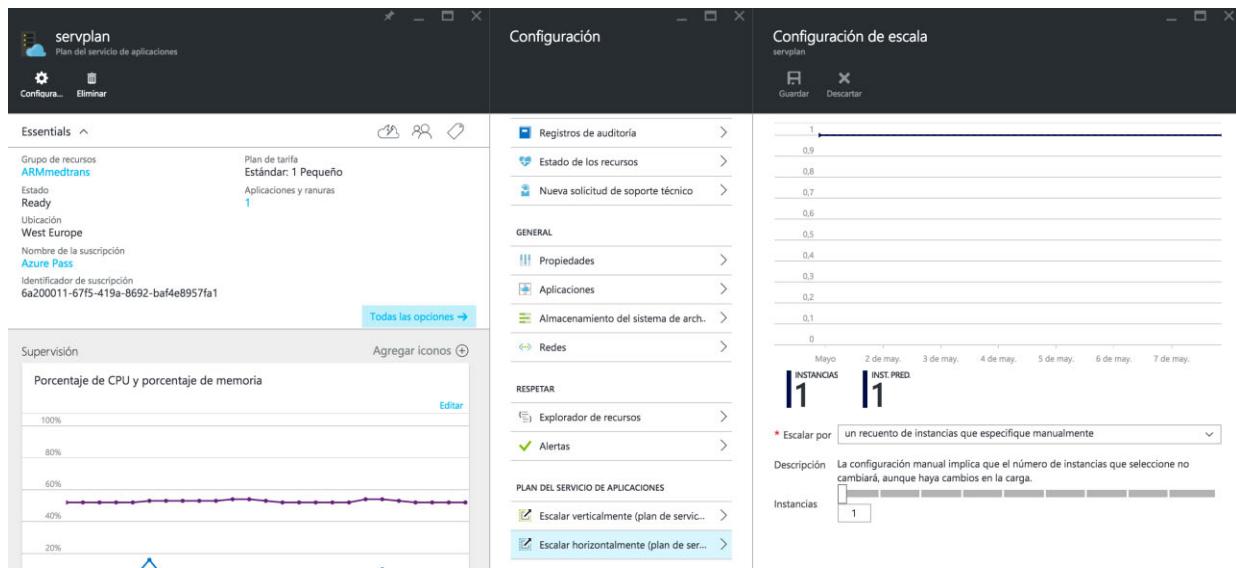


Fig. 29 Escalar horizontalmente el plan del servicio de aplicaciones

2.3.5 Servicio de Aplicaciones: API app

Las aplicaciones API son uno de los cuatro tipos de aplicaciones que ofrece el servicio de aplicaciones de Microsoft Azure. El servicio de aplicaciones es una plataforma de administración que está diseñada para desarrolladores y que integra todo lo que se necesita para crear y gestionar fácilmente estos tipos de aplicaciones. Las aplicaciones API permiten crear y desplegar servicios RESTful (ver Sección 2.2.1). Para crear un API App desde la plataforma necesitamos seleccionar un nombre para la aplicación, una suscripción, un grupo de recursos y un plan del servicio de aplicaciones (ver Fig. 30).

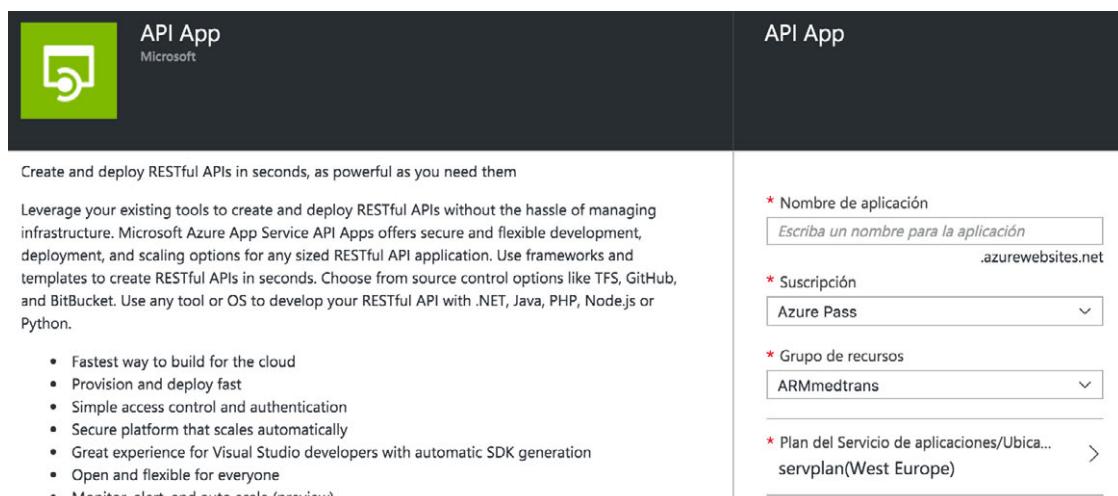


Fig. 30 Crear aplicación API desde la plataforma

Una vez creada la aplicación API, se puede administrar desde el servicio de aplicación de API de la plataforma Microsoft Azure (ver Fig. 31).

The screenshot shows the Azure portal interface for managing the 'apimedtrans' API application. At the top, there's a toolbar with icons for configuration, tools, examine, stop, swap, restart, delete, get profile, and restore profile. Below the toolbar, the application name 'apimedtrans' and its status 'APLICACIÓN DE API' are displayed. A 'Configuración' button is highlighted. The main content area is titled 'Essentials' and contains the following information:

Configuración	Detalles
Grupo de recursos	URL
ARMmedtrans	http://apimedtrans.azurewebsites.net
Estado	Plan del Servicio de aplicaciones/plan de tarifa
Running	servplan (Estándar: 1 Pequeño)
Ubicación	FTP/Nombre de usuario de implementación
West Europe	apimedtrans\amrodriguez
Nombre de la suscripción	Nombre de host de FTP
Azure Pass	ftp://waws-prod-am2-065.ftp.azurewebsite...
Identificador de suscripción	Nombre de host de FTPS
6a200011-67f5-419a-8692-baf4e8957fa1	https://waws-prod-am2-065.ftp.azurewebsit...

Fig. 31 Administración de aplicación de API en Azure

A continuación se va a describir las opciones de administración más interesantes a las que se puede acceder desde la interfaz de la Fig. 31: configuración, herramientas e intercambio.

2.3.5.1 Configuración

Las opciones de configuración son diversas, de entre las más interesantes que se han encontrado para elaborar este PFG y que se detallan en las siguientes subsecciones:

- Configuración de la aplicación (ver Sección 2.3.5.1.1)
- Credenciales de implementación o ensayo (ver Sección 2.3.5.1.2)
- Espacios de implementación (ver Sección 2.3.5.1.3)
- Cambiar el plan del servicio de aplicaciones (ver toda esta Sección 2.3.4).
- Escalar verticalmente (ver Sección 2.3.4.2).
- Escalar horizontalmente (ver Sección 2.3.4.3)
- Inicio rápido (ver Sección 2.3.5.1.2)

2.3.5.1.1 Configuración de la aplicación

La configuración de la aplicación consiste en determinar las características principales de ésta (ver Fig. 32), tales como:

- Versión de .NET framework: solo si esta desactivado el contenedor *Web*.
- Versión de PHP: solo si esta desactivado el contenedor *Web*.
- Tipo y versión del contenedor *Web*: Tomcat, Jetty o desactivado.
- Plataforma: 32/64 bits.
- Cadenas de conexión: variables con las que podemos comunicar dinámicamente a la aplicación nuevas cadenas de conexión si es necesario. Nos evitamos de esta forma tener que modificar el código y volver a desplegar (si las tuviésemos *hardcodeadas*).

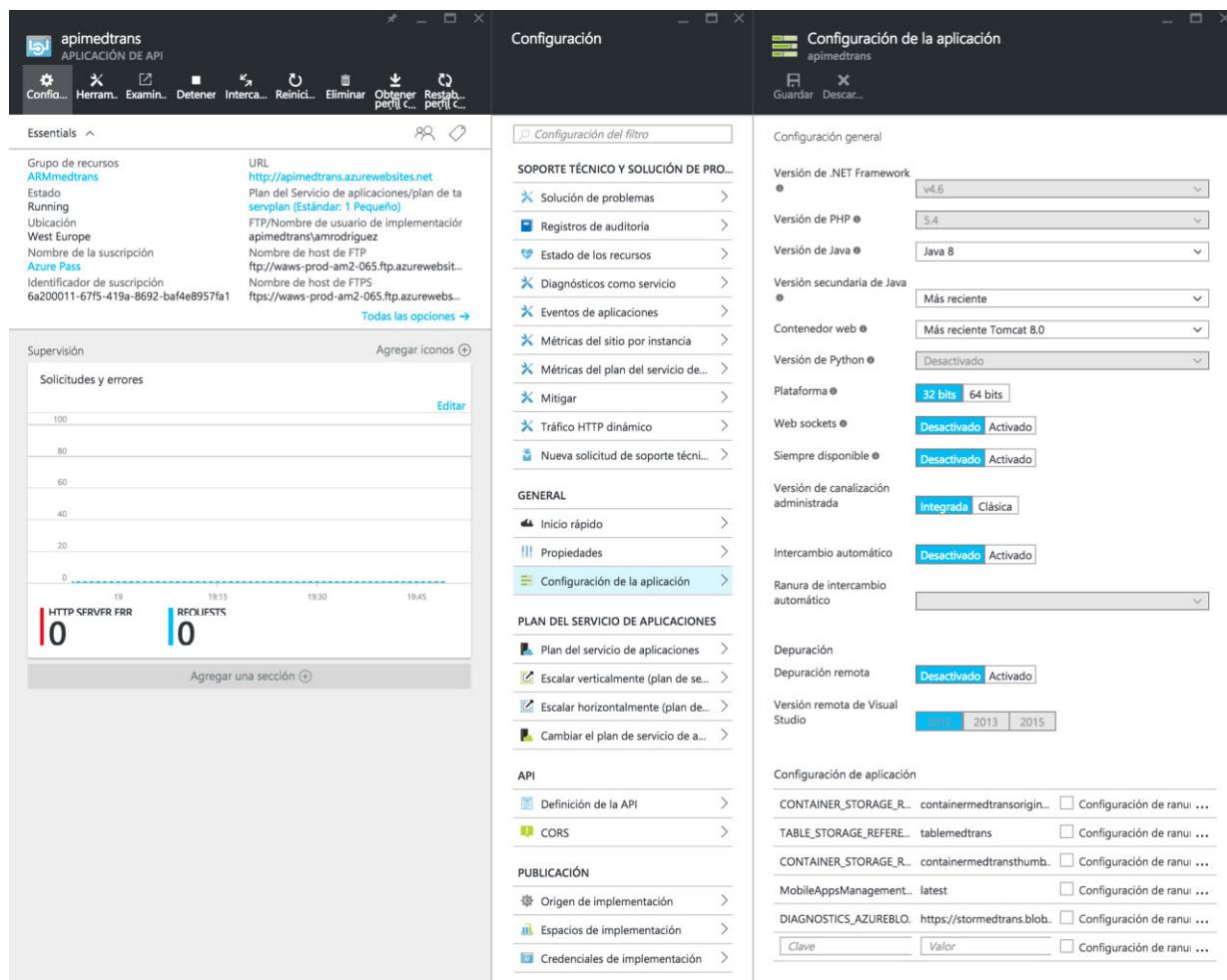


Fig. 32 Configuración de la aplicación API

2.3.5.1.2 Credenciales de implementación

Los credenciales (nombre de usuario y contraseña) son los mismos para todas las aplicaciones de todas las suscripciones asociadas a la cuenta de Microsoft Azure. Para poder acceder vía FTP a la URL donde está alojada una aplicación hay que configurar los credenciales de acceso (ver Fig. 33).

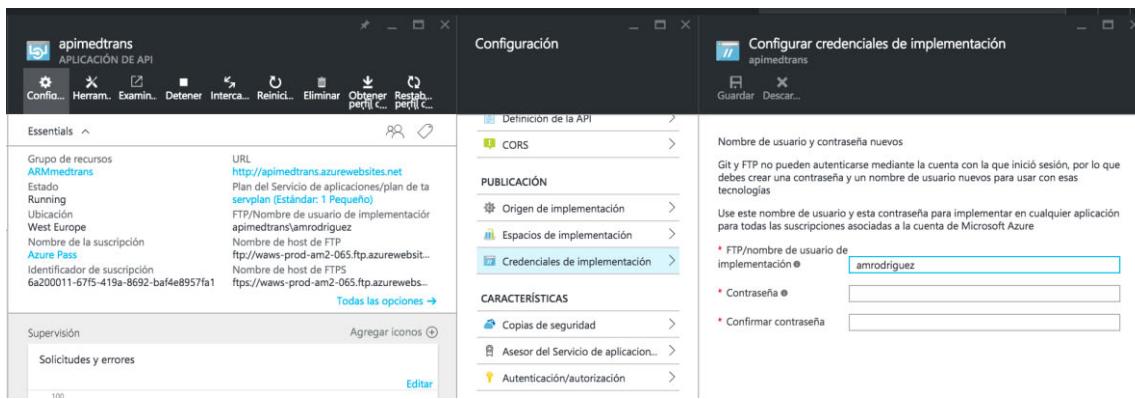


Fig. 33 Configuración API: credenciales de implementación

2.3.5.1.3 Espacios de implementación o ensayo (*staging slot*)

Tanto en el plan del servicio de aplicaciones Estándar, como en el Premium, se pueden configurar espacios de implementación o ensayo (ver Fig. 34) que habilitan un mecanismo de paso a producción con menos riesgos y más rápido ya que la aplicación se despliega primero a este espacio y una vez que ha arrancado y lo ha hecho correctamente, simplemente se intercambia su espacio con el espacio de producción

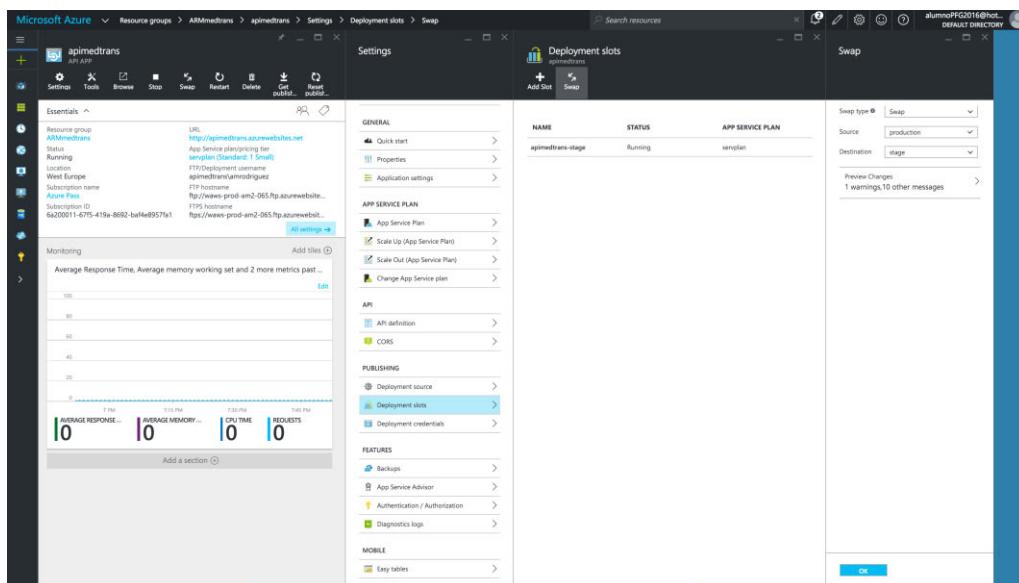


Fig. 34 Configuración API: espacio de implementación (*staying slot*)

2.3.5.1.4 Inicio rápido

La opción de inicio rápido permite descargar un prototipo de un API de ejemplo en el lenguaje elegido (Java, NodeJS, ASP.Net). Esta opción ayuda a disponer de un prototipo de aplicación que ayuda en el arranque del desarrollo.

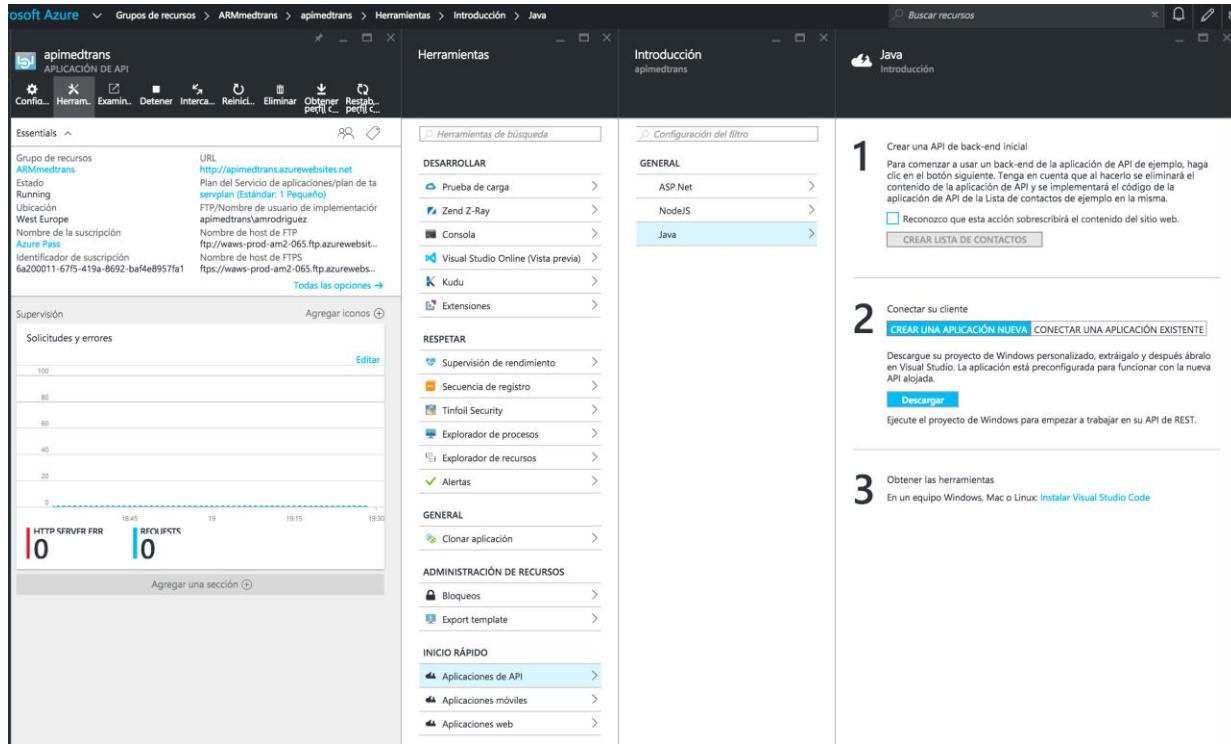


Fig. 35 Configuración API: creación de prototipo API en Java

2.3.5.2 Herramientas

Las herramientas que ofrece Azure son diversas, de entre las más interesantes que se han encontrado para elaborar este PFG y que se detallan en las siguientes subsecciones:

- Explorador de recursos
- Kudu

2.3.5.2.1 Explorador de recursos (*resource explorer*)

El explorador de recursos es una excelente herramienta para comprender cómo se estructuran los recursos y ver las propiedades asignadas a cada recurso. Permite aprender acerca de las operaciones que están disponibles para un tipo de recurso tanto en forma de API REST como de cmdlets de PowerShell, permitiendo ejecutar dichos comandos a través de su interfaz [21]. Se puede acceder a ella a través del enlace "<https://resources.azure>" o bien

pinchando la opción Explorador de Recursos dentro de la Configuración del Plan del servicio de aplicaciones.

En la pestaña *Data* se puede pinchar “*Edit*” para editar el contenido JSON y enviar una petición PUT y cambiarlo (Fig. 36).

The screenshot shows the Azure Resource Explorer interface. On the left, there's a navigation tree with categories like providers, subscriptions, and resource groups. The 'servplan' node under 'serverfarms' is selected. On the right, there's a main content area with tabs for Data (GET, PUT), Actions (POST, DELETE), Create, Documentation, and PowerShell. The Data tab is active, displaying a large block of JSON code representing the service plan. The JSON includes fields such as 'id', 'name', 'type', 'location', 'tags', 'properties', 'serverFarmId', 'workerSizeInMB', 'workerSizeId', 'workerTierName', 'numberOfWorkers', 'currentWorkerSize', 'currentWorkerSizeId', 'currentNumberOfWorkers', 'status', 'adminRuntimeSiteName', 'computeMode', 'hostingEnvironment', 'geoRegion', 'perSiteScaling', 'numberOfSites', 'hostingEnvironmentId', 'tags', 'kind', and 'resourceGroup'. Below the JSON, there's a section for 'PowerShell equivalent script' containing cmdlets like Get-AzureRmResource, Set-AzureRmResource, Remove-AzureRmResource, and Invoke-AzureRmResourceAction.

```

1 - {
2   "id": "/subscriptions/6a200011-67f5-419a-8692-baf4e8957fa1/resourceGroups/ARMmedtrans/providers/Microsoft.Web/serverfarms/servplan",
3   "name": "servplan",
4   "type": "Microsoft.Web/serverfarms",
5   "location": "West Europe",
6   "tags": null,
7   "properties": {
8     "serverFarmId": 0,
9     "workerSizeInMB": 0,
10    "workerSizeId": 0,
11    "workerTierName": null,
12    "numberOfWorkers": 0,
13    "currentWorkerSize": 0,
14    "currentWorkerSizeId": 0,
15    "currentNumberOfWorkers": 0,
16    "status": 0,
17    "adminRuntimeSiteName": "ARMmedtrans-WestEuropewebspace",
18    "adminSiteName": "6a200011-67f5-419a-8692-baf4e8957fa1",
19    "hostingEnvironment": null,
20    "hostingEnvironmentProfile": null,
21    "maximumNumberOfWorkers": 1,
22    "planName": "VirtualDedicatedPlan",
23    "adminRuntimeSiteName": null,
24    "computeMode": 0,
25    "hostingEnvironment": null,
26    "geoRegion": "West Europe",
27    "perSiteScaling": false,
28    "numberOfSites": 1,
29    "hostingEnvironmentId": null,
30    "tags": null,
31    "kind": null,
32    "resourceGroup": "ARMmedtrans"
33  },
34  "value": [
35    {
36      "name": {
37        "tier": "D1",
38        "size": "Shared",
39        "family": "D",
40        "capacity": 0
41      }
42    }
43  ]
}

```

Fig. 36 Herramientas API: explorador de recursos Azure en JSON

En la pestaña PowerShell se muestran los cmdlets que hay que utilizar para interactuar con el recurso que se está explorando (ver Fig. 37).

This screenshot is similar to Fig. 36 but focuses on the 'PowerShell' tab. It shows the PowerShell equivalent script for the selected 'servplan' resource. The script includes cmdlets for getting, setting, and deleting the resource, as well as actions like restarting sites. It also includes examples of removing and invoking specific actions.

```

# PowerShell equivalent script

# GET servplan
Get-AzureRmResource -ResourceGroupName ARMmedtrans -ResourceType Microsoft.Web/serverfarms -ResourceName servplan -ApiVersion 2015-08-01

# SET servplan
$PropertiesObject = @{
  #Property = value;
}
Set-AzureRmResource -PropertyObject $PropertiesObject -ResourceGroupName ARMmedtrans -ResourceType Microsoft.Web/serverfarms -ResourceName servplan -ApiVersion 2015-08-01
-Force

# Actions available on that object
# Delete servplan
Remove-AzureRmResource -ResourceGroupName ARMmedtrans -ResourceType Microsoft.Web/serverfarms -ResourceName servplan -ApiVersion 2015-08-01
# Action restartSites
Invoke-AzureRmResourceAction -ResourceGroupName ARMmedtrans -ResourceType Microsoft.Web/serverfarms -ResourceName servplan -Action restartSites -ApiVersion 2015-08-01
-Force

```

Fig. 37 Herramientas API: explorador de recursos de Azure en PowerShell

2.3.5.2.2 Kudu

Kudu es una herramienta de gran utilidad para depurar las aplicaciones *Web* desplegadas en Microsoft Azure. Al interfaz de Kudu, se accede mediante un navegador y la URL de la aplicación *Web* a la cual se le ha incluido la subcadena "scm". Por ejemplo, "<https://apimedtrans.scm.azurewebsites.net>"

Entre otras cosas, se tiene acceso a un explorador de procesos y las consolas de depuración CMD (ver Fig. 37) o PowerShell (ver Fig. 39). Se puede utilizar como FTP simplemente arrastrando los ficheros.

The screenshot shows the Kudu interface for a web application. At the top, there's a navigation bar with tabs for 'Kudu', 'Environment', 'Debug console' (which is currently selected), 'Process explorer', 'Tools', and 'Site extensions'. A dropdown menu for 'Debug console' is open, showing options for 'CMD' and 'PowerShell'. Below the navigation is a file browser interface with a sidebar showing a folder structure ('/ + 4 items') and a main table listing files:

	Name	Modified	Size
↓ ↗	data	4/16/2016, 7:19:44 AM	
↓ ↗	LogFiles	5/7/2016, 7:28:13 AM	
↓ ↗	site	4/21/2016, 5:47:29 AM	
↓ ↗	SiteExtensions	4/21/2016, 5:47:30 AM	

Below the file browser is a large terminal window titled 'Kudu Remote Execution Console'. It displays the Windows command prompt environment:

```
Kudu Remote Execution Console
Type 'exit' then hit 'enter' to get a new CMD process.
Type 'cls' to clear the console

Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

D:\home>
```

A link 'Use old console' is visible in the top right corner of the terminal window.

Fig. 38 Herramientas API: interfaz Kudu de depuración de aplicaciones web CMD

This screenshot shows the same Kudu interface as Fig. 38, but with the 'PowerShell' tab selected in the 'Debug console' dropdown. The rest of the interface is identical to Fig. 38, including the file browser and the terminal window displaying the Windows command prompt.

Fig. 39 Herramientas API: interfaz Kudu de depuración de aplicaciones Web PowerShell

2.3.5.3 Intercambio (*swap*)

Tanto en el plan del servicio de aplicaciones Estándar como en el Premium antes de desplegar la aplicación en producción se puede desplegar en un espacio de implementación o ensayo (*staging slot*) [22]. Empleando este método, el despliegue de la aplicación en producción consiste en hacer intercambio entre el espacio de producción y el espacio de implementación (o ensayo). Cuenta con la ventaja que mientras que el despliegue de una aplicación puede llevar minutos y puede haber errores que la dejen inactiva, el intercambio entre dichos espacios es nulo (el redireccionamiento del tráfico es impecable y no se anulan las solicitudes como consecuencia del intercambio) y nos garantizamos que no hay errores.

Después de un intercambio, el espacio de ensayo tiene la aplicación anterior y el espacio de producción tiene la aplicación nueva. Si el resultado no es el esperado en el espacio de producción, se puede realizar el mismo intercambio inmediatamente para volver a la situación inicial y tener el "último sitio en buen estado" (ver Fig. 40).

The screenshot shows the Azure portal interface for managing an API application named 'apimedtrans'. The top navigation bar includes icons for configuration, monitoring, examination, stopping, restarting, deleting, and swapping. The 'Intercambiar' (Swap) button is highlighted. The left sidebar displays essential information about the app, such as its group, state, location, subscription, and identifier. The main content area is titled 'Intercambiar' and contains a form to swap between 'production' and 'stage' environments. It shows the current selection of 'Intercambiar' for the type of swap, 'production' as the origin, and 'stage' as the destination. A preview link indicates there are 1 alert and 10 messages related to the swap operation.

Fig. 40 Intercambio de aplicación API entre producción y slot

2.3.6 Servicio de automatización

El servicio de automatización de Microsoft Azure ofrece una forma de automatizar tareas manuales, propensas a errores, de larga duración o que se repiten con frecuencia. Un ejemplo representativo es un proceso *batch* diario que ejecuta un script escrito en Azure PowerShell que carga ficheros desde un servidor a un blob en un servicio de almacenamiento (ver Sección 2.3.3.1.1).

2.3.6.1 Cuenta de Automatización (Automation Account)

Para utilizar el servicio de automatización es necesario crear una cuenta de automatización. Para crear una cuenta de automatización desde la plataforma se necesita seleccionar un nombre para la cuenta de automatización, una suscripción, un grupo de recursos y una ubicación (Oeste de Europa, Norte de Estados Unidos, etc.) del centro de datos donde se desea este alojada la información (ver Fig. 41).

The screenshot shows two windows side-by-side. On the left is the 'Automatización Microsoft' blade, which includes sections for creating a new automation account, explaining its benefits, and listing integrations with other Azure services. On the right is the 'Agregar cuenta de Automatización...' (Add Automation Account) blade, where the user is prompted to enter the account name, select a subscription, choose a resource group, specify a location, and decide whether to create a new Azure execution account. A detailed information box provides further details about the execution account feature.

Automatización Microsoft

Crear una cuenta de automatización

Una cuenta de automatización es un contenedor para los recursos de Automatización de Azure. Le permite separar entornos u organizar mejor los flujos de trabajo y los recursos de Automatización.

Automatización de procesos que simplifica la administración de la nube

Automatización de Azure le permite automatizar la creación, la implementación, la supervisión y el mantenimiento de recursos en su entorno de Azure y en todos sus sistemas externos. Azure usa un motor de ejecución de flujo de trabajo altamente escalable y fiable para simplificar la administración de la nube. Podrá coordinar tareas repetitivas y laboriosas en su sistema de Azure, así como en sistemas de terceros.

Integración con los sistemas de los que depende

Con Automatización, podrá conectarse a cualquier sistema que exponga una API mediante protocolos típicos de Internet. Automatización de Azure incluye la integración con muchos servicios de Azure, entre otros:

- Sitios web (administración)
- Servicios en la nube (administración)
- Máquinas virtuales (administración y compatibilidad con WinRM)
- Almacenamiento (administración)
- SQL Server (administración y compatibilidad con SQL)

¿Necesita que sus flujos de trabajo se integren en otro servicio? Amplíe Automatización de Azure a las soluciones de terceros simplemente importando un módulo PowerShell ya existente o escribiendo el suyo en C# o Windows PowerShell.

ANUNCIANTE Microsoft

VÍNCULOS ÚTILES

[Documentación de Automatización de Azure](#)
[Buscar Runbooks en el Centro de scripts](#)
[Detalles de precios](#)

Agregar cuenta de Automatización...

* Nombre

* Suscripción

* Grupo de recursos

* Ubicación

* Crear cuenta de ejecución de Azure Sí No

Información

La característica de Cuenta de ejecución creará un nuevo usuario de entidad de servicio en Azure Active Directory y asignará un rol Colaborador predeterminado a este usuario en el nivel de suscripción. [Obtenga más información sobre las entidades de servicio y los cambios de asignación de roles.](#)

Fig. 41 Crear cuenta de automatización desde plataforma

Una vez creada la cuenta de automatización, se puede administrar desde la plataforma Microsoft Azure (ver Fig. 42).

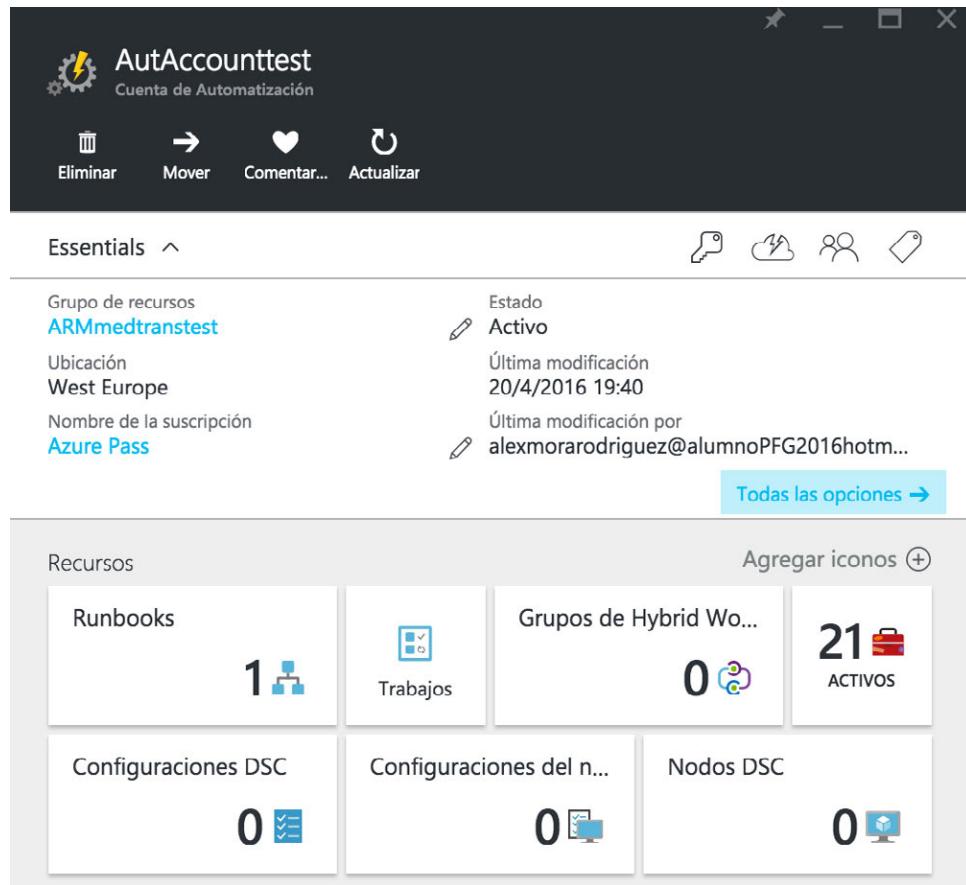


Fig. 42 Administración de cuenta de automatización en Azure

El plan de tarifa por defecto es el gratuito (500 minutos gratis al mes). Posteriormente, si es necesario, se puede cambiar el plan de tarifa al Básico (ver Fig. 43) para obtener más minutos al precio de 0.002 Euros/Minuto de trabajo.

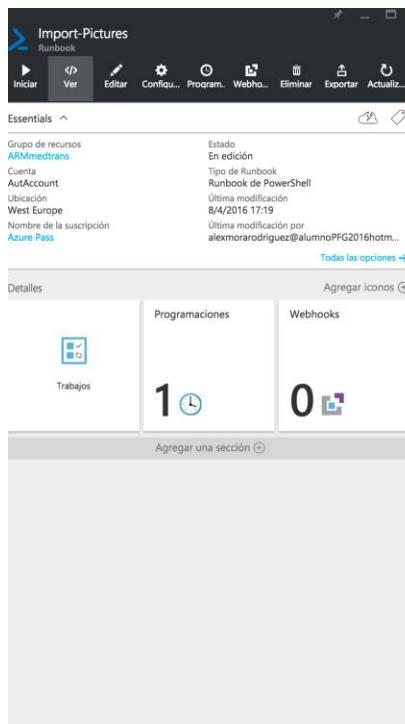
F Gratis	B Básico
- Por suscripción	- Por cuenta de Autom...
500 minutos para tra...	Minutos para trabaj...
Mensualidad gratuita	0.002 EUR/minuto de tra...
5 nodos DSC	Nodos DSC ilimitados
Mensualidad gratuita	6.00 EUR/nodo/mes
99,9 %	99,9 %
Contrato de nivel de serv...	Contrato de nivel de serv...
0.00 COSTE	
0.00 COSTE INICIAL (EUR)	

Fig. 43 Plan de tarifa de la cuenta de automatización Azure

2.3.6.2 Runbook

Un *runbook* es un conjunto de tareas que realizan algún proceso automatizado en el servicio de automatización de Microsoft Azure. Cualquier cosa que se pueda hacer con PowerShell se puede hacer en un *runbook*.

Un *runbook* puede ser de varios tipos (*script* de PowerShell puro, *script* de flujo de trabajo de PowerShell o gráfico). En la Fig. 44 se puede ver un *runbook* del tipo *script* PowerShell que utiliza un activo de la cuenta de automatización de tipo credenciales para conectarse a la suscripción de Microsoft Azure de forma no interactiva (sin ventana de conexión) e invocar un método de un API REST. Para ello se utiliza la función de PowerShell *Invoke-RestMethod* pasándole como parámetro, entre otros, la URL del servicio al que se desea invocar.



The screenshot shows the Azure Automation Runbook Editor interface. On the left, there's a navigation pane with tabs like 'Iniciar', 'Ver', 'Editar', 'Configur.', 'Program.', 'Webho...', 'Eliminar', 'Exportar', and 'Actualizar...'. Below it, there's a 'Essentials' section with details about the runbook: 'Grupo de recursos ARMmedtrans', 'Estado En edición', 'Tipo de Runbook Runbook de PowerShell', 'Última modificación 8/4/2016 17:19', and 'Nombre de la suscripción Azure Pass'. There's also a link to 'Todas las opciones'. On the right, the main area displays the PowerShell script:

```
[String] $automationAccountName = 'AutAccount',
53
54 [Parameter(Mandatory=$false)]
55 [String] $automationCredentialsName = 'AutCredentials',
56
57 [Parameter(Mandatory=$false)]
58 [String] $automationVariableSubscriptionID = 'SubscriptionId',
59
60 [Parameter(Mandatory=$false)]
61 [String] $automationVariableApiName = 'ApiName'
62
63
64 )
65
66 # Returns strings with status messages
67 #[OutputType([String])]
68
69
70 # You can retrieve the credential in a runbook using the Get-AutomationPSCredential
71 # activity and then use it with Login-AzureRmAccount (Add-AzureRmAccount is an alias)
72 # to connect to your Azure subscription.
73 # If the credential is an administrator of multiple Azure subscriptions, then you
74 # should also use Select-AzureSubscription to specify the correct one.
75
76 $cred = Get-AutomationPSCredential -Name $automationCredentialsName
77 Write-Output "Cred" $cred
78 $azAc = Login-AzureRmAccount -Credential $cred -Verbose -ErrorAction Stop
79 Write-Output "AzureAccount" $azAc
80
81 Write-Output "ResourceGroupName" $resourceGroupName
82
83 $subId = Get-AzureRmAutomationVariable -AutomationAccountName $automationAccountName -ResourceGroupName $resourceGroupName -Name $automationVariableSubscriptionID -Value
84 Write-Output "SubId" $subId
85
86 $azSb = Select-AzureRmSubscription -SubscriptionId $subId.value -ErrorAction Stop
87 Write-Output "Subscription" $azSb
88
89 $apiNameVariable = Get-AzureRmAutomationVariable -AutomationAccountName $automationAccountName -ResourceGroupName $resourceGroupName -Name $automationVariableApiName -Value
90 Write-Output "apiNameVariable" $apiNameVariable
91 $apiName= $apiNameVariable.Value
92 Write-Output "apiName" $apiName
93
94 $url = "https://$apiName.azurewebsites.net/mediotransporte/carga"
95 Write-Output "url" $url
96
97 Invoke-RestMethod -Method Post -Uri $url -Credential $credencial -TimeoutSec 8
```

Fig. 44 Runbook de tipo *Script PowerShell*

El *runbook* del ejemplo anterior se puede lanzar online o programarlo. En la Fig. 45 se muestran las programaciones asociadas al *runbook*. Una programación es uno de los activos de la cuenta de automatización que puede ser compartido entre todos los *runbooks* de dicha cuenta.

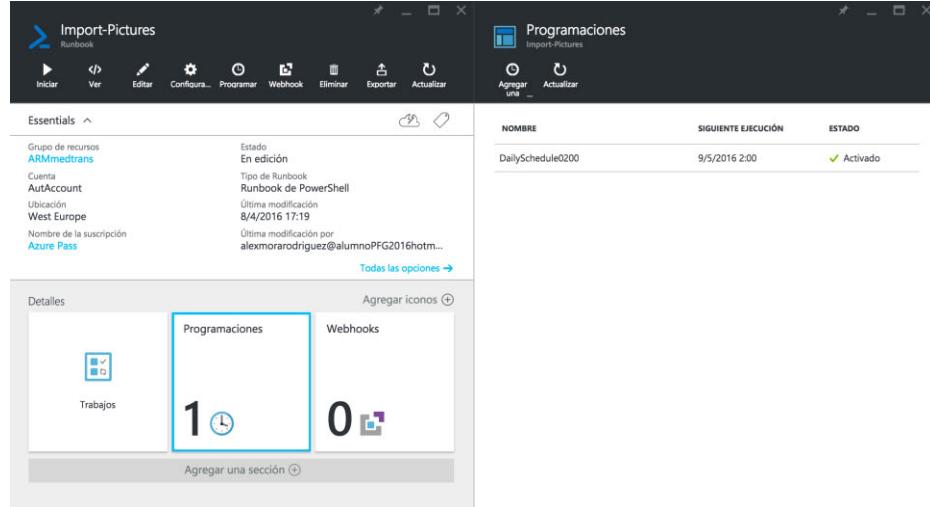


Fig. 45 Programaciones de un *runbook*

2.3.6.3 Activos (*assets*)

Una cuenta de automatización gestiona activos. Los activos son recursos disponibles globalmente y que pueden ser utilizados por los *runbooks*. De los distintos tipos de activos que se pueden gestionar en una cuenta de automatización (ver Fig. 46) se van a hacer especial hincapié en los siguientes: certificados, credenciales, variables y programaciones.

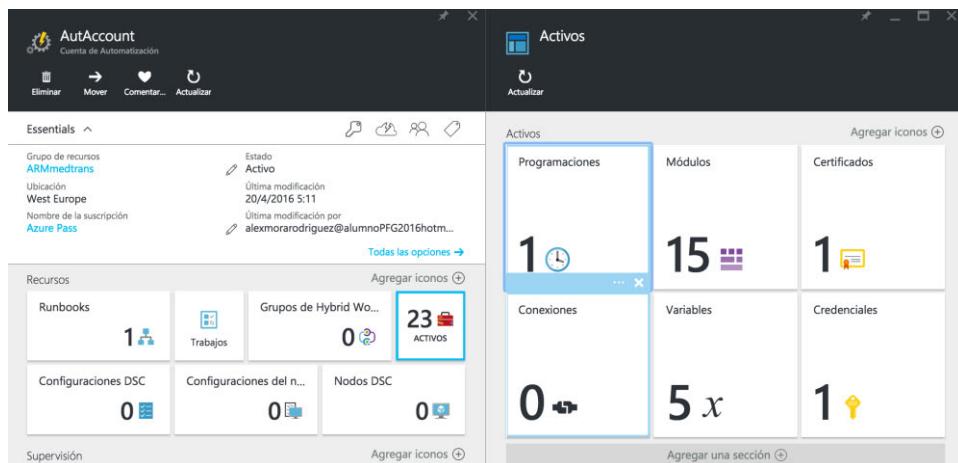


Fig. 46 Activos de una cuenta de automatización

2.3.6.3.1 Certificado

Los certificados se pueden almacenar de manera segura en una cuenta de automatización de Microsoft Azure, de manera que los *runbooks* pueden tener acceso a ellos mediante el uso de la actividad *Get-AutomationCertificate*. Esto permite crear *runbooks* que usen certificados para autenticación (ver Fig. 47).

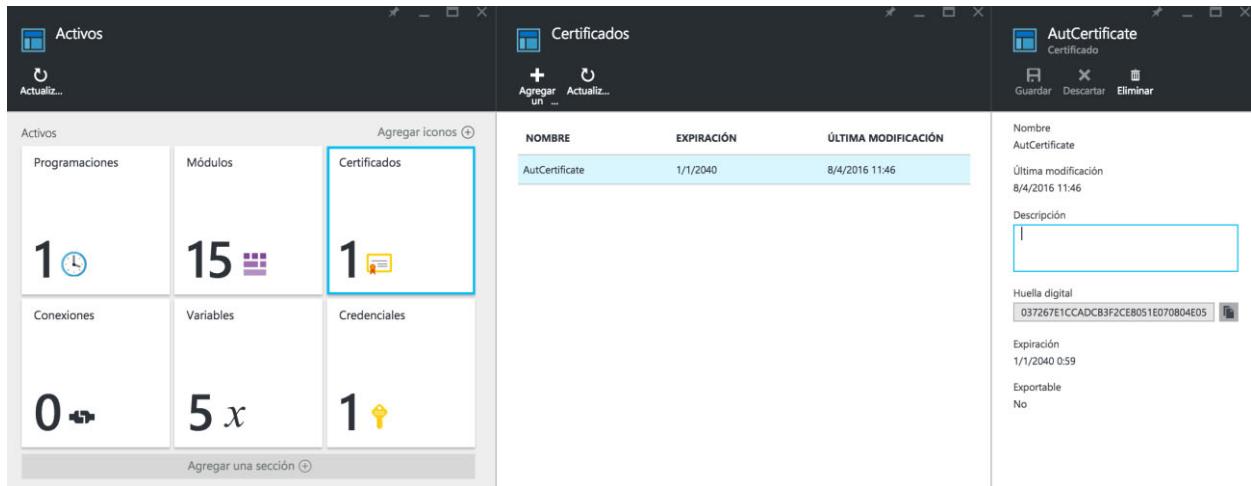


Fig. 47 Certificados de cuenta de automatización

Para crear un certificado propio se pueden seguir los siguientes pasos:

1. Generar un certificado X.509 con la herramienta de Windows "MakeCert.exe"¹²:


```
makecert.exe -r -n "CN=XXX" -pe -a sha256 -len 2048 -ss My "XXX.cer"
```
2. Generar un certificado PFX (Personal Information Exchange) con una clave privada para el certificado creado en el paso 1 utilizando PowerShell (como administrador):


```
$MyPwd = ConvertTo-SecureString -String "password" -Force -AsPlainText
$AzureCert = Get-ChildItem -Path Cert:\CurrentUser\My | where {$_.Subject -match "XXX"}
Export-PfxCertificate -FilePath C:\XXX.pfx -Password $MyPwd -Cert $AzureCert
```
3. Subir el certificado PFX generado en el paso anterior al servicio de automatización en la sección de activos (se solicitará el *password*)

¹² [https://msdn.microsoft.com/en-us/library/windows/desktop/aa386968\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa386968(v=vs.85).aspx)

2.3.6.3.2 Credencial

Un recurso de credencial de automatización contiene un objeto *PSCredential* que contiene credenciales de seguridad, como un nombre de usuario y una contraseña. Los *runbooks* pueden usar cmdlets que aceptan un objeto *PSCredential* para autenticación, o bien pueden extraer el nombre de usuario y la contraseña del objeto *PSCredential* para proporcionarlos a alguna aplicación o servicio que requiere autenticación. Las propiedades de una credencial se almacenan de manera segura en Automatización de Azure y es posible tener acceso a ellas en el *runbook* con la actividad *Get-AutomationPSCredential* (ver Fig. 48).

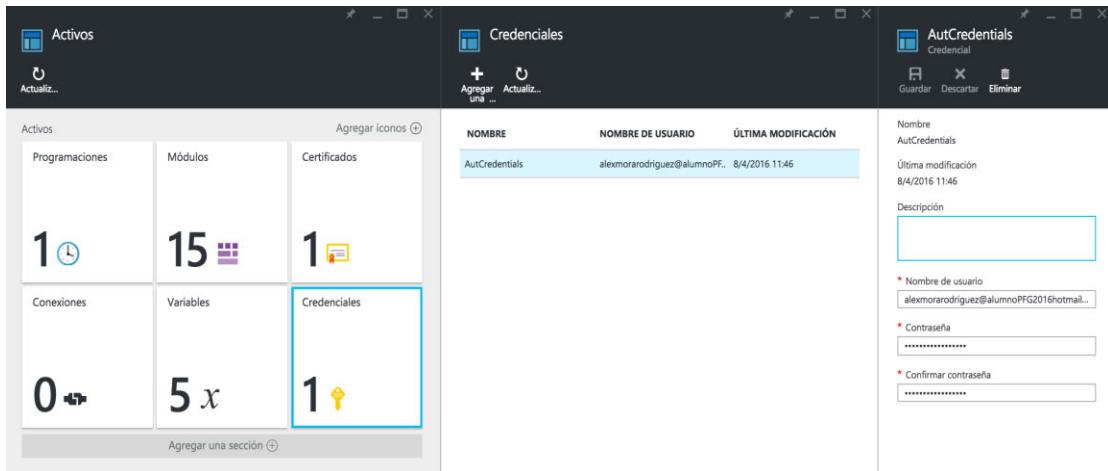


Fig. 48 Credenciales de cuenta de automatización

2.3.6.3.3 Variable

Las variables son valores que están disponibles para todos los *runbooks* de la cuenta de Automatización. Las variables de automatización son útiles para compartir un valor entre varios *runbooks*, compartir un valor entre varios trabajos del mismo *runbook*, administrar un valor desde el portal o desde la línea de comandos de Windows PowerShell que usan los *runbooks* (ver Fig. 49).

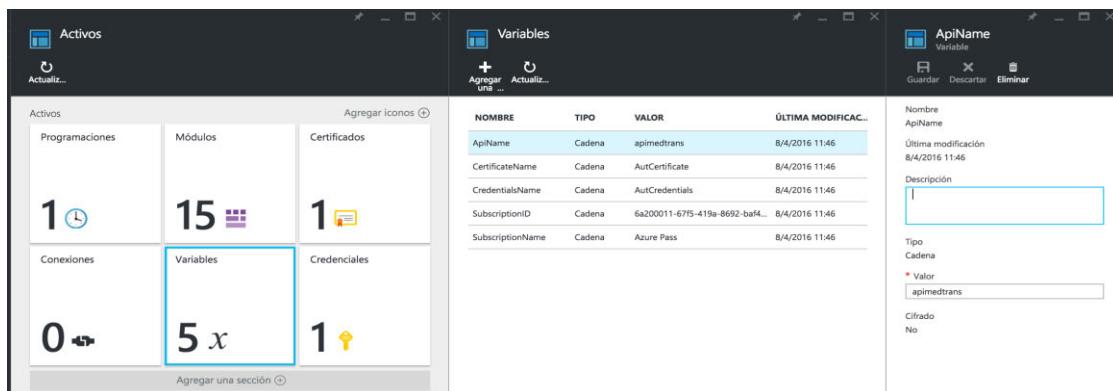


Fig. 49 Variables de cuenta de automatización

2.3.6.3.4 Programación

Las programaciones se usan para programar la ejecución automática de *runbooks*. Podría ser una sola fecha y hora para que el *runbook* se ejecute una vez. O bien, se puede tratar de una programación periódica que se repita cada día o cada hora para iniciar el *runbook* varias veces (ver Fig. 50).

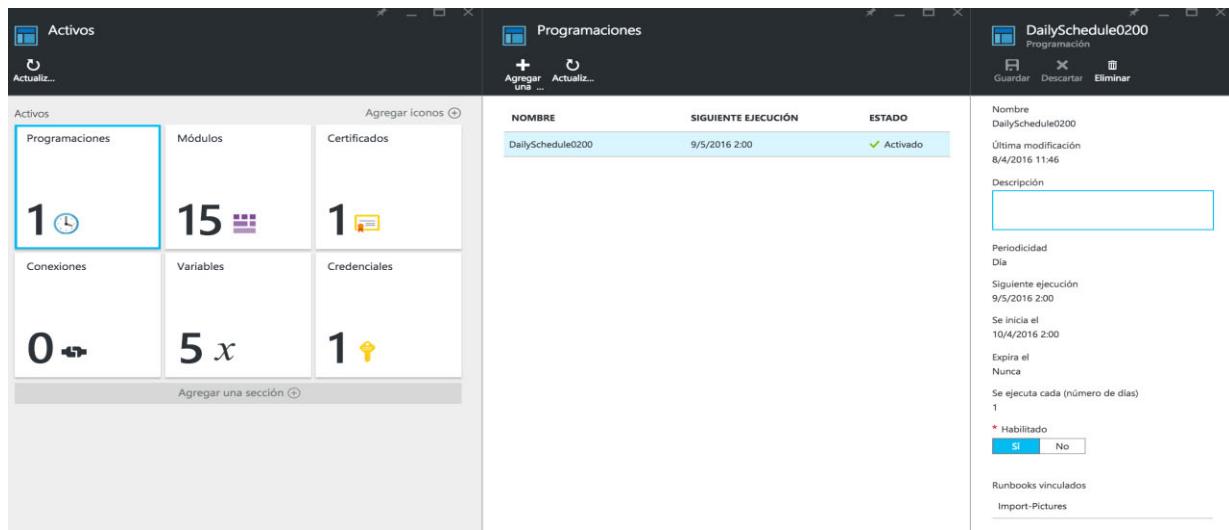


Fig. 50 Programaciones de una cuenta de automatización

2.4Android

Android es un sistema operativo (SO) móvil basado en Linux creado en 2003 por Andy Rubin, Rich Miner, Chris White y Nick Sears y que desde 2005 propiedad de Google [23]. Este SO fue originalmente diseñado para dispositivos móviles con pantalla táctil, como teléfonos móviles o tabletas. En estos dispositivos, el interfaz de usuario se basa principalmente en manipulación directa pinchando, arrastrando, etc. los objetos de la pantalla. Posteriormente, se ha desarrollado Android TV para televisiones, Android Auto para coches y Android Wear para relojes de pulsera, cada uno con su correspondiente interfaz especializado [24].

Android es de libre distribución, apoyado por una enorme comunidad de usuarios desarrolladores. Esto permite a los fabricantes desarrollar dispositivos con menores costos, ya que tan solo deben diseñar el hardware y adaptar el código fuente de Android. Desde el año 2015, según datos de IDC¹³, la cuota de mercado mundial del SO Android lleva 3 años

¹³ <https://www.idc.com/>

consecutivos estando 6 veces por encima de su más directo competidor, iOS (ver Fig. 51). Además, en base a estas mismas estadísticas, parece que en los últimos 3 años se ha mantenido constante el valor de ambas fuerzas. En cualquier caso, en el mercado de las tecnologías nunca está nada dicho y las tendencias pueden cambiar de un día para otro [25].

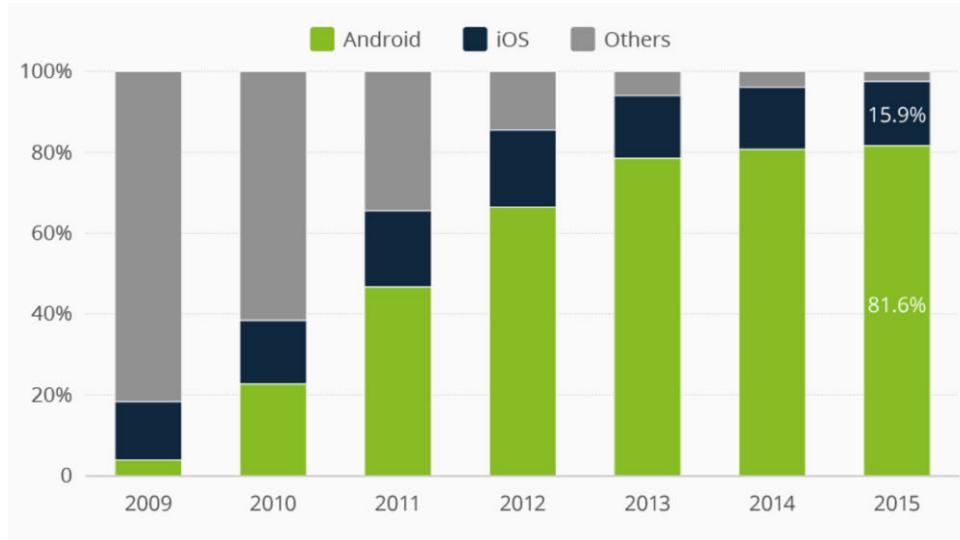


Fig. 51 Evolución cuota de mercado SO móviles entre 2009 y 2015 en el mundo [25]

Respecto a los datos de cuota de mercado en España, en el segundo cuarto del 2015 están muy próximos a la media mundial: Android domina el mercado con un 71.42% frente a un 26.5% de iOS. En la Fig. 52 se puede ver la distribución mundial de iOS y Android.

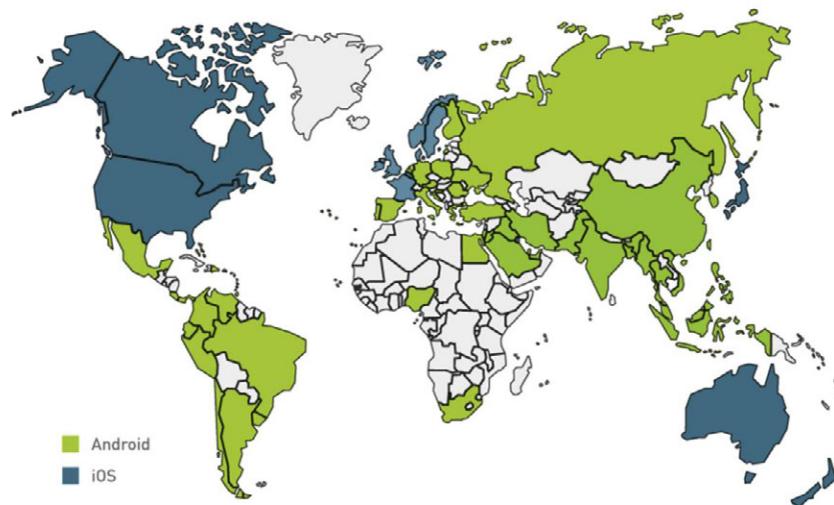


Fig. 52 Mapa distribución mundial iOS y Android [25]

Dentro del historial de versiones de Android, Lollipop con un 35% de cuota es la versión más utilizada de Android actualmente, en el año 2016 (ver Fig. 53).

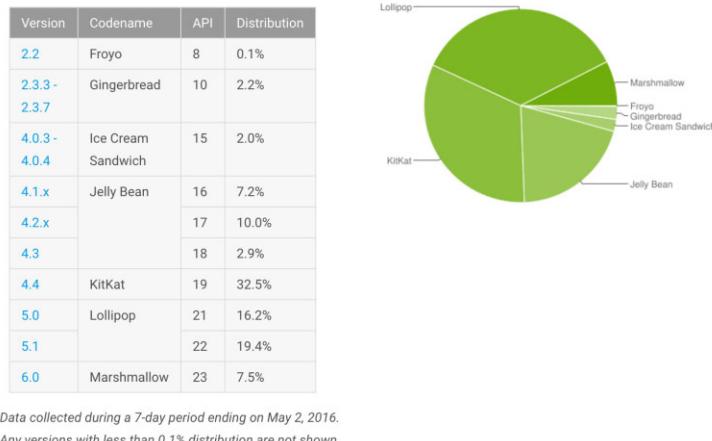


Fig. 53 Porcentaje de uso distintas versiones Android en 2016 [26]

2.4.1 Android Studio 2.0

Android Studio 2.0 es un entorno de desarrollo que hoy en día se ha convertido en la forma más rápida y recomendable de construir aplicaciones de calidad para la plataforma Android, incluyendo teléfonos y tabletas, Android Auto, Android Wear y Android TV [27]. Al ser el IDE oficial de Google, Android Studio incluye todo lo que se necesita para construir una aplicación, incluyendo un editor de código, herramientas de análisis de código, emuladores, etc. Además, ha mejorado considerablemente respecto a las versiones anteriores (ver Fig. 54):

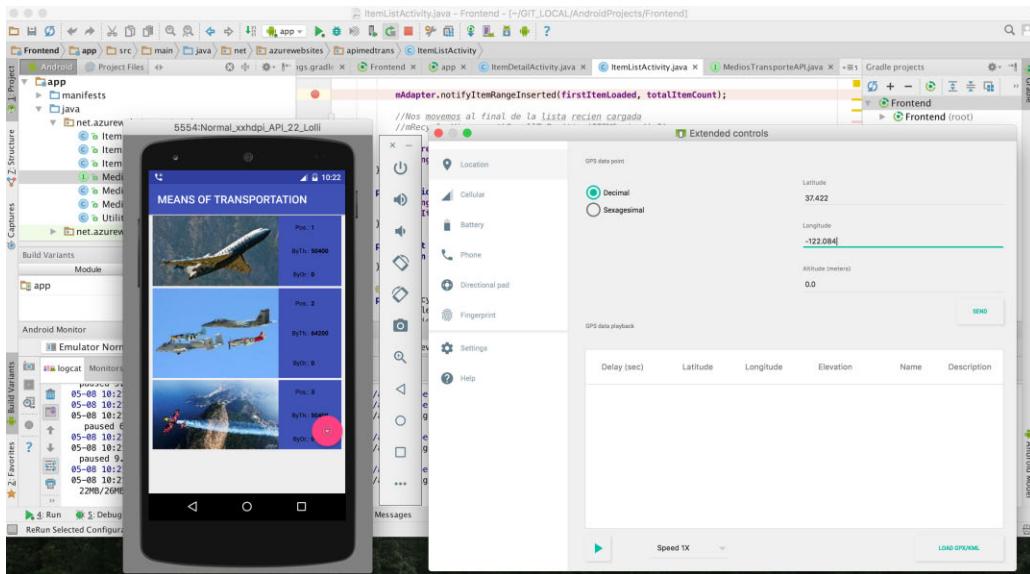


Fig. 54 UI Android 2.0

- Ejecución instantánea: Permite hacer cambios en caliente en la aplicación en ejecución.
- AVD (*Android Virtual Device*): El nuevo emulador corre alrededor de 3 veces más rápido que el anterior y con las nuevas mejoras ADB se pueden instalar datos y aplicaciones 10 veces más rápido en el emulador que en un dispositivo físico. Además incluye otras características tales como manejo de llamadas, batería, red, GPS, etc.
- Integración con laboratorio de test en la nube: permite mejorar la calidad la aplicación facilitando la elaboración de tests en un amplio rango de dispositivos físicos Android en un laboratorio de test en la nube desde justo Android Studio

2.4.2 Estructura de un proyecto en Android Studio 2.0

Un proyecto en Android es único y engloba a todos los demás elementos. Dentro de un proyecto se pueden incluir varios módulos, que pueden representar aplicaciones distintas, versiones diferentes de una misma aplicación, o distintos componentes de un sistema. En la mayoría de los casos, se trabaja con un proyecto que contendrá un sólo módulo que corresponde a la aplicación principal. En el siguiente ejemplo, que corresponde al caso de estudio que se detalla en la Sección 3 se tiene el proyecto "*Frontend*" que contiene al módulo "*app*" que contendrá todo el software de la aplicación (ver Fig. 55).

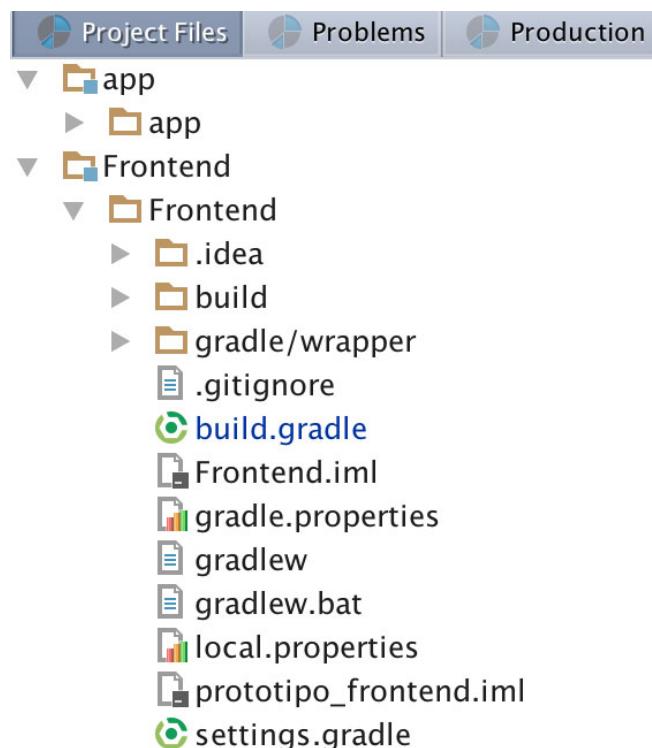


Fig. 55 Proyecto Android "*FrontEnd*" y módulo "*app*"

El módulo "**app**" de la aplicación contiene, generalmente, los componentes que se enumeran y describen a continuación (ver Fig. 56):

- **/app/src/main/java:** contiene todo el código fuente de la aplicación. Inicialmente, Android Studio creará por nosotros el código básico de la pantalla (actividad o *activity*) principal de la aplicación.
- **/app/src/main/res/:** contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, *layouts*, cadenas de texto, etc. Los diferentes tipos de recursos se pueden distribuir entre las siguientes subcarpetas:
 - **/res/drawable:** contiene las imágenes y otros elementos gráficos usados por la aplicación. Para poder definir diferentes recursos dependiendo de la resolución y densidad de la pantalla del dispositivo se suele dividir en varias:
 - **/drawable**
 - **/drawable-ldpi**
 - **/drawable-mdpi**
 - **/drawable-hdpi**
 - **/drawable-xhdpi**
 - **/res/mipmap:** contiene los iconos de lanzamiento de la aplicación (el ícono que aparecerá en el menú de aplicaciones del dispositivo) para las distintas densidades de pantalla existentes. Se dividirá en varias subcarpetas dependiendo de la densidad de pantalla:
 - **/mipmap-mdpi**
 - **/mipmap-hdpi**
 - **/mipmap-xhdpi**
 - **/mipmap-xxhdpi**
 - **/mipmap-xxxhdpi**
 - **/res/layout:** contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica. Para definir distintos *layouts* dependiendo de la orientación del dispositivo se puede dividir también en subcarpetas:
 - **/layout**
 - **/layout-w300dp-land**

- **/res/values:** contiene otros ficheros XML de recursos de la aplicación, como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), arrays de valores (arrays.xml), tamaños (dimens.xml), etc.
 - **/res/raw:** contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.
 - **/res/menu:** contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.
 - **/res/color:** contiene ficheros XML de definición de listas de colores según estado.
 - **/res/animator:** contienen la definición de las animaciones utilizadas por la aplicación.
 - **/app/src/main/AndroidManifest.xml:** contiene la definición en XML de muchos de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, icono, ...), sus componentes (pantallas, servicios, ...), o los permisos necesarios para su ejecución.
 - **/app/build.gradle:** contiene información necesaria para la compilación del proyecto, por ejemplo la versión del SDK (*Software Development Kit*)¹⁴ de Android utilizada para compilar, la mínima versión de Android que soportará la aplicación, referencias a las librerías externas utilizadas, etc. Más adelante veremos también más detalles de este fichero. En un proyecto pueden existir varios ficheros “*build.gradle*”, para definir determinados parámetros a distintos niveles. En el caso de estudio que veremos más adelante, se ve que existe un fichero “*build.gradle*” a nivel de proyecto, y otro a nivel de módulo dentro de la carpeta “*/app*”. El primero de ellos definirá parámetros globales a todos los módulos del proyecto, y el segundo sólo tendrá efecto para cada módulo en particular.
 - **/app/libs:** contiene las bibliotecas java externas (ficheros .jar) que utiliza la aplicación. Normalmente no se incluye directamente aquí ninguna librería, sino que se hace referencia a ellas en el fichero “*build.gradle*”.
-

¹⁴ SDK (*Software Development Kit*) o kit de desarrollo de software es un conjunto de herramientas de desarrollo de software que permite al desarrollador de *software* crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, *frameworks*, plataformas de *hardware*, computadoras, videoconsolas, sistemas operativos, etc. Generalmente incluye APIs, herramientas de desarrollo y documentación.

- **/app/build:** contiene una serie de elementos de código generados automáticamente al compilar el proyecto. A destacar sobre todo el fichero llamado "**R.java**", donde se define la clase R. Esta clase R contendrá en todo momento una serie de constantes con los identificadores (ID) de todos los recursos de la aplicación incluidos en la carpeta `/app/src/main/res/`, de forma que se pueda acceder fácilmente a estos recursos desde el código java a través de dicho dato. Así, por ejemplo, la constante R.layout.activity_main contendrá el ID del layout "activity_main.xml" contenido en la carpeta `/app/src/main/res/layout/`.

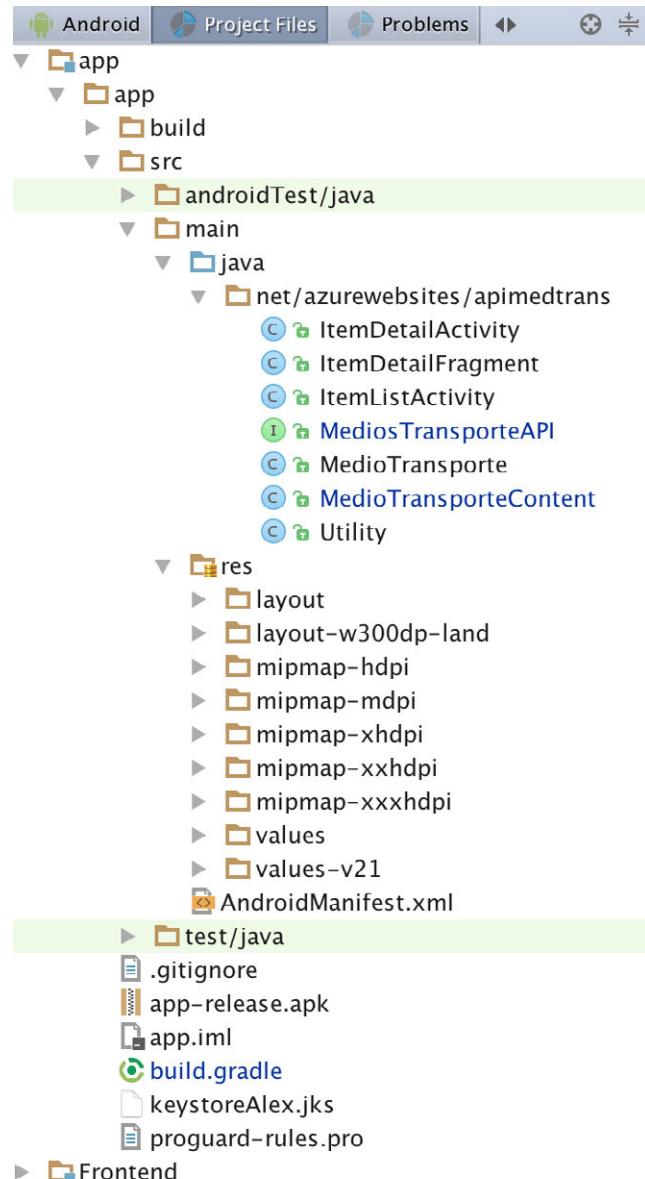


Fig. 56 Estructura de una aplicación Android

2.5 Desarrollo Ágil de Software

El desarrollo ágil de software es un paradigma de desarrollo que se basa en una serie de principios y valores de desarrollo iterativo, con grupos auto-organizados y multidisciplinarios, y soportado por un nuevo conjunto de metodologías de desarrollo de *software* [28].

Su historia se remonta al 17 de febrero de 2001, cuando varios desarrolladores de *software* de reconocido prestigio hoy en día (Robert C. Martin, Jim Highsmith, Alistair Cockburn, Ron Jeffries, Jeff Sutherland, Dave Thomas, Martin Fowler, etc.) se reunieron en Utah para discutir nuevas metodologías de desarrollo. Estos desarrolladores se habían dado cuenta que un gran número de proyectos fracasaban debido a cuestiones relacionadas con el presupuesto y planificación, la interfaz del software que era *non-user-friendly* y a la poca flexibilidad para permitir cambios en su *software*. Además, habían comprobado que la principal causa de todos estos problemas solía ser la falta de comunicación y coordinación entre las partes involucradas. Es aquí cuando surgió el **Manifiesto Ágil** (ver Fig. 57) junto con sus **Principios** (ver Fig. 58).



Fig. 57 Manifiesto por el Desarrollo Ágil de Software [29]

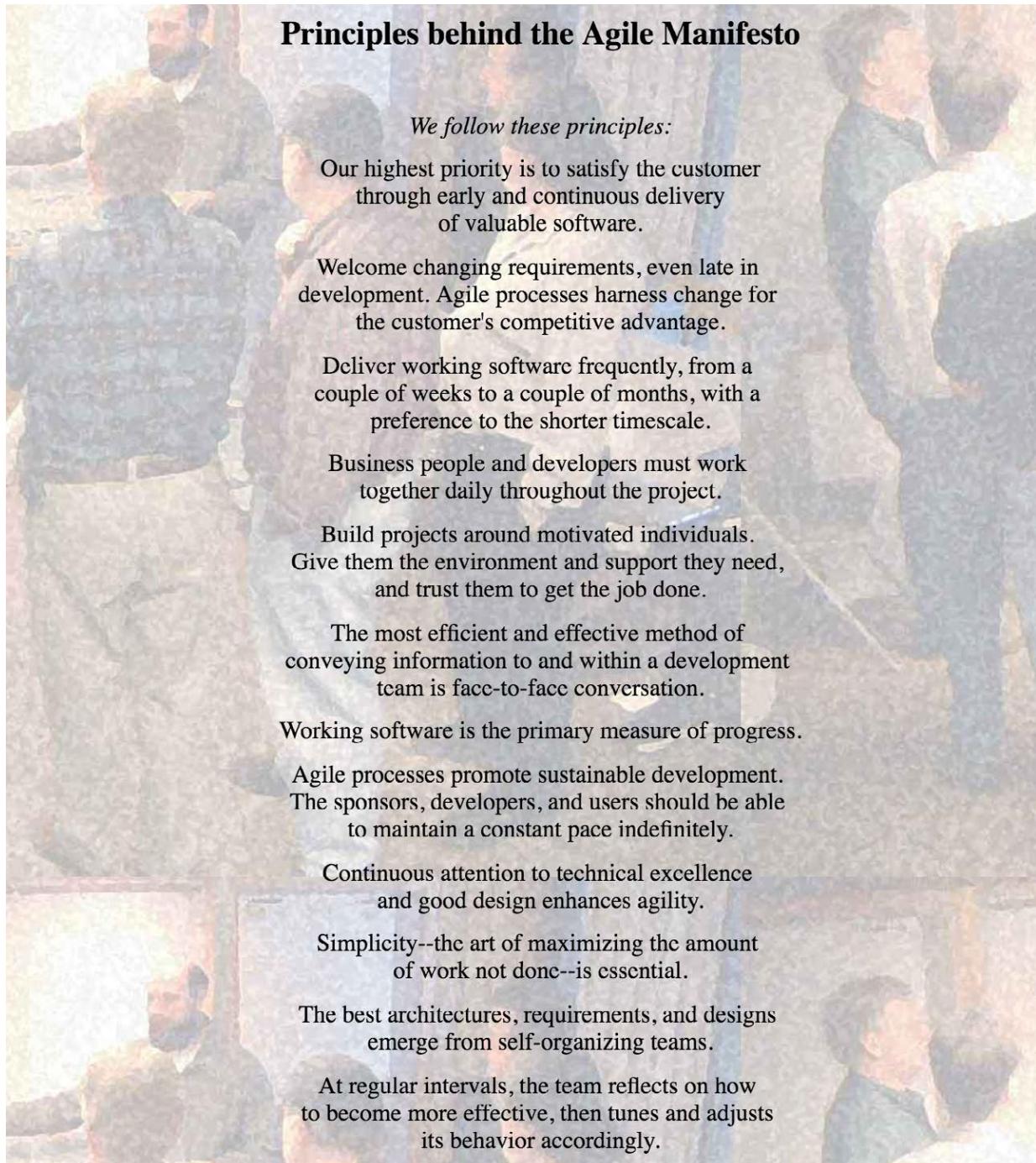


Fig. 58 Principios del Manifiesto Ágil [30]

El desarrollo ágil de software se apoya en un conjunto de **prácticas** que cubren áreas tales como requisitos, diseño, modelado, código, test, gestión de procesos, calidad, etc. Algunas de estas prácticas son:

- **Sobre el desarrollo del software:**

- Planificación conjunta:
 - Recopilación informal de requisitos.
 - Estimación inicial de los requisitos recopilados.
 - Priorización de los requisitos.
 - Selección de los requisitos más importantes para formar versiones parciales del producto, pero completa funcionalmente (constituyen una entrega).
- Pequeñas y frecuentes revisiones:
 - Ciclo de vida incremental y evolutivo.
 - Al final de cada ciclo, validación del producto por parte del cliente.
- Diseño sencillo: desarrollar la solución más sencilla que cumpla las especificaciones y sólo de los requisitos actuales sin predecir las necesidades futuras.
- Accesibilidad del cliente: siempre disponible para clarificar y validar los requisitos

- **Sobre el código:**

- Estandarización: el código desarrollado debe ser entendido por todo el equipo.
- Repositorio común: el código pertenece al equipo y es compartido.
- Revisión continua y refactorización: el código se revisa y reestructura continuamente para mejorar su estructura.
- Integración continua: el repositorio común contiene las últimas versiones del código, el cual se compila y se prueba continuamente para detectar fallos cuanto antes.

- **Sobre pruebas:**

- Test unitario: automático y exhaustivo, previo a la creación del código. Se deben superar todos los tests unitarios para que un módulo de código se considere válido.
- Test de aceptación: garantiza que el sistema proporciona la funcionalidad requerida.

- **Sobre la forma de trabajo:**

- Reuniones diarias: reuniones de 15 minutos, antes de empezar el trabajo, en el mismo lugar y hora para que cada miembro del equipo haga un resumen de que es lo hizo ayer y que es lo que va a hacer.
- Revisión del *sprint*/iteración: demo informativa donde se presenta lo que se ha hecho para el *sprint*/iteración.
- Reuniones retrospectivas: se evalúa como se ha trabajado en el último *sprint*/iteración para tratar de mejorar el proceso.
- Programación en parejas: colaboración e interacción entre los miembros del equipo.
- No hacer horas extra: implican cansancio y poca calidad en el trabajo.
- Áreas de trabajo abiertas y comunes: facilitan la comunicación entre los desarrolladores y clientes.

Respecto al desarrollo ágil de software, existen varios metodologías [31] tales como Agile, Scrum, Extreme Programming (XP), Lean Development, etc. En particular, la metodología Scrum (ver Fig. 59) comienza con una pila priorizada de objetivos o requisitos, escritos en forma de historias de usuario (HU), que de alguna manera van a configurar el plan de proyecto. Una HU describe necesidades requeridas por el sistema en una o dos frases utilizando el lenguaje común del usuario, así como las discusiones con los usuarios y las pruebas de validación. Las HU deben ser escritas por los clientes. Éstas son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las HU permiten responder rápidamente a los requisitos cambiantes. El conjunto de HU constituye el *product backlog*. Del *product backlog* el equipo selecciona el grupo de HU que se van a realizar en el siguiente iteración (*sprint*), de 2 a 4 semanas. Generalmente, esta selección está basada en el ratio prioridad/esfuerzo, seleccionándose las HU que aportan mayor valor y requieren menos esfuerzo. Una vez que el equipo ha decidido qué HU se van a implementar comienza el *sprint*. Durante el *sprint*, los miembros del equipo llevan a cabo una corta reunión diaria de unos 15 minutos como máximo, llamada *daily meeting*, a modo de sincronización. Cuando acaba el *sprint*, se presenta el valor al dueño del producto para que lo revise y apruebe su subida a producción en la *review meeting*, y por otro lado, el equipo tiene una *retrospective meeting*, en la cual se evalúa cómo se ha trabajado en el último *sprint*.

The Agile - Scrum Framework

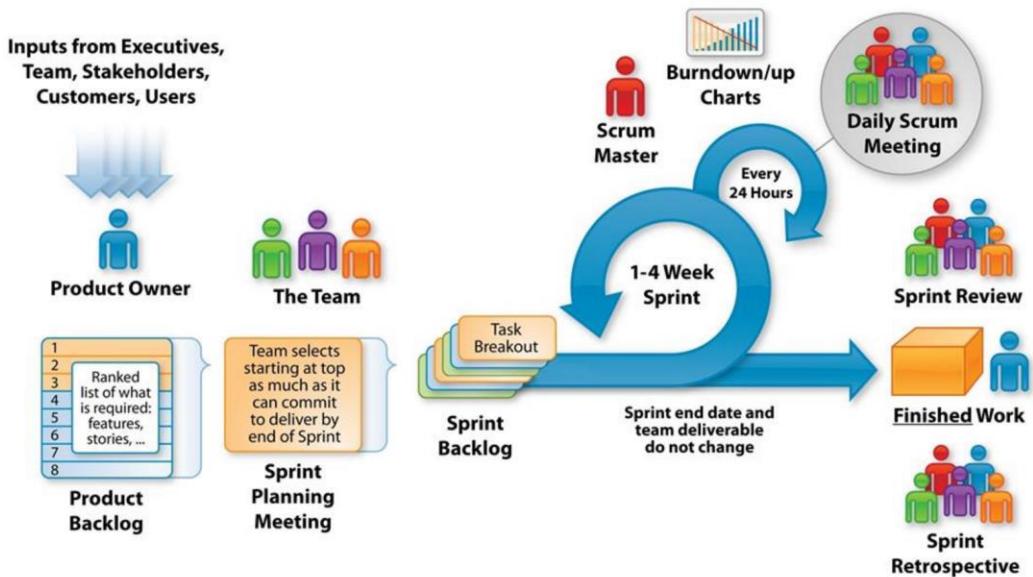


Fig. 59 Proceso de desarrollo ágil de software. Metodología SCRUM [32]

Un elemento que se suele utilizar en Scrum para ayudar a los miembros del equipo en la organización del *sprint* es el *board*. Pueden ser físicos o digitales y consisten en un panel que representa el *sprint* actual con varias columnas que representan los distintos estados (*To do/In progress/Done*) en que se pueden encontrar las tareas que componen una HU (ver Fig. 60).

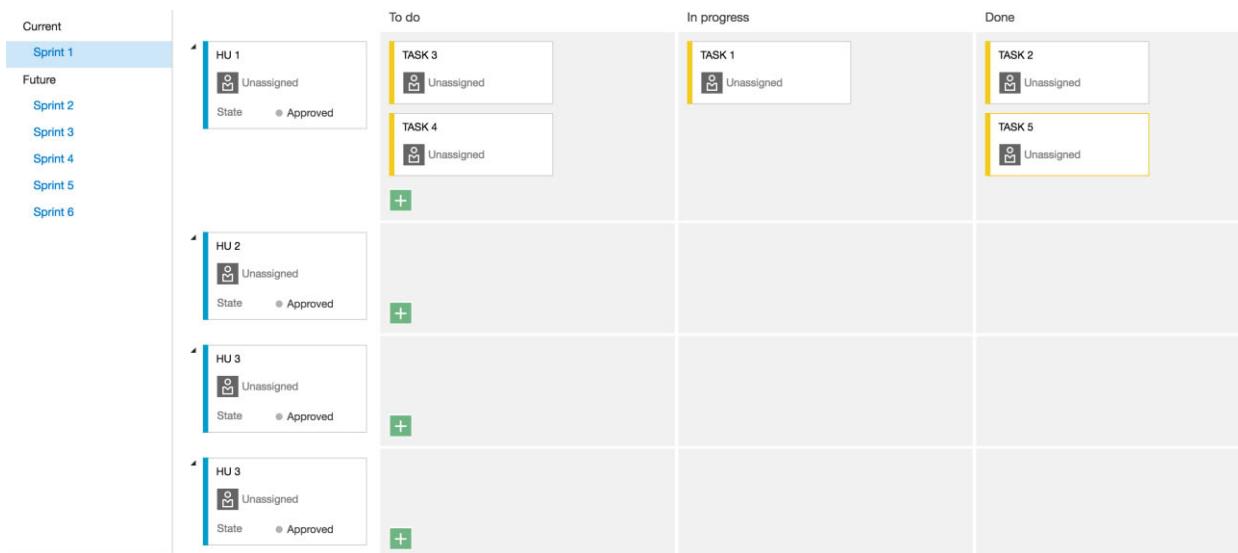


Fig. 60 Board de Scrum de VSTS

2.6 DevOps

Tradicionalmente, en las grandes compañías suele existir una división de responsabilidades muy estricta donde las tareas son llevadas a cabo por grupos muy especializados, que se acaban convirtiendo en silos. En TIC, los desarrolladores crean aplicaciones y los equipos de operaciones las despliegan en las infraestructuras que gestionan. Esta organización se ha comprobado que es ineficiente ya que se producen retrasos e imprecisiones cuando un grupo tiene que transferir las tareas al otro (ver Fig. 61).

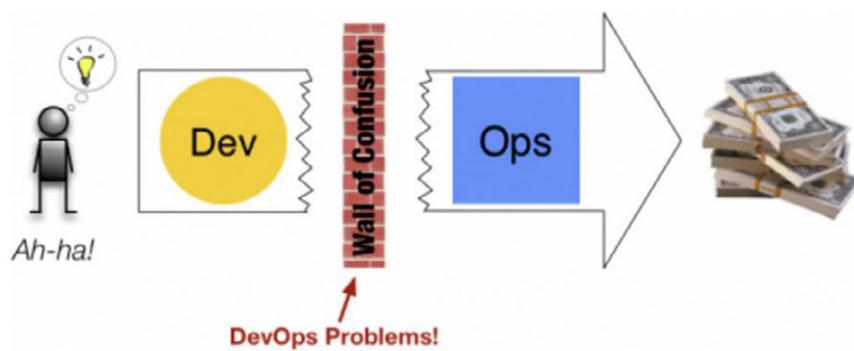


Fig. 61 Desarrollo y Operaciones por separado [33]

Para mejorar la eficiencia y, de esta manera, reducir el coste y el *Time To Market* (TTM) surge el movimiento técnico y cultural DevOps, que integra las tareas de Desarrollo y Operaciones (ver Fig. 62).

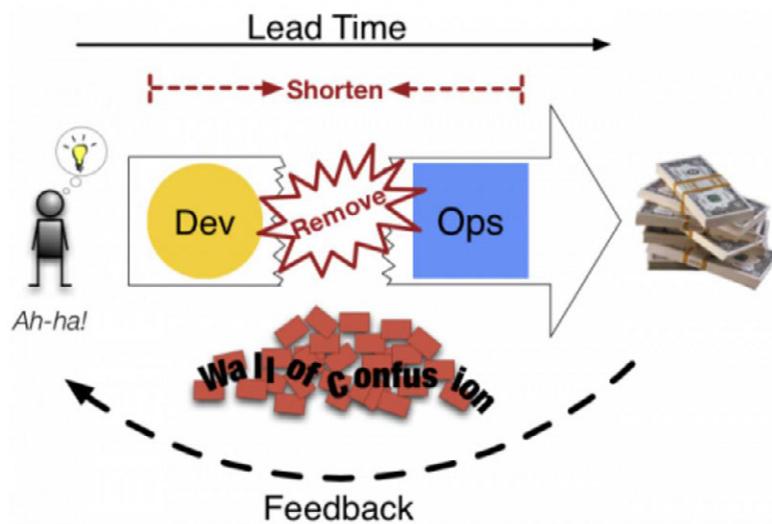


Fig. 62 Desarrollo y Operaciones integrados [33]

DevOps se origina de alguna manera a partir de las metodologías ágiles [34]. Por todos es sabido que uno de los principios del proceso de desarrollo ágil es entregar software funcional en pequeñas y frecuentes entregas para aumentar el valor entregado al cliente. De esta forma, se elimina la primera grieta en el ciclo de vida de desarrollo (ver Fig. 63) pero tiene como consecuencia inmediata que al equipo de Operaciones de TIC se le empieza a acumular un elevado número de entregables para desplegar. Por lo tanto, el equipo de Operaciones no puede ir al ritmo del equipo de Desarrollo y se convierte en un cuello de botella [35].

Por tanto, se puede decir que DevOps surge como un complemento al proceso ágil de desarrollo de software, eliminando la segunda grieta (ver Fig. 63) en el ciclo de vida de desarrollo, asegurando así que el código se despliega inmediatamente en producción y no se interrumpe el proceso.

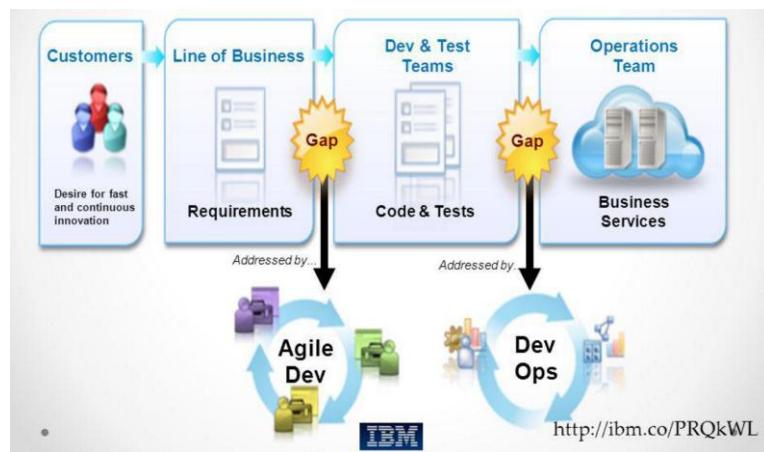


Fig. 63 Eliminación grietas en el ciclo de vida de desarrollo [35]

El flujo de trabajo (*workflow*) de DevOps (ver Fig. 64) consiste en desarrollar una aplicación, desplegarla, monitorizarla y aprender de su uso en producción, modificarla en respuesta a lo que se haya visto y repetir el ciclo [36].

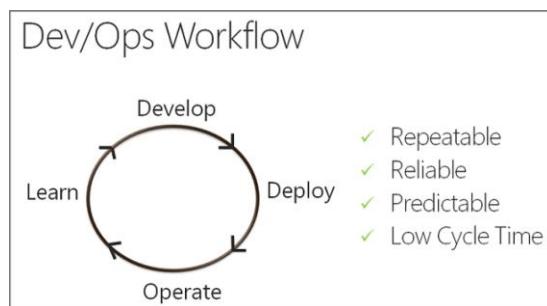


Fig. 64 DevOps workflow [36]

Los equipos más exitosos de desarrollo pueden llegar a desplegar varias veces en un día, ya que esta es la mejor manera de responder a las necesidades del cliente. Para que sea posible, el ciclo de desarrollo y despliegue tiene que ser repetible, predecible y tan corto como sea posible para entregar valor a los clientes más rápidamente y mejorar la calidad del *software*. La idea es el tiempo que transcurre desde que se tiene una idea para una característica y los clientes la están usando y proporcionando *feedback* nuevamente, debe ser tan corto como sea posible.

Para establecer este proceso se deben seguir los siguientes buenas prácticas o patrones: automatización total, control de versiones, e integración y entrega continua

2.6.1 Automatización Total

Las tareas manuales son lentas y propensas a errores, por lo tanto, automatizar tantas como sea posible ayuda a establecer un *workflow* rápido, fiable y ágil [12]. La automatización de las tareas de desarrollo y despliegue de una aplicación que hace uso de una gran cantidad de servicios en la nube es si cabe más importante para evitar errores. El portal de gestión de Microsoft Azure permite monitorizar y gestionar todos los servicios. Ésta gestión es una forma fácil de crear, configurar y eliminar servicios, sin embargo no deja de ser un proceso manual. Si se van a desarrollar aplicaciones en producción, sea el tamaño que sea, especialmente en equipo, se recomienda utilizar el portal de Microsoft Azure para su gestión y después automatizar el proceso que se va a estar repitiendo continuamente. Esta automatización se puede conseguir mediante el uso de *scripts* en PowerShell (ver Sección 2.3.1) para los que utilicen sistemas Windows o *scripts* en *Azure Command-Line Interface* (CLI)¹⁵ para los que utilicen sistemas Linux o Mac.

2.6.2 Control de versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación. El control de versiones es esencial en todos los proyectos de desarrollo y no solo cuando se trabaja en equipo ya que, entre otras cosas, nos da la posibilidad de deshacer acciones y salvaguardar todo lo referente a un proyecto evitándonos perder tiempo si algo sale mal [12].

¹⁵ <https://azure.microsoft.com/en-us/documentation/articles/xplat-cli-install>

Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión. En el mercado existen multitud de sistemas de control de versiones como, por ejemplo, el sistema de control de versiones distribuido Git¹⁶ o el sistema de control de versiones centralizado Team Foundation Version Control (TFVC)¹⁷.

En este sentido, Visual Studio Team Services (VSTS) proporciona una solución ALM completa, gratuita para equipos de hasta 5 usuarios, que ofrece un conjunto de servicios de colaboración con tecnología de la nube para que el equipo pueda trabajar de manera eficiente en proyectos de software de cualquier índole y envergadura [37]. Así VSTS soporta:

- Sistemas de control de versiones, en concreto Git y TFVC.
- Integración continua y entrega continua (despliegue) en Microsoft Azure.
- Test de carga automatizados: simulan una carga de miles de usuarios posibilitando encontrar cuellos de botella antes de desplegar en producción.
- Comunicación en tiempo real entre los miembros del equipo.
- Gestión de proyectos ágil mediante la gestión del *product backlog* del producto, de los *sprints* y de boards (pizarras/kanban) de tareas pendientes, realizadas y cerradas.

2.6.3 Integración y Despliegue Continuo

La integración continua [38] es una práctica, propuesta por Martin Fowler, cuya base es integrar de manera automática lo más frecuentemente posible la compilación, ejecución de pruebas y despliegue de un proyecto. De esta manera, la capacidad de detectar errores se incrementa, ya que los errores se detectan en un tiempo temprano. La integración continua activa la construcción automática (que suele incluir test unitarios) de la aplicación en cuanto se produzca algún cambio en el código fuente gestionado en el sistema de control de versiones.

El despliegue continuo va un paso más allá: después que se ha construido la aplicación, se despliega en el entorno deseado. Generalmente se recomienda que se haga despliegue continuo de forma automática a los entornos de desarrollo, test y ensayo y de forma manual y controlada, tras unas pruebas de aceptación, al entorno de producción. Si no se considerase necesario un chequeo previo, se puede optar por desplegar en producción directamente.

¹⁶ <https://git-scm.com>

¹⁷ <https://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx>

3 Caso de Estudio

Un caso de estudio es una técnica didáctica o investigación para el análisis y evaluación cualitativa de un fenómeno susceptible de estudio. Consiste en la descripción de una situación real o ficticia para que sea analizada para mejorar el conocimiento y habilidades en el área. Un caso de estudio aporta un toque de realismo para practicar y aprender a identificar como aplicar el conocimiento y habilidades recientemente adquiridas.

3.1 Descripción

El caso de estudio de este PFG consiste en el desarrollo de una aplicación móvil Android con *backend* desplegado en la nube con el fin de demostrar el uso de servicios avanzados de la plataforma Microsoft Azure. El caso de estudio diseñado es lo suficientemente realista y completo como para permitir poner en práctica un conjunto de servicios para el desarrollo y despliegue de aplicaciones en la nube de la PaaS de Microsoft Azure, incluyendo servicios para gestión integral del ciclo de vida de las aplicaciones (ALM) ofrecidos por VSTS. Para su desarrollo se ha contado con una suscripción de 4 meses a la plataforma de servicios *Cloud* Microsoft Azure con un crédito de 80 Euros/Mes que ha permitido costear el contrato de los recursos y servicios *Cloud* necesarios. En concreto, los recursos y servicios de Microsoft Azure que este caso de estudio aborda son: el servicio de aplicaciones, en concreto API app; el servicio de almacenamiento, en particular *blobs* y tablas; el servicio de automatización; y finalmente, los servicios ofrecidos por VSTS para ALM, en concreto sistemas de control de versiones Git, integración y despliegue continuo, y test de carga automatizados. Además se ha automatizado la creación y configuración de todos los recursos y servicios de la nube necesarios.

La aplicación desarrollada permite visualizar fotos en forma de lista de miniaturas (*thumbnails*) y, en caso que el usuario seleccione algún *thumbnail* en particular, su foto original. La aplicación no requiere autenticación ni autorización de acceso. A este respecto, es posible que, en momentos puntuales, el número de accesos se incremente de forma ilimitada e indeterminada. En este caso, la solución propuesta no debe bajar su rendimiento. Por otra parte, se estima que nunca va a tener picos superiores a los 100.000 accesos. Además, la aplicación no requiere de funcionalidad de mantenimiento de las fotos: la carga de nuevas fotos se realiza mediante un proceso *batch* con planificación diaria. En este sentido, no deben existir impedimentos al crecimiento ilimitado del tamaño necesario para almacenar las fotos. En cualquier caso, se estima que dicho tamaño no va a superar 1TB. Finalmente, el mercado objetivo inicial de la aplicación son los usuarios españoles de Android pero debe estar preparada para ser extendida a otros países y a iOS en el futuro. Es decir, la solución tiene que ser capaz de operar con sistemas de diversa índole.

El caso de estudio también contempla una serie de requisitos que el proceso de desarrollo y entrega debe cumplir. El objetivo es desarrollar una aplicación móvil en la nube que tiene que estar en el mercado en menos de 4 meses. La entrega de versiones de la aplicación es iterativa e incremental. La primera versión se entrega a las 4 semanas del comienzo del desarrollo y el resto de entregas de versiones se realiza regularmente cada 2 semanas. Además, el desarrollo del proyecto soporta la gestión de cambios temprana y en cualquier momento, para dar respuesta a posibles fallos, así como cambios en los requisitos no planificados (ej. cambios en el negocio). Esto implica modificaciones en el desarrollo de la aplicación y su despliegue en producción de forma inmediata y transparente al usuario. Con este objeto, los desarrolladores trabajan en el mismo entorno de desarrollo y, cuando acaben sus tareas particulares y pasen sus pruebas unitarias, lo suben al repositorio Git. La anterior subida desencadena de forma automática la construcción y despliegue en el susodicho entorno de desarrollo para hacer visibles los cambios al resto de desarrolladores lo antes posible. Cuando todos los desarrollos que vayan a componer la entrega se hayan finalizado y estén subidos al repositorio, un operador despliega dicha entrega en el entorno de test. Despues de que se pasen las pruebas de carga y los test correspondientes de forma manual, un operador despliega la entrega al espacio de implementación o ensayo (*staging slot*) del entorno de producción. Finalmente, un usuario cualificado debe dar el visto bueno a este despliegue en este entorno pre-productivo, tras lo cual un operador efectuará el intercambio de espacios y la aplicación pasa a producción. Como se puede ver, gracias a los espacios de producción y ensayo, el despliegue en producción es más seguro y evitamos demoras de despliegue y configuración.

3.2 Stakeholders

Los *stakeholders* son las personas interesadas en el funcionamiento del sistema. Estas personas pueden estar involucradas en el desarrollo o ser usuarios finales del sistema. En el caso de estudio, teniendo en cuenta que se está aplicando un flujo de trabajo DevOps dentro de un proceso de desarrollo ágil de software, los *stakeholders* a tener en cuenta son los siguientes:

- **Usuario:** es el que utiliza la aplicación.
- **Dueño del producto (*Product Owner*):** es el cliente o puede representarlo. En el PFG este rol es desempeñado por las tutoras.
 - Se encarga de definir las características del producto, decidir las fechas de entregables y el contenido de los mismos.
 - Consolida las necesidades de los usuarios y otros *stakeholders* para crear una visión única y priorizada de los requisitos del sistema.
 - Mantiene y prioriza el *product backlog* y sus historias de usuario.

- Es responsable de los beneficios y coste del producto.
 - Debe respetar las estimaciones del equipo.
 - Debe mantener la comunicación con el equipo.
 - Acepta o rechaza el resultado del trabajo del equipo.
- **Líder del equipo ágil:** es el que guía el proceso y preserva su buena adopción. En el PFG este rol es desempeñado por el autor.
 - **Desarrollador:** es el responsable de crear el producto.
 - Ayuda a realizar las estimaciones.
 - Colabora con el *product owner* en la organización del *product backlog*.
 - Hace todo lo necesario para lograr el objetivo de la iteración.
 - Forma parte de un equipo auto-administrado y multifuncional.
 - En general, forma parte de un equipo de entre 5 y 9 miembros. En el PFG el equipo de desarrollo está formado únicamente por el autor.
 - **Operador:** es el responsable de poner el producto desarrollado en producción.
 - En un flujo de trabajo DevOps, este rol puede ser llevado a cabo por el desarrollador en mayor o menor medida según se determine. En el PFG este rol es llevado a cabo por el autor.

3.3 Diseño Conceptual

Un primer objetivo del caso de estudio es el desarrollo de una aplicación móvil (ver **OBJ1**, Sección 1.2). La Fig. 65 muestra el diseño conceptual de dicha aplicación cuyos principales aspectos técnicos se detallan a continuación:

- *Backend*: servicio API REST Java que gestione el acceso al servicio de almacenamiento de datos de Microsoft Azure. Este servicio puede ser utilizado por cualquier cliente REST, ya sea Android, iOS, etc. De esta forma se satisface el requisito de interoperabilidad. Además, el servicio va a utilizar la SDK de Microsoft para Java para acceder directamente al servicio de almacenamiento.
- Un servicio de aplicaciones API que aloje el API. Este servicio se configura para indicar, principalmente, que el API está implementado en la versión 8 de Java y que se ejecuta en un contenedor *Web* Tomcat 8.0.

- *Frontend*: cliente Android que acceda al servicio API REST. El cliente no va a utilizar la SDK de Microsoft para Android para acceder directamente al servicio de almacenamiento.

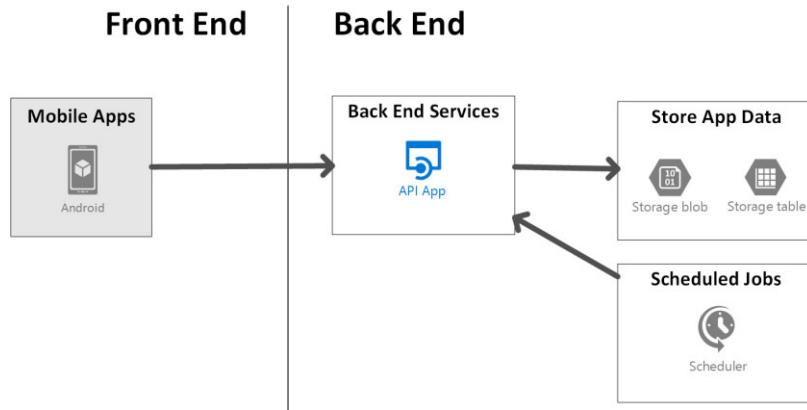


Fig. 65 Diseño conceptual del caso de estudio

- Un servicio de almacenamiento en la plataforma Microsoft Azure con replicación LRS que utilice un *blob* para guardar las fotos originales, otro *blob* para los *thumbnails* y una tabla para guardar entidades con datos referentes a los medios de transporte y relaciones a los correspondientes blobs con las fotos. LRS es el modo de replicación más barato y garantiza tres copias de los datos en una única CDP de una sola región. Es decir, solo se tendría peligro de pérdida de datos si dicho CDP se destruyese por completo. Por tanto, se considera suficiente para lo que se necesita. Para el uso estimado del recurso (menos de 1GB de datos y menos de 100.000 transacciones), el coste de este servicio con modo LRS de replicación es de 0 Euros.
- Un servicio de cuenta de automatización con el plan de tarifa gratuito que planifique la ejecución diaria de un *runbook* que invoca un servicio del API REST para que suba las fotos del *host* a la cuenta de almacenamiento. Para la invocación, es necesario la utilización de un credencial para conectarse a Microsoft Azure en forma no interactiva. Como el coste de este servicio (para el plan de tarifa gratuito) es gratuito los 500 primeros minutos de cada mes, teniendo en cuenta se ha planificado una vez al día y que la ejecución dura menos de 1 minuto se estima que no se va a incurrir en gastos ya que solo se va a gastar 30 minutos al mes.
- Una solución cuyo coste no sobrepase los 80 Euros/Mes. Por tanto, se puede gastar una media de 2.66 Euros/Día al mes (30 días) en servicios de Microsoft Azure.
- La ubicación ideal de todos los recursos de Microsoft Azure que se vayan a emplear en la solución es el CPD del “Oeste de Europa”. Esto se debe a que es la ubicación que ofrece Microsoft Azure más cercana al mercado objetivo de la aplicación, es decir, España. Si

posteriormente se desea implantar en otros mercados, solo habría que determinar el CPD deseado.

- Un plan del servicio de aplicaciones “S1 Estándar”. Este plan permite escalar el número de instancias de la aplicación hasta 10. En cualquier caso, siempre hay que estudiar si es suficiente y más barato subir las instancias o subir a otro plan superior con más memoria y procesadores. Este plan también permite agregar un espacio de implementación (de ensayo o *staging slot*), satisfaciendo así el requisito del usuario de que el pase a producción sea limpio y transparente. El coste de este plan es de 62,74 Euros/Mes para una instancia de la aplicación.

Finalmente, un segundo objetivo del caso de estudio es la gestión de todo el ciclo de vida de la aplicación (ALM) con VSTS (ver **OBJ2**, Sección 2.2). VSTS está totalmente integrada con la plataforma Microsoft Azure y ofrece un conjunto de servicios (control de versiones, integración continua, entrega continua, gestión de proyectos ágil, test automatizados) que permite aplicar las buenas prácticas y el *workflow* de DevOps. Una buena gestión va a repercutir en TTM y por ende en el coste, además de facilitar el requerimiento de entrega continua cada 2 semanas.

3.4 Arquitectura

Existen multitud de definiciones del concepto arquitectura. Una de las más conocidas es la siguiente:

La arquitectura software de un sistema es el conjunto de estructuras necesarias para razonar acerca del sistema, y que comprenden elementos software, relaciones entre ellos y las propiedades de ambos [10].

Todo sistema tiene una arquitectura, puede ser desordenada, pequeña, caótica pero siempre hay una estructura subyacente. La arquitectura existe independientemente de su descripción o especificación. Documentar una arquitectura es extremadamente importante ya que ayuda a la toma de decisiones y evolución de un sistema software, entre otras muchas funciones. Dicha documentación debe tener como objetivo minimizar el tiempo que tarda un nuevo desarrollador en ser productivo en la aplicación. Una de las formas más comunes de documentar una arquitectura es a través de vistas [39]. Y aunque no se pueda decir de forma taxativa que haya descripciones de arquitecturas buenas o malas, hay que reconocer que ciertas arquitecturas encajan mejor para el propósito para el que se han diseñado que otras y esto es clave para tener éxito en el desarrollo de un sistema software [10].

Hay que destacar que la noción de calidad es fundamental en la arquitectura del *software*, ya que una arquitectura del *software* es concebida para conocer las cualidades de un sistema lo antes posible. Algunos de estos atributos de calidad son observables en la ejecución

(rendimiento, seguridad, disponibilidad, usabilidad) y otros en el desarrollo (modificabilidad, portabilidad, reusabilidad, etc.). Por tanto, la arquitectura se debe evaluar en su habilidad de cumplir los atributos de calidad para los que se ha diseñado [40].

Además, el término arquitectura suele asociarse con metodologías pesadas con meses de diseño por adelantado sin escribir nada de código, parece que esto no encaja con las metodologías ágiles. Sin embargo, es posible tener un desarrollo guiado por la arquitectura y trabajar en un contexto ágil [31] como se verá a lo largo de este capítulo.

Los requisitos del sistema, las decisiones de diseño y las vistas arquitectónicas son puntos imprescindibles para describir y documentar la arquitectura. Estos se describen en las siguientes subsecciones.

3.4.1 Requisitos del sistema

Los sistemas de *software* se construyen para satisfacer los objetivos de negocio de las organizaciones. En general, los objetivos de negocio suelen estar orientados hacia la consecución de beneficios aunque no siempre. En nuestro caso de estudio el objetivo de negocio es principalmente el estudio y aprendizaje [10]. Los objetivos de negocio son de interés para las arquitecturas porque son los precursores de los requisitos del sistema cuyo logro señalará el éxito de dicha arquitectura. Es decir, las arquitecturas existen para construir sistemas que satisfagan requisitos pero hay que interesarse especialmente por aquellos que impactan en la arquitectura. Por tanto, no es de esperar que se construya una buena arquitectura sin conocer los requisitos [10]. En este PFG los requisitos se extraen de la descripción del caso de estudio efectuada en la Sección 3.1 y se categorizan de la siguiente manera:

- Requisitos funcionales
- Requisitos sobre atributos de calidad
- Restricciones

Posteriormente se va a ver cada uno de ellos en detalle, pero es importante destacar que los sistemas muchas veces son rediseñados no porque su funcionalidad sea deficiente sino porque su calidad es deficientes (son difíciles de mantener, escalar, portar, son muy lentos o han sido *hackeados*) [10].

En este PFG se ha seguido una metodología ágil. En consecuencia, los requisitos se describen en forma de HU (ver Fig. 66). Para la gestión del *product backlog* (ver Fig. 67), *board* (ver Fig. 68), *iterations/Sprints* (ver Fig. 69), y todo lo que tiene que ver con un proceso ágil de desarrollo se ha utilizado VSTS.

USER STORY 178*

178 Visualizar lista thumbnails

New alumno PFG2016 0

Area PFGAlex2016 Iteration PFGAlex2016\Iteration 4

Updated by alumno PFG2016 just now

Add Tag

Description

B I U A Ø Ø Æ Æ Í Í Ð Ð Ñ Ñ Como usuario, quiero poder visualizar una lista de thumbnails siempre que pulse el botón de refresco situado en la parte inferior derecha de la ventana del dispositivo móvil

Status

Reason New

Development

Development hasn't started on this item. [Create a new branch](#)

Classification

Value area Business

Planning

Story Points

Priority 2

Risk

Acceptance Criteria

B I U A Ø Ø Æ Æ Í Í Ð Ð Ñ Ñ

1. Se pulsa el botón de refrescar lista de thumbnails del dispositivo móvil
2. Se muestra lista de thumbnails

Fig. 66 Desarrollo ágil de sw a través de VSTS: definición HU

https://alumnopfg2016.visualstudio.com/DefaultCollection/PFGAlex2016/_backlogs/iteration/Iteration%206

Visual Studio Team Services / PFGAlex2016

HOME CODE WORK BUILD TEST RELEASE

Search work items alumno PFG2016

Backlogs Queries

PFGAlex2016 Team Iteration 6 4 April - 15 April 12 work days

Work details On

Backlog Board Capacity

New + Create query Column options |

Type User Story

Title Refactorizar android

Add

Title	State	Assigned To	Remainder
Creacion de releases	Active	alumno PFG20...	
Refinar visualización de la lista y del detalle	Closed	alumno PFG20...	
Refactorizar android	Closed	alumno PFG20...	
Problemas fotos muy grandes -> TransactionTooLargeException	Closed	alumno PFG20...	
Memory leaks	Closed	alumno PFG20...	
Icono de la aplicación y normalización de nombres (inglés)	Closed	alumno PFG20...	
Refactorizar tomcat	Active	alumno PFG20...	
Test android	Active	alumno PFG20...	
Test tomcat	Active	alumno PFG20...	

Epics Features Stories

Past Iteration 1 Iteration 2 Iteration 3 Iteration 4 Iteration 5 Current Iteration 6

Recycle Bin

Fig. 67 Backlog Agile desde donde se gestionan las HU en la pestaña WORK de VSTS

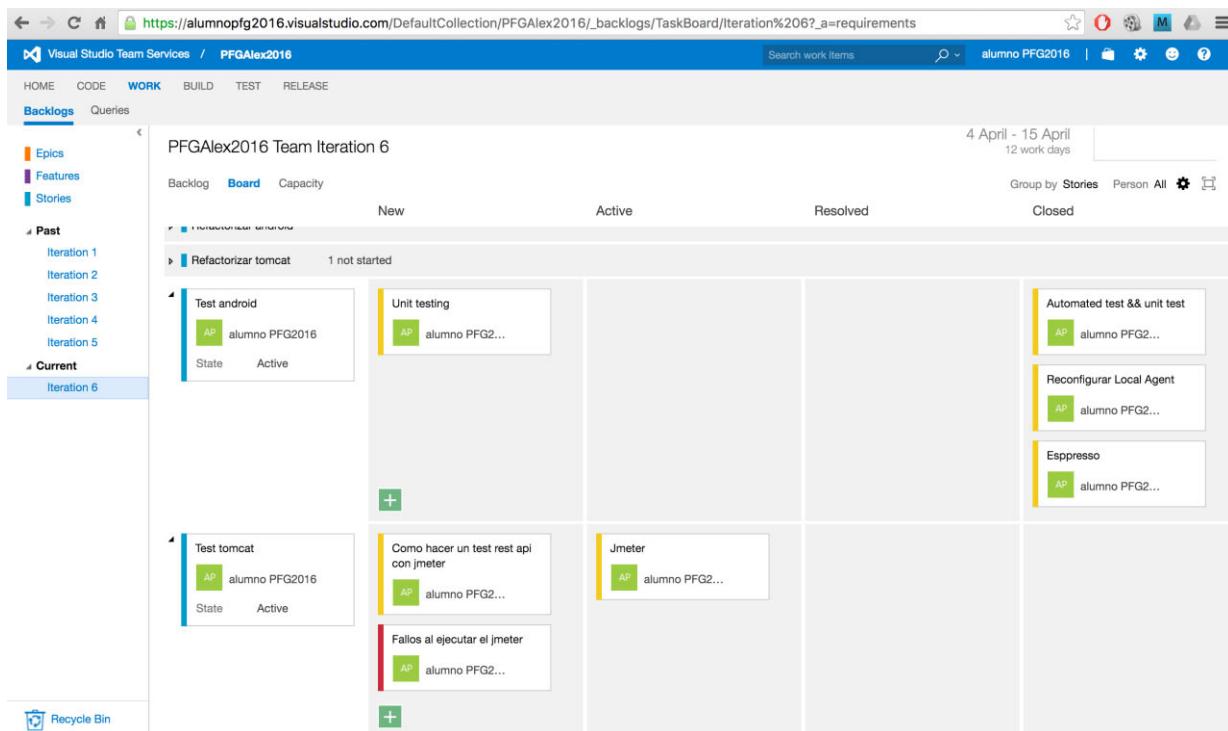


Fig. 68 Board Agile en VSTS en la pestaña WORK de VSTS

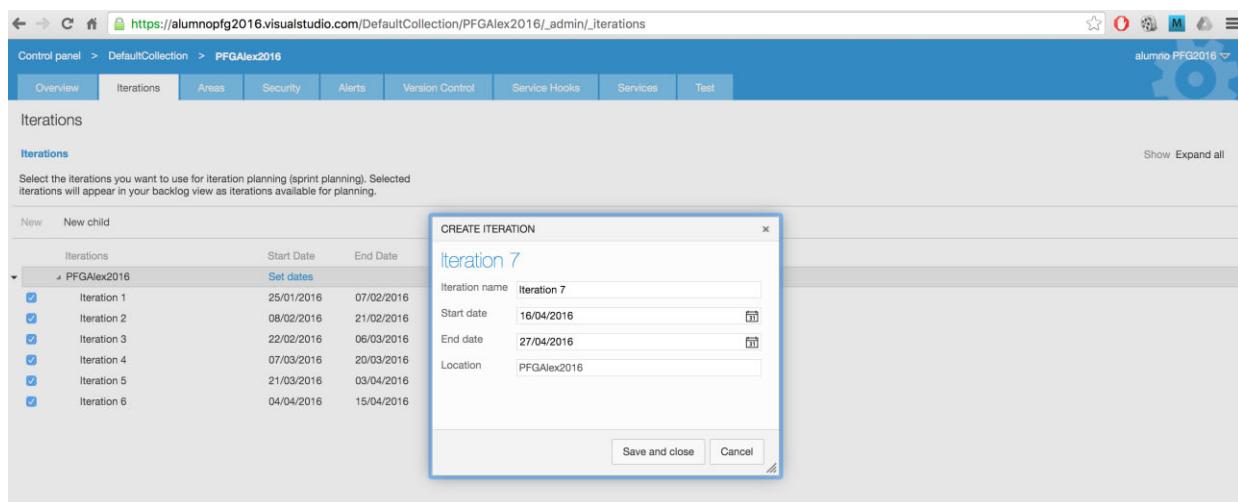


Fig. 69 Iterations/Sprints Agile en el panel de control de VSTS

3.4.1.1 Requisitos funcionales

Los requisitos funcionales determinan lo que el sistema debe hacer y cómo se debe comportar o reaccionar ante un estímulo en tiempo de ejecución [10]. La funcionalidad no es lo único que determina la arquitectura. Como ya se ha comentado anteriormente, los sistemas

muchas veces son rediseñados no porque su funcionalidad sea deficiente, sino porque sus atributos de calidad son deficientes, el sistema tiene falta de mantenibilidad, escalabilidad, portabilidad, rendimiento o seguridad [10].

Para la consecución de este PFG se han extraído un conjunto de requisitos funcionales del caso de estudio descrito en la Sección 3.1. A continuación se enumeran dichos requisitos funcionales:

HU1. Como usuario, quiero poder visualizar una lista de *thumbnails* (imágenes en miniatura) siempre que pulse el botón de refresco situado en la parte inferior derecha de la ventana del dispositivo móvil.

HU2. Como usuario, quiero poder visualizar la foto original de un thumbnail siempre que pinche en dicho thumbnail

HU3. Como administrador, quiero que el sistema cargue diariamente en el almacén de datos en modo *batch* las fotos existentes en una jerarquía de directorios

3.4.1.2 Requisitos sobre atributos de calidad

Los atributos de calidad son aquellas cualidades o propiedades medibles que debe satisfacer el producto software. Son requisitos no funcionales en tanto que no afectan a la funcionalidad del producto, pero sí a su desempeño y a la calidad del mismo [10]. Los atributos de calidad deben estar alineados con los objetivos de negocio y su priorización es la clave de la estrategia de diseño de una arquitectura dirigida por atributos, en la cual dichos atributos condicionan las decisiones de diseño que se van a adoptar [10].

Por otra parte, es obvio que una arquitectura que pudiese ser óptima en el cumplimiento de los máximos estándares de calidad sería idónea pero hay que ser conscientes que el tiempo y el dinero no son infinitos. Además, hay que tener en cuenta que en sistemas complejos, es difícil aislar la consecución de los atributos de calidad y muchas veces conseguir uno puede tener efectos tanto positivos como negativos en otros y viceversa [10].

Extraídos del caso de estudio descrito en la Sección 3.1, en el caso de estudio la solución arquitectónica se va a diseñar dando prioridad a los siguientes atributos de calidad, los cuales serán desglosados en requisitos:

- **TTM:** representa el tiempo que se necesita para desarrollar un producto y lanzarlo al mercado. A este respecto, se han extraído los siguientes requisitos del caso de estudio:

RT1. El sistema debe estar en el mercado en 4 meses.

- **Coste y Beneficios:** representa la relación entre los beneficios económicos del sistema y los costes necesarios para su desarrollo e implantación. A este respecto, se han extraído los siguientes requisitos del caso de estudio:

RC1. No se tiene que hacer ninguna inversión inicial.

RC2. El coste de mantenimiento de la aplicación no debe superar los 80 euros mensuales.

RC3. No tiene ingresos. Es una inversión a fondo perdido para estudiar el mercado.

- **Mantenibilidad:** es la facilidad con la que un sistema o componente puede ser modificado para corregir los fallos detectados, mejorar el rendimiento o adaptarlo a cambios del entorno. A este respecto, se han extraído los siguientes requisitos del caso de estudio:

RM1. Tiene que ser posible hacer modificaciones frecuentes y fácilmente.

RM2. Tiene que ser posible desplegar de forma limpia y transparente para el usuario siempre que se deseé.

- **Interoperabilidad:** es la capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información que se ha intercambiado. A este respecto, se han extraído los siguientes requisitos del caso de estudio:

RI1. Tiene que ser capaz de operar en sistemas heterogéneos.

- **Rendimiento y Escalabilidad:** el rendimiento es una indicación del grado de reacción de un sistema a ejecutar cualquier acción dentro de un intervalo de tiempo. Se puede medir en términos de latencia o de rendimiento. Latencia es el tiempo que le lleva responder a un evento. Rendimiento es el número de eventos que se llevan a cabo en un intervalo de tiempo. Por otra parte, la escalabilidad es la capacidad de un sistema para manejar incrementos de carga sin impactar en el rendimiento aumentando los recursos disponibles si es necesario. El rendimiento de una aplicación puede afectar directamente su escalabilidad y una falta de escalabilidad puede afectar al rendimiento. Si se mejora el rendimiento de una aplicación se suele mejorar su escalabilidad ya que se reduce la probabilidad de contención por recursos compartidos. Las aplicaciones *Cloud* suelen tener cargas variables y picos en actividad. Predecirlos, especialmente en un escenario de tenencia múltiple, es casi imposible. En su lugar, las aplicaciones deben ser capaces de escalar tanto para añadir recursos como para decrementar recursos en función de los picos de demanda. La escalabilidad no tiene solo que ver con instancias de computación sino con otros elementos tales como almacenamiento de datos, etc. A este respecto, se han extraído los siguientes requisitos del caso de estudio:

RE1. No existen impedimentos al crecimiento del número de fotos de medios de transporte. En cualquier caso, se estima que vaya a tener menos de 1TB.

RE2. No existen impedimentos al crecimiento del número de accesos a las fotos de medios de transporte. En cualquier caso, se estima que vaya a tener menos de 100.000 accesos al mes.

- **Usabilidad:** los interfaces de usuario se deben diseñar con el usuario en mente para que sean intuitivos de utilizar, puedan ser internacionalizados fácilmente, faciliten el acceso a usuarios discapacitados y proporcionen una buena experiencia en general. A este respecto, se han extraído los siguientes requisitos del caso de estudio:

RU1. Diseño de la interfaz de usuario multipanel.

RU2. La aplicación no se debe quedar bloqueada al efectuar las consultas.

- **Resistencia:** es la habilidad de un sistema de manejar y recuperarse de errores elegantemente. En la nube, donde las aplicaciones son multitenedor, utilizan servicios compartidos de la plataforma, compiten por recursos y ancho de banda y se comunican en Internet existe una alta probabilidad que sucedan errores tanto permanentes como transitorios. Detectar fallos y recuperarse rápida y eficientemente es necesario para conseguir que la aplicación sea resistente. A este respecto, se han extraído los siguientes requisitos del caso de estudio:

RR1. Si sucede cualquier tipo de fallo achacable a la red, la aplicación tiene que mostrar un mensaje significativo, seguir operativa y debe permitir reintentar la operación (refrescar) tantas veces sea necesario a través de un botón.

- **Seguridad:** es la capacidad del sistema de prevenir acciones no permitidas ya sean maliciosas o accidentales, y prevenir la divulgación o pérdida de información. Las aplicaciones *Cloud* tienen especial riesgo ya que están expuestas a Internet, fuera de los límites de confianza una red local, frecuentemente están abiertas al público y pueden servir a usuarios de poca confianza. Las aplicaciones se deben diseñar y desplegar en una forma que se las proteja de ataques maliciosos, se restrinja el acceso solo a usuarios autorizados y se protejan los datos sensibles. A este respecto, se han extraído los siguientes requisitos del caso de estudio:

RS1. No se requiere autorización ni autenticación para utilizar la aplicación.

- **Disponibilidad:** define la proporción de tiempo que el sistema es funcional y está trabajando. Se ve afectada por errores del sistema, problemas de infraestructura, ataques malignos y cargas del sistema. Habitualmente se mide como un porcentaje del tiempo de uso. Las aplicaciones *Cloud* generalmente proporcionan a sus usuarios un SLA (*Service Level Agreement*)¹⁸ para que diseñen e implementen las aplicaciones en una forma que maximice

¹⁸ SLA (*Service Level Agreement*) o acuerdo de nivel de servicio es un contrato escrito entre un proveedor de servicio y su cliente con objeto de fijar el nivel acordado para la calidad de dicho servicio en términos del nivel de calidad del servicio, en aspectos tales como tiempo de respuesta, disponibilidad horaria, documentación disponible, personal asignado al servicio, etc.

la disponibilidad. A este respecto, se han extraído los siguientes requisitos del caso de estudio:

RD1. Debe estar disponible principalmente desde España.

3.4.1.3 Restricciones

Una restricción es una decisión de diseño que ya ha sido tomada y que se debe aceptar como tal. A este respecto, se han extraído las siguientes restricciones del caso de estudio descrito en la Sección 3.1:

- C1.** El *backend* debe utilizar el máximo número de servicios Azure que sea posible.
- C2.** El *backend* debe ser implementado en Java.
- C3.** El *frontend* debe ser implementado en Android.

3.4.2 Decisiones de diseño

Una arquitectura es el conjunto de las principales decisiones de diseño hechas en un sistema. El objetivo es que dichas decisiones sirvan para diseñar un sistema que cumpla con los atributos de calidad especificados. Hay que tener en cuenta, sin embargo, que el desarrollo de una aplicación hospedada en la nube no es sencillo. Al igual que cualquier otro sistema distribuido conlleva numerosos problemas. Afortunadamente, para ayudarnos a solucionar gran parte de estos problemas, podemos apoyarnos en patrones arquitectónicos [14].

Por otra parte, es muy importante tener en cuenta que, en una arquitectura, siempre deberíamos poder justificar todas las decisiones ya que si no se puede a lo mejor es que dicha decisión de diseño sobra.

En el caso de estudio se han tomado una serie de decisiones de diseño para cumplir con sus atributos de calidad (ver Sección 3.4.1.2). Estas decisiones se describen en las siguientes secciones:

3.4.2.1 Estilo arquitectónico REST

Para cumplir con el requisito **RI1** de interoperabilidad del caso de estudio y facilitar en la medida de lo posible el cumplimiento de los requisitos **RE1** y **RE2** de escalabilidad, se aplica el estilo arquitectónico REST.

A continuación se describe cómo se efectúa una interacción típica de la aplicación del caso de estudio entre un cliente REST en Android y un servicio RESTful en Java (ver Fig. 70):

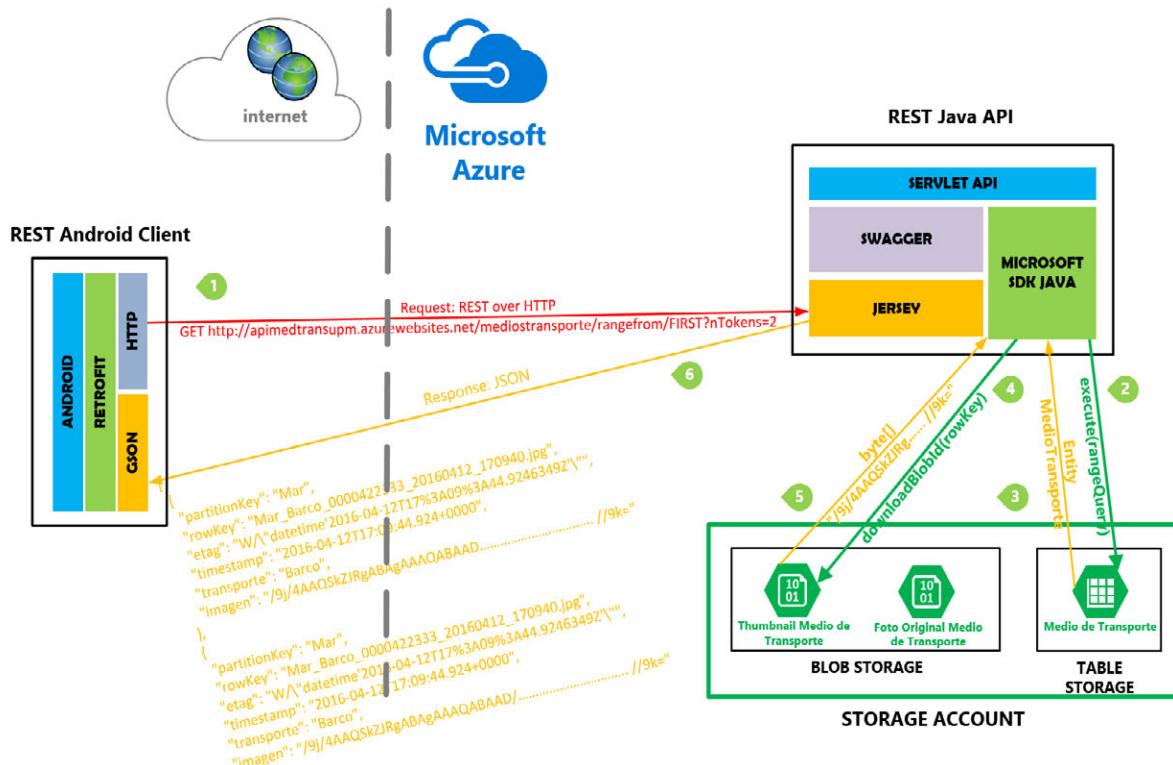


Fig. 70 Interacción típica de la aplicación entre cliente REST y servicio RESTful

- 1 Desde un cliente REST en Android se pretende obtener los 2 primeros *thumbnails* del almacén de datos de Microsoft Azure. Es por ello que dicho cliente efectúa la petición HTTP GET "`http://apimedtransupm.azurewebsites.net/mediotransporte/rangefrom/FIRST?nTokens=2`". El cliente, para facilitar la implementación de las peticiones REST HTTP, utiliza la biblioteca Retrofit (ver Sección 4.2.2). El servicio del API, para facilitar la implementación de la funciones que van a atender a las peticiones HTTP, hace uso de las anotaciones proporcionadas por la especificación JAX-RS e implementadas en este PFG por la biblioteca Jersey (ver Sección 4.1.2).

2 **3** La función que implementa la petición hecha por el cliente, determina que debe ir a buscar los datos a una tabla dentro de la cuenta de almacenamiento de Azure. Para acceder a dicha tabla, hace uso de la biblioteca Microsoft SDK para Java.

4 **5** A partir de la entidad recuperada de la tabla, se obtiene la referencia al *blob* donde se guarda el *thumbnail* que corresponde a dicha entidad. Por tanto, mediante el uso de la biblioteca Microsoft SDK para Java, se accede al *blob* y se obtiene el *thumbnail*.

6 Los datos se pasan de modelo (el modelo se puede consultar en la Fig. 80 de la Sección 4.1.2) a JSON automáticamente por Jersey y se envían al cliente. El cliente hace la operación inversa, es decir, los pasa de JSON a modelo mediante el convertidor GSON de Retrofit (ver Sección 4.2.2). Respecto a las imágenes, en el modelo Java se han definido como *array* de *bytes* y en el modelo Android como *String* (hay que pasarlas a *Bitmap* para visualizarlas en el móvil).

3.4.2.2 Táctica caché cliente

Entre las restricciones que debe cumplir un servicio para ser considerado RESTful se encuentra la que dice que debe ser “cacheable” (ver Sección 2.3.5). En la práctica, esta restricción no se suele tener mucho en cuenta y, sin embargo, ayuda a implementar servicios más rápidos, escalables y tolerantes a fallos.

En nuestro caso de estudio la caché va en el cliente. Lo normal es que la caché este en el cliente pero nada impide que se refuerce con cachés en el servidor. El objetivo de que la caché esté en el cliente es evitar al cliente ir al servidor a buscar las fotos originales de los *thumbnails* seleccionados para mejorar su rendimiento. Esta caché tiene capacidad para 5 fotos, descartándose las más antiguas cuando sea necesario. Esta versión de caché no tiene en cuenta ni la capacidad del dispositivo en que está instalada la aplicación ni otras consideraciones que se pudieran tener en cuenta para efectuar cacheos más eficientes. Es decir, tanto si el dispositivo tuviese capacidad para 1000 como para 2 fotos siempre va a preservar 5.

3.4.2.3 Patrón particionado

En el caso de estudio, el esquema de base de datos es muy simple. En un primer análisis, constaría de una única tabla donde se guardarían las fotos originales y sus *thumbnails* junto con el tipo de medio (tierra, mar, aire) y transporte (avión, barco, coche, tren, etc.) al que corresponden (ver Fig. 71). Dicha tabla almacena datos de muy diversa naturaleza. Algunas columnas almacenan cadenas de caracteres, las cuales se pueden almacenar eficientemente en una tabla. Otras, sin embargo, almacenan *arrays* de *bytes* de los ficheros de imágenes.

Con esta solución se encuentran con dos potenciales problemas. En primer lugar, esto sería bastante caro si el almacén de datos fuese una base de datos relacional en la nube. En segundo lugar, esto no sería posible en la mayoría de los casos, ya que en una tabla de una cuenta de almacenamiento de Microsoft Azure, solo se permite almacenar entidades de un tamaño de 1MB.

clave	medio	transporte	thumbnail	foto
1	aire	avion	3KB	3MB
2	mar	barco	2KB	4MB
3	tierra	tren	2KB	5MB
4	tierra	coche	1KB	2MB

Fig. 71 Almacenamiento sin particionar

Para solucionar este problema, se partitionan los datos verticalmente (ver Fig. 72), de forma que se eligen almacenes de datos más apropiados para cada columna de la tabla:

- Las columnas de tipo cadena de caracteres se guardan en un almacenamiento de tipo tabla.
- Las columnas de tipo *array of bytes* se guardan en un almacenamiento de tipo *blob*.

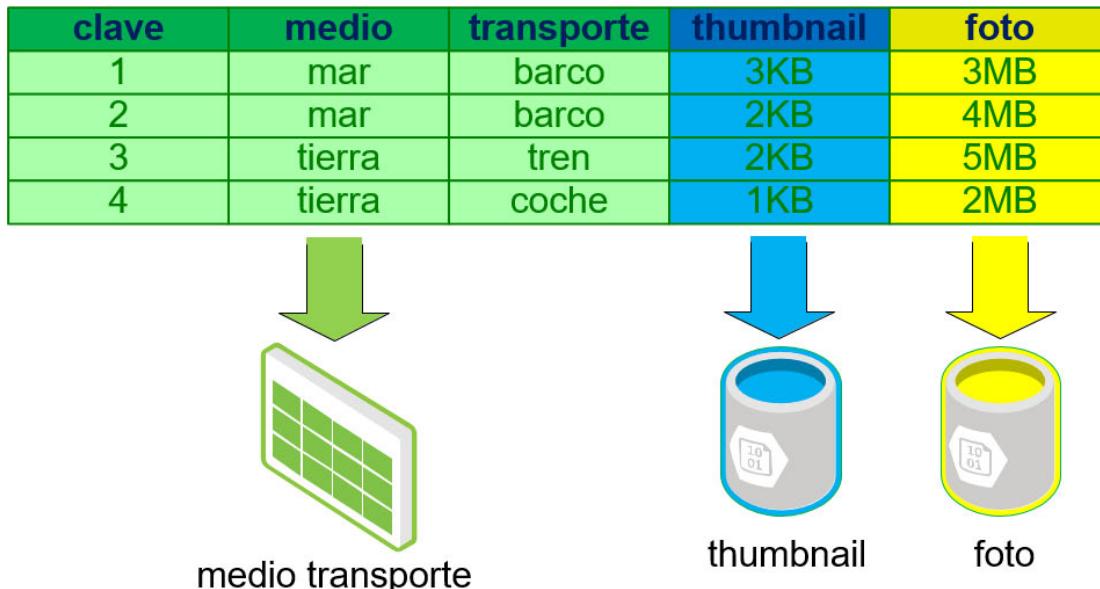


Fig. 72 Partición vertical

En los almacenamientos de tipo tabla se almacenan entidades. Cada entidad define su propia colección de propiedades que no tiene que ser siempre la misma. Cada propiedad es un par clave-valor que se define con su nombre, valor y el tipo de datos del valor. Las entidades deben definir las siguientes propiedades, además de las suyas propias:

- **PartitionKey**: almacena valores de cadena que identifican la partición a la que pertenece una entidad. Esto significa que los valores con la misma *PartitionKey* pertenecen a la misma partición.
- **RowKey**: almacena valores de cadena que identifican de forma única entidades incluidas en cada partición.
- **Timestamp**: es un valor *DateTime* que indica la última vez que la entidad se modificó.

En el caso de estudio, en el paso a tabla de Azure, la columna “medio” se ha convertido en *PartitionKey* y como *RowKey* se ha puesto una referencia a la foto en el *blob*. Por tanto, el tipo de medio (mar, tierra, aire, etc.) determina el particionado horizontal (ver Fig. 73).

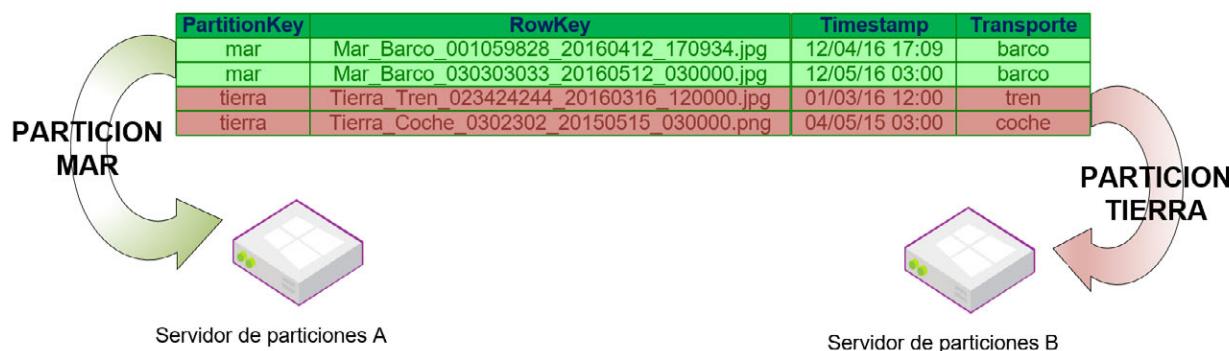


Fig. 73 Partición horizontal

En la figura anterior, si hay muchas solicitudes para la partición “mar”, es posible que el servidor A pase a estar demasiado activo. Para aumentar el rendimiento del servidor, el sistema de almacenamiento equilibra la carga de las particiones en otros servidores. El resultado es que el tráfico se distribuye entre otros muchos servidores. Para un equilibrio de carga de tráfico óptimo, se deben usar más particiones porque eso permitirá que el servicio tabla de Microsoft Azure distribuya las particiones en más servidores de particiones.

3.4.2.4 Patrón vista materializada

En el caso de estudio, la carga de fotos en los almacenes de datos es mediante un proceso planificado que lee las fotos de una jerarquía de directorios situada en el directorio “/site/wwwroot” de la máquina donde se aloja el API, al que se ha llamado “*apimedtrans*”. (ver Fig. 74). Durante esta carga, se implementa el patrón vista materializada. Las fotos originales

se guardan en un *blob* llamado "**foto**" y, a su vez, se transforman en *thumbnails*, los cuales se guardan en el *blob* "**thumbnail**". Esto mejora el rendimiento de la aplicación enormemente cuando el cliente móvil consulta las fotos. Las fotos ya están convertidas en *thumbnails* con lo que ya no hay que perder tiempo convirtiéndolas. Además, no pasa nada porque se borre este almacén. Se puede volver a generar a partir de las fotos originales.

A continuación se describe cómo se efectúa el proceso de carga de las fotos en los almacenes de datos y cómo se implementa el patrón vista materializada con más detalle (ver Fig. 74):

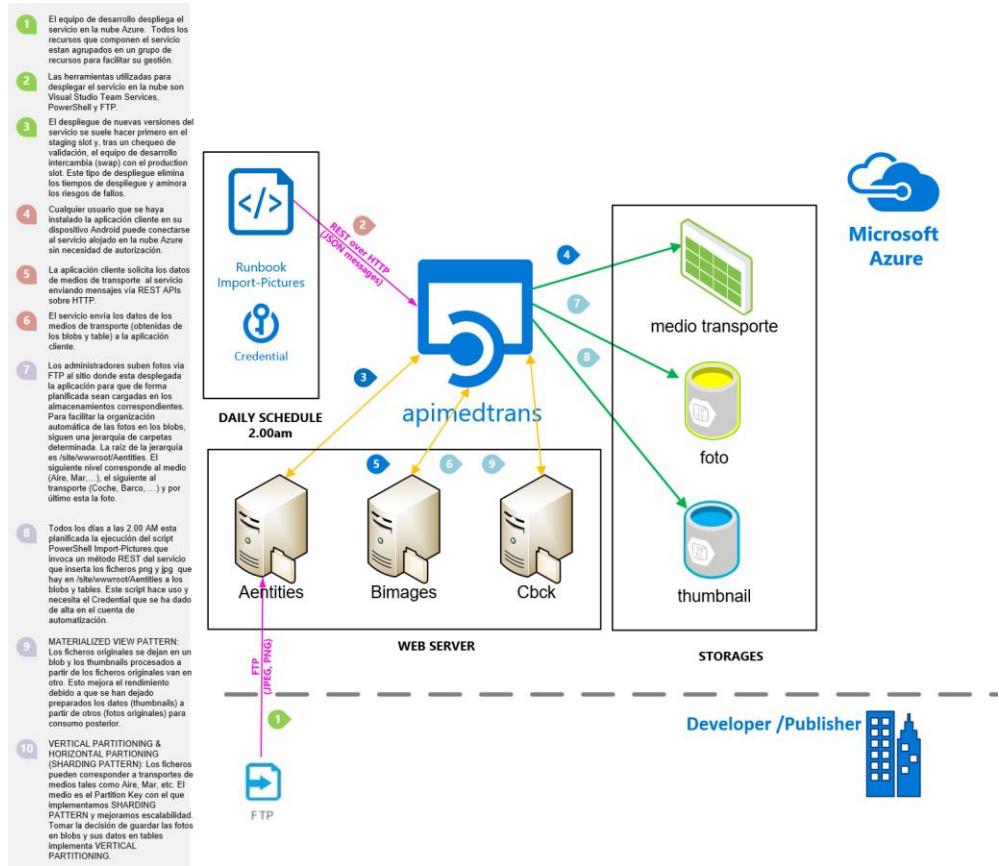


Fig. 74 Vista materializada

- 1 Un administrador del sistema se conecta vía FTP al servidor Web¹⁹ y copia los ficheros con las fotos originales en formato png o jpg en una estructura de directorios organizada

¹⁹ <ftp://waws-prod-am2-065.ftp.azurewebsites.windows>

jerárquicamente. En la raíz de esta jerarquía se encuentra el directorio “*Aentities*”. En el siguiente nivel en profundidad se encuentran los medios (que van a corresponder a las particiones en la tabla). En el último nivel de profundidad, las hojas, están los tipos de transporte (que se van a corresponder con el campo transporte) que es, precisamente, donde se copian las fotos.

2 Todos los días a las 2.00 PM está planificado el lanzamiento de un *script PowerShell*, de nombre “*Import-Pictures*”, que invoca un método del servicio API REST “*medtrans*” que está desplegado en la nube en la suscripción de Azure.

3 **4** **5** El API REST “*medtrans*” copia todas las fotos que existan en la jerarquía de directorios “*medtrans_Entities*” en la tabla “*medtrans*” y las mueve a “*medtrans_Bimages*”. Realmente registra las fotos como entidades con las siguientes propiedades: el nombre del archivo (*RowKey*), el medio (*PartitionKey*), el tipo de transporte (*Transporte*) y la fecha y hora del registro (*Timestamp*). Los arrays de bytes que forman una foto se guardan en un paso posterior.

6 **7** **8** **9** En un último paso, el API REST “*medtrans*” copia los arrays de bytes que componen las fotos en un *blob* y los de los *thumbnails* procesados a partir de los ficheros originales en otro. Esto mejora el rendimiento debido a que se han dejado preparados los datos (*thumbnails*) a partir de otros (fotos originales) para consumo posterior. Finalmente se mueven todas las fotos al directorio “*medtrans_Cbck*”.

3.4.2.5 Patrón asíncrono

En lo que respecta al caso de estudio, tanto la petición del cliente para obtener los *thumbnails* como la foto original es efectuada en modo asíncrono. De esta forma el dispositivo no pierde su capacidad de respuesta y cubre los requisitos de usabilidad impuestos.

3.4.2.6 Táctica manejo de excepciones

En nuestro caso de estudio, se gestionan las excepciones y, en el caso del cliente, por ejemplo, donde es habitual que se produzcan fallos transitorios de red, se informa del error al usuario y se le da la posibilidad de volver a intentarlo si así lo desea.

3.4.2.7 Conclusiones

En la solución arquitectónica del caso de estudio, se han adoptado una serie de decisiones de diseño para cumplir los atributos de calidad especificados. En la Fig. 75 se muestra una tabla donde se traza cada atributo de calidad con la decisión de diseño arquitectónico que se adopta para cumplirlo.

Atributo Calidad	TTM	Coste y Beneficio	Mantenibilidad	Interoperabilidad	Rendimiento y Escalabilidad	Usabilidad	Resistencia	Seguridad	Disponibilidad
Tácticas y Requisitos Patrones	RT1	RC1 RC2 RC3	RM1 RM2	RI1	RE1 RE2	RU1 RU2	RR1	RS1	RD1
Agile architecting / DevOps	✓	✓ ✓ ✓ ✓ ✓							
Estilo arquitectónico REST				✓	✓ ✓				
Patrón particionado					✓ ✓				
Patrón vista materializada					✓ ✓				
Caché cliente					✓ ✓	✓ ✓	✓		✓
Patrón asíncrono						✓ ✓	✓ ✓		
Táctica manejo de excepciones						✓ ✓	✓ ✓		✓

Fig. 75 Trazabilidad entre decisiones de diseño y requisitos

3.4.3 Vistas Arquitectónicas

Una de las formas más comunes de documentar una arquitectura es a través de vistas. Una vista es la representación de un conjunto de estructuras y las relaciones entre ellas, documentada de acuerdo a una plantilla en una notación determinada, con algún propósito determinado, a un apropiado nivel de detalle y para algún *stakeholder* en particular. Los propósitos generalmente tienen que ver con la construcción, análisis y mantenimiento del sistema así como formación de los nuevos *stakeholders* acerca del sistema [10].

Reseñar que toda documentación debe tener una intención y una audiencia determinada en mente. **Uno de los principios fundamentales acerca de la documentación técnica es "Escribir para el lector". Esto significa que hay que entender quien leerá la documentación y como la va a utilizar.**

Las vistas facilitan la comunicación entre los diversos miembros del equipo y la comprensión de sistemas complejos. Por ello, la descripción de la arquitectura del caso de estudio se va a basar en las dos siguientes vistas:

1. Vista de interrelaciones entre los servicios Azure.
2. Vista de flujo de trabajo DevOps.

3.4.3.1 *Vista de interrelaciones entre los servicios Azure*

Esta vista va dirigida a los desarrolladores y operadores. Para su elaboración no se ha utilizado ninguna notación especial, simplemente diagramas de cajas y líneas. En ella se han combinado vistas de los siguientes tipos:

- Modulares: las cajas son componentes del sistema (clases, capas, etc., en definitiva, unidades de implementación) y las líneas denotan alguna relación entre los mismos. Dan una visión estática del sistema. Ayudan a conocer la funcionalidad asignada a cada módulo, dependencias entre módulos, de qué se componen los módulos, si sus módulos están estructurados como capas (siendo capas servicios a través de un interfaz y sabiendo que las capas solo se pueden utilizar las capas adyacentes). Útiles para evaluar cualidades como modificabilidad y portabilidad.
- Componentes y conectores (C&C): las cajas son componentes del sistema en tiempo de ejecución (servicios, clientes, etc.) y las líneas son conectores (vehículos de comunicación entre componentes como una llamada-retorno, etc.). Dan una visión dinámica del sistema. Ayudan a conocer los principales componentes en ejecución del sistema, los principales almacenes de datos. Útiles para evaluar cualidades como rendimiento y disponibilidad.
- Asignación: define como los elementos de C&C y los módulos se distribuyen entre cosas que no son software (*hardware*, equipos, sistemas de ficheros, etc.). Ayudan a conocer en qué

procesador se ejecuta cada elemento de *software* y que rutas de comunicación se utilizan (despliegue), en qué directorios o ficheros esta cada elemento guardado (implementación), quién está haciendo qué (asignación de trabajo). Útiles para evaluar cualidades como rendimiento, seguridad y disponibilidad.

El propósito de esta vista es mostrar cómo interactúan y de qué se componen las tres capas que componen el universo de la aplicación: Internet, Microsoft Azure y Desarrollo/Operaciones (ver Fig. 76):

- Por un lado sirve a los operadores para ver qué compone la parte *backend* de la aplicación y cómo se despliega en Azure. Principalmente, se compone del **grupo de recursos** (*"Resource Group"*) *"ARMmedtrans"* que mediante VSTS se despliega en el CPD ubicado en *"Europa Occidental"* (*"West Europe"*). También se pone de manifiesto en la vista que existen otros mecanismos de despliegue aparte de VSTS. Se pueden desplegar tanto los servicios Microsoft Azure como el API Java mediante *scripts* PowerShell o, si solo se quiere desplegar el API Java, mediante FTP copiando su correspondiente WAR²⁰ en la carpeta *"%site/wwwroot/webapps"* del servidor donde se encuentra Tomcat.
- Por otro lado sirve a los desarrolladores para ver que el grupo de recursos *"ARMmedtrans"* se compone de los recursos que se enumeran a continuación (todos ellos ubicados en el CPD de *"Europa Occidental"*):
 - El plan del servicio de aplicaciones (*"Service Plan"*) *"servplan"* de tipo *"S1"* cuyo coste mensual es de 62.74 Euros. La razón por la cual se elige esta opción, en vez de otras opciones gratuitas, es porque permite desplegar la aplicación en un espacio de implementación (*"staging slot"*) antes de hacerlo en el espacio de producción (*"live API"*). De esta forma, cuando ya se confirma que el despliegue en dicho espacio de implementación ha sido un éxito, se lleva a cabo el intercambio (*"swap"*) con el espacio de producción. De esta forma, se eliminan los tiempos de despliegue y se aminoran los riesgos de fallos.

²⁰ WAR (*Web application ARchive*) es un fichero JAR que se utiliza para distribuir una colección de JSPs, Servlets Java, Clases Java, ficheros XML, páginas estáticas HTML y otros recursos que junto constituyen una aplicación *Web*.

JAR (*Java ARchive*): es un tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java. Los archivos JAR están comprimidos con el formato ZIP y cambiada su extensión a .jar.

- El servicio de aplicaciones "***API app***" cuyo contenedor *Web* es Tomcat y aloja a la API REST "***apimedtrans***" desarrollada en Java.
- El servicio/cuenta de almacenamiento ("*Storage Account*") "***stormedtrans***" con replicación LRS (es el modo de replicación más barato y garantiza tres copias de los datos en una única instalación de una sola región). Para mejorar la escalabilidad de la capa de datos y el rendimiento de la aplicación, se ha aplicado el patrón "**particionado de datos**" (ver Sección 3.4.2.3) en base a lo cual se han creado los tipos de almacenamiento que se enumeran a continuación:
 - Un *blob* "***containermedtransoriginales***" para guardar las fotos originales.
 - Un *blob* "***containermedtransthumbnails***" para los *thumbnails*.
 - Una tabla "***tablemedtrans***" para guardar entidades con datos referentes a los medios de transporte y relaciones a los correspondientes blobs con las fotos.
- Un servicio/cuenta de automatización ("*Automation Account*") "***AutAccount***" que tiene define un planificación "***DailySchedule0200***" diaria a las 2:00AM y que ejecuta el *runbook* "***Import-Pictures***". Este *script* PowerShell invoca un método HTTP del servicio RESTful "***apimedtrans***" que efectúa la carga, en los almacenes de datos de Microsoft Azure, de las fotos ubicadas en una jerarquía de ficheros determinada. En el mismo proceso, las fotos son convertidas y almacenadas como *thumbnails* también. De **esta forma, se aplica el patrón "vista materializada"** (ver Sección 3.4.2.4) que va a mejorar el rendimiento de la aplicación ya que los *thumbnails* que van a ser necesarios para su funcionamiento son calculados y almacenados previamente. En la ejecución del *runbook* para poder invocar los métodos del API es necesario conectarse a la suscripción de Azure. Es por ello que se hace uso del activo "***AutCredentials***" de tipo "***Credential***" que va a posibilitar dicha conexión de forma interactiva.
- Por último permite a todos los *stakeholders* ver como interactúa a través de Internet y mediante peticiones HTTP el *frontend* (móviles Android) con el *backend* (servicio RESTful) alojado en Azure.

MedTrans Architecture

MedTransUPM gestiona una enorme colección de fotos de medios de transporte de cualquier época y que ambiciona con convertirse en la mayor del mundo. Se puede acceder mediante dispositivos Android tanto en su formato Phone como Tablet.

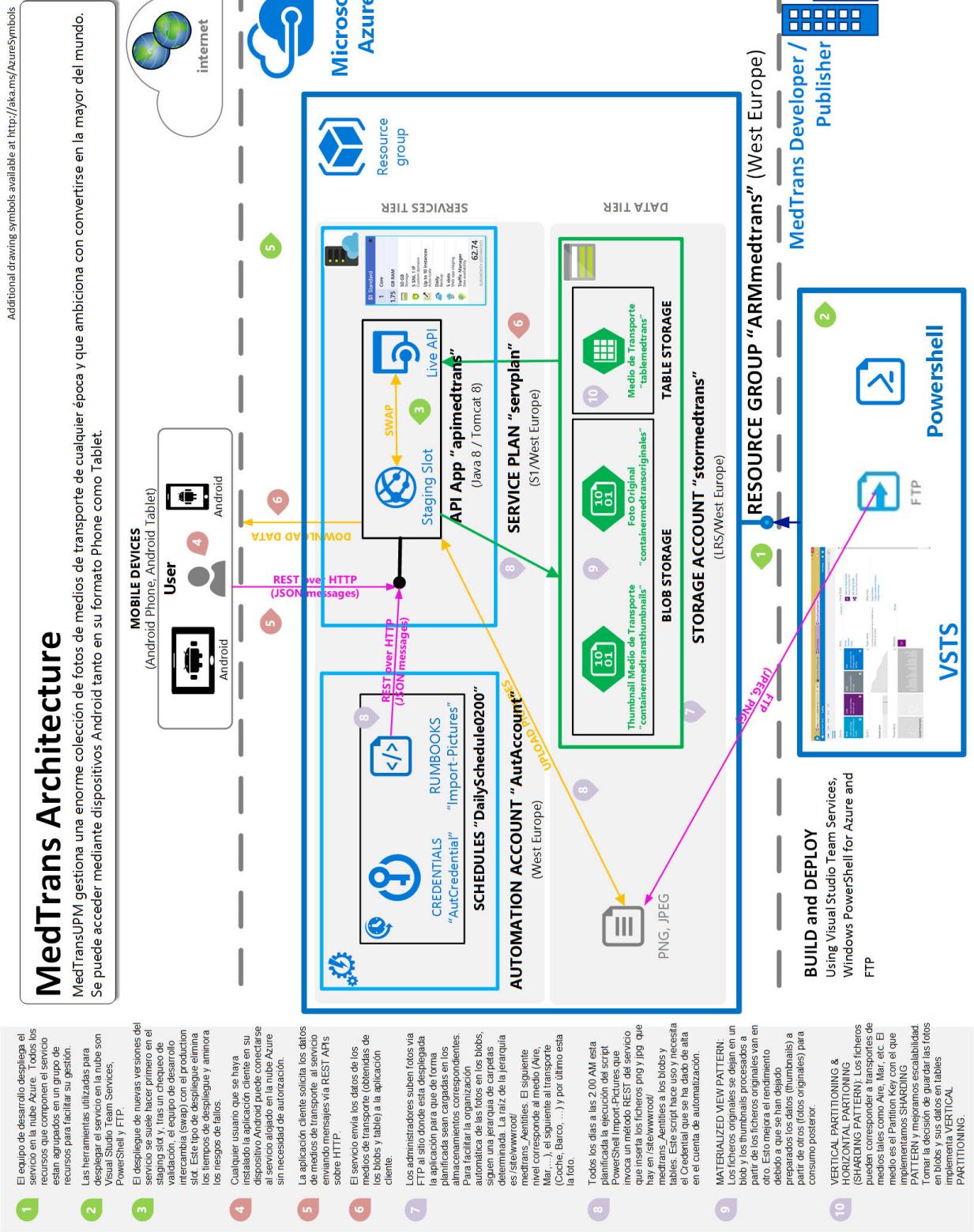


Fig. 76 Vista de los servicios Azure

3.4.3.2 Vista de flujo de trabajo DevOps

Esta vista va dirigida a los desarrolladores y operadores. Al igual que la vista anterior, para su elaboración simplemente se han empleado diagramas de cajas y líneas y se han combinado vistas modulares, C&C y de asignación.

Su propósito es mostrar cómo a través del ALM de VSTS se lleva a cabo el flujo de trabajo propuesto por DevOps (ver Sección 2.6). Dicho flujo consiste en desarrollar una aplicación, desplegarla, monitorizarla y aprender de su uso en producción, modificarla en respuesta a lo que se haya visto y repetir el ciclo. Dicho ciclo tiene que ser repetible, predecible y corto. Es decir, el periodo de tiempo entre que se tiene una idea para una característica y los clientes la están usando y proporcionando *feedback* nuevamente, debe ser tan corto como sea posible. Añadir que para establecer este proceso se deben seguir las siguientes buenas prácticas o patrones: automatización total, control de versiones, e integración y entrega continua. A continuación se describe la vista de la Fig. 77:

- En primer lugar, se debe tener configurado el entorno de trabajo (ver Sección 5.1). Dicha configuración consiste en las siguientes acciones y pasos:
 - Se da de alta la cuenta Microsoft "**alumnopfg2016@hotmail.com**" y se activa la suscripción de Microsoft Azure. A la suscripción Microsoft Azure se le asocia la cuenta Microsoft anterior (ver Sección 5.1.1).
 - A la cuenta Microsoft "**alumnopfg2016@hotmail.com**" también se le asocia el alta de la cuenta VSTS "**alumnoPFG2016**" (ver Sección 5.1.2)
 - Se configura la cuenta VSTS "**alumnoPFG2016**".
 - Se crea el proyecto VSTS "**PFGAlex2016**". se determina que la gestión del mismo va a ser mediante la metodología Scrum y que el sistema de control de versiones va a ser Git (ver Sección 5.1.2).
 - Se conceden permisos de acceso al repositorio de versiones²¹ (ver Sección 5.1.3) de dicho proyecto por un lado y al proyecto (ver Sección 5.1.4) por otro. Solo se pueden conceder permisos sobre el proyecto a miembros del repositorio.
 - Se configura un agente de compilación privado (ver Sección 5.1.5) para compilar localmente en Android. Para compilar en Java se puede hacer uso

²¹ https://alumnopfg2016.visualstudio.com/DefaultCollection/_git/PFGAlex2016

de los agentes de compilación hospedados en Microsoft Azure. Los agentes de compilación se utilizan para construir las aplicaciones en base al código fuente del repositorio.

- Se configura el punto de conexión "**Azure Pass(SPN)**" entre VSTS y la suscripción de Microsoft Azure (ver Sección 5.1.6). De esta forma, se podrán efectuar despliegues en dicha suscripción mediante *scripts* PowerShell.
 - Se crean los grupos de recursos de desarrollo "**ARMmedtransdes**", test "**ARMmedtranstest**" y producción "**ARMmedtrans**" en la suscripción Microsoft Azure (ver Sección 5.1.7) mediante la ejecución desde VSTS del *script* PowerShell "**Deploy-AzureApi.ps1**" que se encuentra en el repositorio. Los tres grupos de recursos son inicialmente iguales con la excepción que en el de desarrollo y en el de test contienen un plan de servicio de aplicaciones "**Shared D1**" y el de producción contiene uno de tipo "**Standard S1**". La diferencias principales son que el "**Shared D1**" es gratuito mientras que el "**Standard S1**" tiene un coste de 62.74 Euros y permite configurar espacios de implementación.
- Una vez se ha configurado el entorno, el flujo de trabajo DevOps comienza con el desarrollo de los componentes de la aplicación:
 - El *frontend* se desarrolla en Android Studio en lenguaje Android.
 - El API del *backend* se desarrolla en Eclipse en lenguaje Java y para un contenedor *Web Tomcat*.
 - El despliegue de grupos de recursos y recursos en la suscripción de Microsoft Azure se desarrolla mediante *scripts* PowerShell.
 - Siguiendo las buenas prácticas de DevOps, se efectúa un control de versiones sobre los componentes de la aplicación. El sistema de gestión de versiones utilizado es Git. Al repositorio del proyecto se puede acceder desde VSTS.
 - Además, sobre el entorno de desarrollo y, continuando con las buenas prácticas de DevOps, se ha activado la integración continua (ver 5.2.1). De esta forma, cada vez que se actualiza el repositorio, se construyen todos los componentes de la aplicación automáticamente en base al mismo. Para llevarlo a cabo, en VSTS se han definido varios *scripts*, los cuales se ejecutan cada vez que cambia el contenido del repositorio, y que definen cómo y qué construir. A continuación se enumeran los distintos automatismos de construcción definidos:
 - "**PFGAlex2016 Deploy AzureRM Resources**" define como desplegar un grupo de recursos en la suscripción Microsoft Azure utilizando un agente hospedado (ver

Sección 5.2.2). Tras su ejecución se despliega en Microsoft Azure el grupo de recurso de desarrollo "**ARMmedtransdes**" con sus correspondientes recursos. En este despliegue se ha definido que el contenedor *Web* es Tomcat, que el API va a estar desarrollado en Java y que se llama "**apimedtransdes**". Sin embargo, esto es solo la definición del API. El WAR correspondiente a dicho API no se despliega en esta construcción sino en la que se describe a continuación.

- **"PFGAlex2016 Deploy Tomcat"**: define como construir el API Java (ver Sección 5.2.3). Utiliza un agente hospedado. En este caso, la construcción consiste en compilar, pruebas unitarias y copiar el WAR en el servidor *Web* alojado en Microsoft Azure. El contenedor Tomcat del servidor *Web* despliega el WAR en cuanto lo detecta. En esta construcción se completa la parte *backend*.
- **"PFGAlex2016 Deploy Android"**: define como construir la aplicación Android (ver Sección 5.2.4). Utiliza un agente privado, el cual tiene que estar en ejecución cuando se lanza la construcción. En este caso, la construcción consiste en compilar, pruebas unitarias, generar, firmar el APK e instalarlo en un AVD.
- En el proceso definido en el PFG, no se ha activado la integración continua para los entornos de test y producción. En este caso, se ha decidido que el despliegue en dichos entornos no sea automático: un operador desde VSTS se encarga del versionado (agrupar una serie de componentes bajo una etiqueta que los identifica) de la aplicación y de su despliegue al entorno de test (de ejecutan los mismos automatismos que en el paso anterior) y, tras una serie de pruebas, al de producción.
- En definitiva, una vez que la versión ha pasado las pruebas funcionales, de integración y de carga (ver Sección 5.2.5) en el entorno de test, se puede desplegar la aplicación en el entorno de producción. El operador desde VSTS despliega la versión de la parte *backend* de la aplicación en el espacio de implementación (*staging slot*). Una vez pasadas las pruebas de aceptación cuyo objetivo es chequear que el despliegue ha ido bien, el operador desde el portal de Microsoft Azure efectúa el intercambio (*swap*) del espacio de implementación por el de producción.
- Por tanto, ya finalizadas fases de desarrollo y despliegue correspondientes al flujo de trabajo DevOps, es momento de monitorizar la aplicación y aprender de su uso en producción para modificarla en respuesta a lo que se haya visto o el *feedback* de los clientes. De esta forma, se repite nuevamente el ciclo, el cual debe ser tan corto como sea posible para entregar valor a los clientes más rápidamente y mejorar la calidad del software. Todas las modificaciones propuestas se gestionan aplicando la metodología ágil de desarrollo desde el *product backlog* de VSTS.

DevOps

Entrega valor a los clientes más rápidamente, mejora la calidad del software y obtiene feedback acerca del rendimiento y uso. Elimina barreras entre desarrolladores, operadores y usuarios para racionalizar el flujo de desarrollo desde el backlog hasta producción.

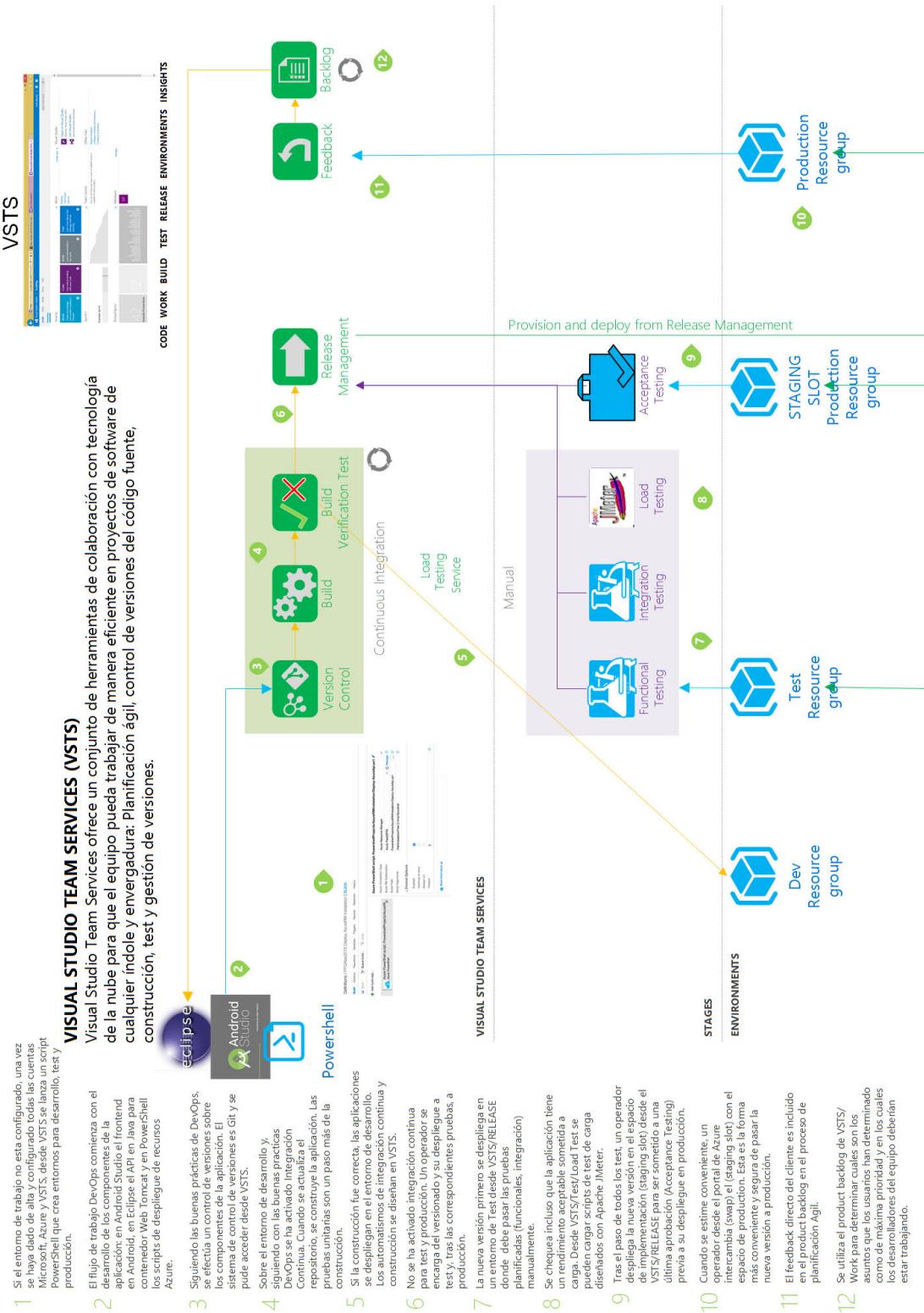


Fig. 77 Vista de flujo de trabajo

4 Desarrollo de una aplicación Android utilizando servicios de Microsoft Azure

En este capítulo se abordan los **OBJ1** y **OBJ2** y se demuestran mediante el caso de estudio descrito en el capítulo 3 para lo cual se van a tratar con cierto detalle los dos módulos principales que componen la aplicación, es decir, el *frontend* y el *backend*.

4.1 Servicio API REST Java (*backend*)

Para el caso de estudio, se debe cumplir un requisito referente a la interoperabilidad, el **RI1**, que especifica que el sistema debe ser capaz de operar en sistemas heterogéneos. Como ya hemos visto (ver Sección 2.2.1), la forma más sencilla de conseguirlo es mediante servicios REST. Además, la restricción **C2** imponía que el servicio debía ser escrito en Java. En este sentido, desde el portal de Azure, es posible descargar un prototipo de API REST para Java (otras posibilidades son NodeJS y ASP.NET) que accede a un almacén de datos en la nube de prueba. Afortunadamente, después de analizar dicho prototipo, se ha considerado que era aprovechable en gran medida y con unas pocas adaptaciones iba a facilitar y agilizar la tarea.

Por tanto, se ha construido un RESTful API en Java alojado en Microsoft Azure y accesible a través de la URL "<http://apimedtrans.azurewebsites.net/mediotransporte>". Este servicio define un interfaz que debe ser respetado por los clientes que deseen hacer uso de él (ver Sección 4.1.2). Para facilitar la implementación de dicho servicio *Web* en Java de acuerdo con el estilo arquitectónico REST se utiliza Jersey.

4.1.1 Construcción: Eclipse y Maven (*pom.xml*)

La aplicación se desarrolla en Eclipse²² y se construye con Maven²³ (herramienta de *software* para la gestión y construcción de proyectos Java), en cuyo fichero de configuración ("*pom.xml*") se especifican las bibliotecas necesarias y otros aspectos relacionados con la configuración de la aplicación (ver Fig. 78):

²² <https://eclipse.org/>

²³ <https://maven.apache.org/>

- El *groupId* junto con el *artifactId* debe especificar una aplicación única. El *artifactId* es el nombre con el que se conoce a nuestra aplicación. En este caso "**swagger-jaxr-server**".
- El *packaging* especifica el tipo de empaquetamiento: jar, war, ejb, rar, ear, etc. En nuestro caso, se ha especificado que el empaquetamiento debe ser "jar".
- En el bloque *Dependencies* se especifican las bibliotecas que se necesitan para construir el módulo:
 - Swagger: esta biblioteca se deben incluir para poder documentar RESTful APIs en Java con anotaciones en el código fuente. De esta forma los consumidores del API podrán conocer de forma actualizada las operaciones que les ofrece dicho API.
 - swagger-jersey-jaxrs 1.5.4
 - Api Servlet: esta biblioteca incluye la especificación JAX-RS que permite crear RESTful APIs en Java con anotaciones en el código fuente.
 - servlet-api 2.5
 - Jersey: esta biblioteca implementa la especificación JAX-RS.
 - jersey-core 1.18.1
 - jersey-client 1.18.1
 - jersey-server 1.18.1
 - jersey-multipart 1.18.1
 - jersey-servlet 1.18.1
 - jersey-json 1.18.1
 - Azure: esta biblioteca se debe incluir para poder interactuar con la cuenta de almacenamiento Azure.
 - Azure-storage 4.0.0

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>io.swagger</groupId>
  <artifactId>swagger-jaxrs-server</artifactId>
  <packaging>jar</packaging>
  <name>swagger-jaxrs-server</name>
  <version>1.0.0</version>
  <dependencies>
    <dependency>
      <groupId>io.swagger</groupId>
      <artifactId>swagger-jersey-jaxrs</artifactId>
      <version>1.5.4</version>
    </dependency>

    <dependency>
      <groupId>com.microsoft.azure</groupId>
      <artifactId>azure-storage</artifactId>
      <version>4.0.0</version>
    </dependency>

    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
    </dependency>

    <dependency>
      <groupId>com.sun.jersey</groupId>
      <artifactId>jersey-core</artifactId>
      <version>1.18.1</version>
    </dependency>
    ....
  </dependencies>
  <repositories>
    <repository>
      <id>sonatype-snapshots</id>
      <url>https://oss.sonatype.org/content/repositories/snapshots</url>
      <snapshots><enabled>true</enabled></snapshots>
    </repository>
  </repositories>
</project>

```

Fig. 78 pom.xml

4.1.2 Interfaz API: JAX-RS con Jersey

JAX-RS (*Java API for RESTful Web Services*) es una especificación y un API del lenguaje de programación Java, parte de Java EE 6, que usa anotaciones para simplificar el desarrollo de los servicios *Web* de acuerdo con el estilo arquitectónico REST [41].

Por otra parte, Jersey es la implementación de la especificación JAX-RS que se ha elegido en este PFG. Otras opciones eran RESTEasy de Jboss o Restlet. Se ha elegido Jersey por ser la más empleada [42].

A continuación se muestra el código fuente en Java de la clase principal del API (Fig. 79). En ella se ven anotaciones JAX-RS tales como:

- **@Path**: identifica el patrón de ruta al cual el recurso responde. Se puede aplicar tanto a nivel de clase como de método.

En nuestro caso:

- A nivel de clase responde a:
 - `"/mediotransporte"`
- A nivel de método responde a:
 - `"/rangefrom/{fromId}"`
 - `"/{id}"`
 - `"/carga"`
- **@GET, @POST, etc.**: identifica los métodos HTTP a los que el recurso responde.
- **@Produces**: especifica el tipo de medio MIME o representaciones que un recurso puede producir y enviar de vuelta al cliente. Se puede aplicar a nivel de clase o a nivel de método. Puede consumir más de un tipo.

En nuestro caso, nuestro API produce JSON que es, básicamente, un formato de texto ligero para el intercambio de datos.
- **@ApiParam (@QueryParam/@PathParam)**: identifica los parámetros que acepta el API y que se extraen de la petición. Pueden ser, entre otras cosas, QueryParam yPathParam. Básicamente, los QueryParam se añaden al final de la URL después de `?` y losPathParam son parte de la URL.

Por ejemplo, FIRST sería un ApiParam y 2 un QueryParam en la siguiente petición HTTP:

`"http://apimedtrans.azurewebsites.net/mediotransporte/rangefrom/FIRST?nTokens=2"`

```

import javax.ws.rs.core.Response;
import javax.ws.rs.*;

@Path("/mediotransporte")

public class MediosTransporteApi {

    private final MediosTransporteApiService delegate = MediosTransporteApiServiceFactory.getMediosTransporteApi();

    @GET
    @Path("/{fromId}")
    @Produces({ "application/json", "text/json" })

    public Response mediosTransporteGetNThumbnailsFromKey(
        @ApiParam(value = "ID de la primera foto de referencia.",required=true) @PathParam("fromId") String fromId,
        @ApiParam(value = "Total de fotos a obtener.",required=true) @QueryParam("nTokens") int nTokens)
        throws NotFoundException {
        Response response=delegate.mediosTransporteGetNThumbnailsFromKey(fromId, nTokens);
        return response;
    }

    @GET
    @Path("/{id}")
    @Produces({ "application/json", "text/json" })

    public Response medioTransporteGetOriginalById(
        @ApiParam(value = "ID de la foto.",required=true) @PathParam("id") String id)
        throws NotFoundException {
        Response response= delegate.medioTransporteGetOriginalById(id);
        return response;
    }

    @POST
    @Path("/carga")
    @Produces({ "application/json", "text/json" })

    public boolean mediosTransportePostAll()
        throws NotFoundException {
        Boolean response=delegate.mediosTransportePostAll();
        return response;
    }
}

```

Fig. 79 Extracto del API con las anotaciones JAX-RS

En la Fig. 80 se muestra el código fuente en Java del modelo. La única anotación JAX-RS que hay es `@JsonProperty` y solo sirve para cambiar el nombre de una propiedad en el JSON. Las otras anotaciones, `@ApiModel` y `@JsonProperty`, son de Swagger y solo sirven para documentar. Es importante destacar, que este modelo hereda de la clase `TableServiceEntity` de Microsoft Azure los atributos “`partitionKey`” y “`rowKey`”. Estos atributos junto con los atributos de tipo `String` “`medio`” y “`transporte`” y el atributo de tipo `array` de `bytes` “`imagen`” conforman una entidad “`MedioTransporte`”.

```

import com.fasterxml.jackson.annotation.JsonProperty;
import com.microsoft.azure.storage.table.TableServiceEntity;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value="MedioTransporte", description = "Modelo de medio de transporte.")

public class MedioTransporte extends TableServiceEntity {

    @ApiModelProperty(value = "Tipo de transporte", required =true)
    @JsonProperty("transporte")
    private String transporte;

    @ApiModelProperty(value="Foto del medio de transporte", required=true)
    @JsonProperty("imagen")
    private byte[] imagen;

    public MedioTransporte( String id, String medio, String transporte) {
        this.partitionKey = medio;
        this.rowKey = id;
        this.transporte=transporte;
        this.imagen = new byte[0];
    }

    public String getTransporte() {
        return transporte;
    }

    public void setTransporte(String transporte) {
        this.transporte = transporte;
    }

    public byte[] getImagen() {
        return imagen;
    }

    public void setImagen(byte[] imagen) {
        this.imagen = imagen;
    }
}

```

Fig. 80 Extracto del modelo con las anotaciones JAX-RS

Para finalizar este capítulo, se especifica de forma completa y detallada el interfaz del API en la tabla que se muestra en la Fig. 81.

ANOTACIONES REST @Path(/mediotransporte)	Ejemplo: Petición URL / Respuesta JSON
<pre data-bbox="186 460 616 572">@GET @Path("/rangefrom/{fromId}") @Produces({"application/json", "text/json"})</pre> <p data-bbox="186 601 616 777">Obtiene los N thumbnails siguientes a una de referencia. Útil para implementar paginación y mejorar el rendimiento. Las fotos se ordenan por fecha de alta: la primera foto se puede referenciar con id=FIRST.</p>	<p data-bbox="638 432 1405 487">GET http://apimedtrans.azurewebsites.net /mediotransporte/rangefrom/FIRST?nTokens=2</p> <pre data-bbox="638 496 1286 958">[{ "partitionKey": "Mar", "rowKey": "Mar_Barco_0000422333_20160412_170940.jpg", "etag": "W/\\"datetime'2016-04-12T17%3A09%3A44.9246349Z\\\"", "timestamp": "2016-04-12T17:09:44.924+0000", "transporte": "Barco", "imagen": "/9j/4AAQSkZJRgBAgAAAQABAAAD..... //9k="} }, { "partitionKey": "Mar", "rowKey": "Mar_Barco_0000422333_20160412_170940.jpg", "etag": "W/\\"datetime'2016-04-12T17%3A09%3A44.9246349Z\\\"", "timestamp": "2016-04-12T17:09:44.924+0000", "transporte": "Barco", "imagen": "/9j/4AAQSkZJRgBAgAAAQABAAAD/..... //9k="}]</pre>
<pre data-bbox="186 1001 616 1113">@GET @Path("/{id}") @Produces({"application/json", "text/json"})</pre> <p data-bbox="186 1142 616 1197">Obtiene la foto original correspondiente al id.</p>	<p data-bbox="638 967 1388 1022">GET http://apimedtrans.azurewebsites.net/mediotransporte/Mar_Barco_48_22_941.jpg</p> <pre data-bbox="638 1030 1286 1220">{ "partitionKey": "Mar", "rowKey": "Mar_Barco_0002478648_20160412_170941.jpg", "etag": "W/\\"datetime'2016-04-12T17%3A09%3A50.9823645Z\\\"", "timestamp": "2016-04-12T17:09:50.982+0000", "transporte": "Barco", "imagen": "/9j/4AAQSkZJRgBAgAAAQABAAAD/..... //9k="}</pre>
<pre data-bbox="186 1262 616 1374">@POST @Path("/carga") @Produces({ "application/json", "text/json" })</pre> <p data-bbox="186 1404 616 1628">Las fotos originales que se deseen incorporar a la base de datos se tienen que encontrar previamente en la ruta del servidor home\site\wwwroot\medtrans_AEntities. Las fotos deben estar adecuadamente organizadas en la jerarquía de directorios propuesta.</p>	<p data-bbox="638 1227 1204 1281">POST http://apimedtrans.azurewebsites.net /mediotransporte/carga</p> <pre data-bbox="638 1290 682 1315">true</pre>

Fig. 81 Peticiones HTTP REST implementadas en el caso de estudio

4.1.3 Documentación API: Swagger

Swagger es un *framework* para documentar RESTful APIs escritos en lenguaje tales como Java, Node.js o .NET para que los consumidores del API puedan conocer de forma dinámica y actualizada las operaciones que les ofrece dicho API [43].

El API se documenta mediante anotaciones [44] en el código fuente (ver Fig. 82). Una vez se haya anotado convenientemente el API y con el API ya desplegado, utilizando la herramienta Swagger UI [45] se genera dinámicamente la documentación del API. Dicha herramienta proporciona adicionalmente un para poder probar las operaciones del API.

Para instalar Swagger UI, hay que descárgalo de Internet²⁴, descomprimirlo y copiar el directorio “*dist*” en la ubicación deseada. Como Swagger UI no es más que es una colección de HTML, Javascript y CSS sin dependencias, se va a poder alojar en cualquier entorno del servidor o en su máquina local no teniendo que recopilar ni construir nada.

```
package io.swagger.api;

import io.swagger.annotations.ApiParam;
import io.swagger.annotations.Contact;
import io.swagger.model.MedioTransporte;
import io.swagger.utils.Utils;
import io.swagger.api.NotFoundException;

import javax.ws.rs.core.Response;
import javax.ws.rs.*;

@Path("/mediostransporte")

@io.swagger.annotations.SwaggerDefinition(
    info = @io.swagger.annotations.Info(
        description = "Obtiene fotos de medios de transporte",
        version = "V1.0.0",
        title = "API de fotos de medios de transporte",
        termsOfService = "https://www.etsisi.upm.es/",
        contact = @io.swagger.annotations.Contact(
            name = "Alejandro Mora Rodriguez",
            email = "alejandro.mora.rodriguez@alumnos.upm.es",
            url = "https://www.etsisi.upm.es/"
        ),
        license = @io.swagger.annotations.License(
            name = "Apache 2.0",
            url = "http://www.apache.org/licenses/LICENSE-2.0"
        )
    ),
    consumes = {"application/json", "application/xml"},
    produces = {"application/json", "application/xml"},
    schemes = {io.swagger.annotations.SwaggerDefinition.Scheme.HTTP, io.swagger.annotations.SwaggerDefinition.Scheme.HTTPS},
    externalDocs = @io.swagger.annotations.ExternalDocs(value = "ETSI", url = "https://www.etsisi.upm.es/")
)

@javax.annotation.Generated(value = "class io.swagger.codegen.languages.JaxRSServerCodegen", date = "2015-12-03T07:51:23.751Z")
public class MediosTransporteApi {
    .....
}
```

Fig. 82 Extracto de código del API con anotaciones Swagger

²⁴ <https://github.com/swagger-api/swagger-ui>

En nuestro caso de estudio la carpeta “*dist*” se ha renombrado a “*swagger*” y se ha copiado vía FTP en la ubicación del servidor “*/site/wwwroot/webapps*”. Para consultar la documentación dinámica que genera Swagger de la API de nuestro caso de estudio en base a las anotaciones efectuadas en el código, se introduce en cualquier navegador de Internet (ver Fig. 83) la dirección *Web* “<http://apimedtrans.azurewebsites.net/swagger>”.

The screenshot shows a web browser displaying the Swagger UI for the **MedioTransporte** API. The URL in the address bar is <http://apimedtrans.azurewebsites.net/swagger>. The page title is "Swagger Server".

Obtiene fotos de medios de transporte

ETSI SI
<https://www.etsisi.upm.es/>
Created by Alejandro Mora Rodriguez
See more at <https://www.etsisi.upm.es/>
[Contact the developer](#)
[Apache 2.0](#)

MedioTransporte

Show/Hide | List Operations | Expand Operations

POST	/mediotransporte/carga	Incorpora las fotos localizadas en un directorio del servidor a la base de datos.
GET	/mediotransporte/rangefrom/{fromId}	Obtiene los N thumbnails siguientes a una de referencia.
GET	/mediotransporte/{id}	Obtiene la foto original correspondiente al ID.

[BASE URL: , API VERSION: 1.0.0] VALID { }

Fig. 83 Documentación de API generada con Swagger UI

De esta forma, se facilita tanto a desarrolladores como no desarrolladores interactuar con el API para tener una visión clara sobre como el API responde a parámetros y a opciones (ver Fig. 84).

GET /mediotransporte/rangefrom/{fromId} Obtiene los N thumbnails siguientes a una de referencia.

Implementation Notes
Util para implementar paginación y mejorar el rendimiento. Las fotos se ordenan por fecha de alta: la primera foto se puede referenciar con ID=FIRST.

Response Class (Status 200)
OK

Model	Example Value
<pre>{ "partitionKey": "string", "rowKey": "string", "etag": "string", "timestamp": "2016-04-25T06:57:23.772Z", "transporte": "string", "imagen": [...] }</pre>	

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
fromId	FIRST	ID de la primera foto de referencia.	path	string
nTokens	5	Total de fotos a obtener.	query	integer

Try it out! Hide Response

Curl
curl -X GET --header 'Accept: application/json' 'http://apimedtrans.azurewebsites.net/mediotransporte/rangefrom/FIRST?nTokens=5'

Request URL
http://apimedtrans.azurewebsites.net/mediotransporte/rangefrom/FIRST?nTokens=5

Response Body

```
{
    "transporte": "Barco",
    "imagen": "/9j/4AAQSkZJRgABAgAAAQABAAD/2wBDAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UH
},
{
    "partitionKey": "Mar",
    "rowKey": "Mar_Barco_0002478648_20160412_170941.jpg",
    "etag": "W/\\"datetime'2016-04-12T17%3A09%3A50.9823645Z'\\\"",
    "timestamp": "2016-04-12T17:09:50.982+0000",
    "transporte": "Barco",
    "imagen": "/9j/4AAQSkZJRgABAgAAAQABAAD/2wBDAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UH
},
{
    "partitionKey": "Mar",
    "rowKey": "Mar_Barco_0003398148_20160412_170942.jpg",
    "etag": "W/\\"datetime'2016-04-12T17%3A09%3A56.632709Z'\\\"",
```

Response Code
200

Fig. 84 Prueba de operación de API con Swagger UI

4.2 Cliente REST Android (*frontend*)

Para el caso de estudio, se debe cumplir un requisito referente a la interoperabilidad, el **RI1**, que especifica que el sistema debe ser capaz de operar en sistemas heterogéneos. La forma más sencilla de conseguirlo es mediante servicios REST (ver Sección 3.4.2.1). Además, la restricción **C3** imponía que el cliente debe ser escrito en Android. En este sentido, desde el portal de Microsoft Azure, es posible la descarga de un prototipo de aplicación para Android (otras posibilidades son iOS, Xamarin, etc.) que accede a un almacén de datos en la nube de prueba y muestra su contenido en forma de lista. Desafortunadamente, después de analizar dicho prototipo, se ha optado por partir de cero ya que se estima que el coste de rediseño y adaptación sería mayor por los siguientes motivos:

- Accede directamente, con "*Android client SDK for Mobile Apps de Microsoft Azure*", a los datos. Por el contrario, en nuestra aplicación queremos que únicamente el servicio gestione el acceso a los almacenes de datos, aun a pesar de poder convertirse en un cuello de botella.
- El prototipo no tiene un diseño multipanel, y este es uno de los requisitos del caso de estudio. En este sentido, habría que rediseñarlo casi por completo.

Por tanto, se ha construido un cliente REST en Android que se comunica con el servicio API REST alojado en la URL "<http://apimedtrans.azurewebsites.net/mediotransporte>" en base interfaz de dicho servicio (ver Sección 4.1.2). Para facilitar la implementación de la comunicación HTTP en el cliente REST Android se ha utilizado la biblioteca Retrofit (ver Sección 4.2.2). Además, esta biblioteca permite efectuar las llamadas en forma asíncrona. De esta forma, se cumple tanto con el requisito **RU2** (la aplicación cliente no debe quedar bloqueada al consultar) de usabilidad como con el requisito **RI1** de interoperabilidad.

4.2.1 Construcción: Android Studio y Gradle (*build.gradle*)

La aplicación se desarrolla con Android Studio 2.0 y se construye con Gradle²⁵(herramienta para automatizar la construcción de proyectos en cualquier lenguaje), en cuyo fichero de configuración ("*build.gradle*") se especifican las bibliotecas necesarias y otros aspectos relacionados con la configuración de la aplicación (ver Fig. 85):

- Se aplica el *plugin* de Android para Gradle lo cual hace posible que en el bloque "**android**" se puedan especificar opciones específicas para Android, tales como:

²⁵ <http://gradle.org/>

- Versión compilación de la aplicación: 23 del API de Android
 - Versión de las herramientas SDK de construcción: 23.0.0
 - **Nombre con que se publica el paquete:** "*net.azurewebsites.apimedtrans*"
 - Versión mínima con la que compila la aplicación: 10 del API de Android
 - Versión testeo de la aplicación: 23 del API de Android
 - Número de versión de la aplicación: 1
 - Nombre de versión de la aplicación: 1.0
- En el bloque "***dependencies***" se especifican las bibliotecas que se necesitan para construir el módulo:
 - Android: estas bibliotecas se deben incluir para poder compilar una aplicación Android. En general, se recomienda incluir las bibliotecas "***v4 support***" y "***v7 appcompat***" ya que de esta manera la aplicación va a soportar un amplio rango de versiones Android y así nuestra aplicación de fotos no tendrá problemas de ejecución en las distintas versiones.
 - support-v7 23.1.1
 - appcompat-v7 23.1.1
 - recyclerview-v7 23.1.1
 - design 23.1.1
 - Retrofit: esta biblioteca se incluye para poder interactuar fácilmente con un API REST (en nuestro caso de estudio con el API "***apimedtrans***") usando JSON o XML e interfaces y anotaciones en el código evitando así la construcción manual de peticiones HTTP. Además, para trabajar con JSON, como sucede en nuestro caso de estudio, se debe incluir la biblioteca Gson.
 - retrofit 2.0.1
 - converter-gson 2.0.1

```

/**
 * Se aplica el plugin de Android para Gradle en esta construccion y hace posible que
 se puedan especificar opciones específicas para Android en el bloque {}.
 */
apply plugin: 'com.android.application'
/**
 * El bloque Android {} es donde se configuran las opciones específicas de construccion para Android.
 */
android {
    /**
     * compileSdkVersion especifica el nivel de Android API que debe utilizar Gradle
     * para compilar la aplicacion. Esto significa que la aplicacion puede utilizar
     * caracteristicas incluidas en este nivel y en inferiores.
     * buildToolsVersion especifica la version de las herramientas SDK de construccion, utilidades de
     * linea de comandos y compilador que Gradle debe utilziar para construir la aplicacion. Se
     * descargan utilizando el SDK Manager.
    */
    compileSdkVersion 23
    buildToolsVersion "23.0.2"
    defaultConfig {
        /**
         * applicationId identifica de forma unica el paquete para su publicacion.
         * Sin embargo, el codigo debe referenciarlo como esta en main/AndroidManifest.xml
        */
        applicationId "net.azurewebsites.apimedtrans"
        // Determina el nivel de API minimo requerido para ejecutar la aplicacion.
        minSdkVersion 10
        // Especifica el nivel de API utilizado para testear la aplicacion.
        targetSdkVersion 23
        // Especifica el numero de version de la aplicacion.
        versionCode 1
        // Determina un nombre de version para la aplicacion.
        versionName "1.0"
    }
}
dependencies {
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'com.android.support:support-v4:23.1.1'
    compile 'com.android.support:recyclerview-v7:23.1.1'
    compile 'com.android.support:design:23.1.1'
    compile 'com.squareup.retrofit2:retrofit:2.0.1'
    compile 'com.squareup.retrofit2:converter-gson:2.0.1'
}

```

Fig. 85 *build.gradle*

4.2.2 Interactuación con API REST: Retrofit

Retrofit es una biblioteca de código abierto para Java y Android desarrollada por Square Open Source²⁶ que proporciona un *framework* para que dichas aplicaciones Java o Android

²⁶ <http://square.github.io/>

puedan interactuar fácilmente con un API REST usando JSON o XML e interfaces y anotaciones en el código evitando así la construcción manual de peticiones HTTP [46].

A continuación se enumeran los pasos que se han seguido para utilizar Retrofit en el cliente Android:

- **Se declara un interfaz:** se definen las diferentes funciones a la API REST con las anotaciones @POST, @GET de Retrofit. El patrón para definir las funciones es el mismo independientemente si son síncronas o asíncronas. El valor de retorno de estas funciones siempre es un objeto parametrizado Call<T> siendo T el POJO (*Plain Old Java Object*)²⁷ a serializar (ver Fig. 86).

```
import java.util.List;
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Path;
import retrofit2.http.Query;

public interface MediosTransporteAPI {
    /*Retrofit get annotation with our URL
     And our method that will return us the list of Book
     */
    @GET("/mediotransporte/rangefrom/{fromId}")
    Call<List<MedioTransporte>> mediosTransporteGetNThumbnailsFromId(@Path("fromId") String fromId,@Query("nTokens") int nTokens);

    @GET("/mediotransporte/{id}")
    Call<MedioTransporte> medioTransporteGetOriginalById( @Path("id") String id);
}
```

Fig. 86 Definición interfaz Retrofit

- **Se crea una instancia de Retrofit:** maneja la conexión con lo cual se debe especificar la URL base del servicio. Se debe especificar también un convertidor determinado porque si no Retrofit solo será capaz de aceptar *String* como resultado. Por lo tanto, si se quiere aceptar JSON como resultado, se debe indicar explícitamente con el método *addConverterFactory* (ver Fig. 87).
- **Se crea un servicio:** que se utilizará para realizar las peticiones al API REST (ver Fig. 87).

²⁷ POJO es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un *framework* en especial.

```

/* Creating a rest adapter */
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("http://apimedtrans.azurewebsites.net/")
    .addConverterFactory(GsonConverterFactory.create())
    .build();

//Creating an object of our api interface
MediosTransporteAPI apiService =
    retrofit.create(MediosTransporteAPI.class);

```

Fig. 87 Creación servicio Retrofit

- **Invocación síncrona del API:** se utiliza el método *execute* del servicio (ver Fig. 88). Esta llamada bloquea el hilo, así que no se debe llamar desde el hilo principal en Android o saltará *NetworkOnMainThreadException*. En nuestro cliente no se ha utilizado este tipo de invocaciones.

```

Call<List<MedioTransporte>> call = apiService.mediosTransporteGetNThumbnailsFromId(fromId, NTokens);
call.execute();

```

Fig. 88 Llamada síncrona en Retrofit

- **Invocación asíncrona del API:** se utiliza el método *enqueue* del servicio (ver Fig. 89). Si la petición tiene éxito se obtendrá la respuesta en *response.body()* del método *onResponse*. Si la petición falla se saldrá por *onFailure*.

```

Call<List<MedioTransporte>> call = apiService.mediosTransporteGetNThumbnailsFromId(fromId, NTokens);

call.enqueue(new Callback<List<MedioTransporte>>() {
    @Override
    public void onResponse(Call<List<MedioTransporte>> call, Response<List<MedioTransporte>> response) {
        if (response.isSuccessful()) {

            ListIterator<MedioTransporte> iterator = response.body().listIterator();
            while (iterator.hasNext()) {
                MedioTransporte medioTransporte = iterator.next();
                //Storing names to string array -> solo de aquellos que tengan thumbnail
                // (las cargas incongruentes -referencia sin thumb- no me interesa visualizarlas)
                if (medioTransporte.getImagen() != null && !medioTransporte.getImagen().isEmpty()) {
                    addItem(createMedioTransporte(medioTransporte.getRowKey(), medioTransporte.getPartitionKey(),
                        medioTransporte.getTransporte(), medioTransporte.getImagen(), null));
                } else {
                    //Con este error continuamos. Falta el thumbnail y probablemente el original.
                    // Error de coherencia o quizás no se ha ejecutado la fase 2 de la carga planificada.
                    Log.e(Utility.TAG, Utility.APPLIP_MESSAGE_ERROR + ": " + medioTransporte.getRowKey());
                }
            }

            if (onLoadThumbnailsListener != null) {
                onLoadThumbnailsListener.onLoadThumbnailsReady();
            }
        } else {
            if (onErrorListener != null) {
                String errorCode=String.valueOf(response.code());
                String errorMessage= response.message().toString();
                onErrorListener.onErrorReady(errorCode, errorMessage);
            }
        }
    }

    @Override
    public void onFailure(Call<List<MedioTransporte>> call, Throwable t) {
        if (onErrorListener != null) {
            String errorCode=Utility.ERROR_CODE_INTERNET;
            String errorMessage= Utility.APPLIP_MESSAGE_NETPROBLEMS;
            onErrorListener.onErrorReady(errorCode, errorMessage);
        }
    }
}
);

```

Fig. 89 Llamada asíncrona en Retrofit

4.2.3 Interfaz gráfica de usuario

La interfaz visual es multipanel en teléfonos de tipo tableta (ver Fig. 91) y de único panel en el resto (ver Fig. 92) con lo que se consigue cumplir con el requisito de usabilidad **RU1**. Para conseguirlo, desde Android Studio se ha tomado como base la **plantilla “Master/Detail Flow”** que nos proporciona este *framework* de desarrollo (ver Fig. 90) para realizar la implementación del multipanel.

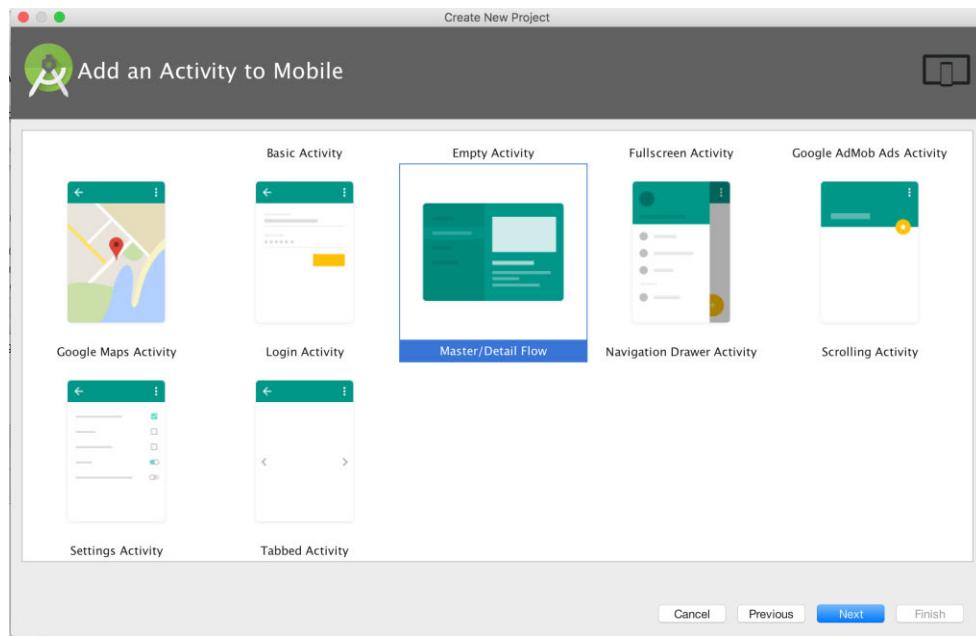


Fig. 90 Creación aplicación Android partiendo de la plantilla Master/Detail Flow

El interfaz gráfico de usuario de la aplicación es muy simple. A continuación se enumeran sus componentes:

- Una lista de medios de transporte. La lista se dispone en vertical y muestra para cada medio de transporte:
 - Vista en miniatura (*thumbnail*)
 - Posición en la lista.
 - Tamaño en bytes del *thumbnail*.
 - Tamaño en *bytes* de la foto original. Este dato inicialmente tiene valor 0 para todos los elementos de la lista. Se le da valor cuando el usuario selecciona algún thumbnail para visualizar su foto original. Adicionalmente, se utiliza como indicativo que la foto original la tenemos ya “cacheada” y no hay que ir a buscarla al servidor. Aunque sería idóneo, por problemas de almacenamiento es imposible “cachear” todas las fotos. Por consiguiente, independientemente del dispositivo y su capacidad, se ha determinado tener solo 5 fotos “cacheadas” simultáneamente: las fotos originales más antiguas en cache son sustituidas por las nuevas para mantener esta constante.
- La foto original del thumbnail seleccionado por el usuario.

- Un botón de refresco de la lista y cuyo ícono es 

La ubicación de estos componentes en el dispositivo móvil depende de si es multipanel (en teléfonos de tipo tableta) o de único panel (en el resto de teléfonos):

- En un multipanel (ver Fig. 91) se muestra una lista de medios de transporte en un tercio del lateral izquierdo del dispositivo. En los dos tercios restantes se muestra la foto original seleccionada. El botón de refresco se muestra en el borde inferior derecho de la pantalla.

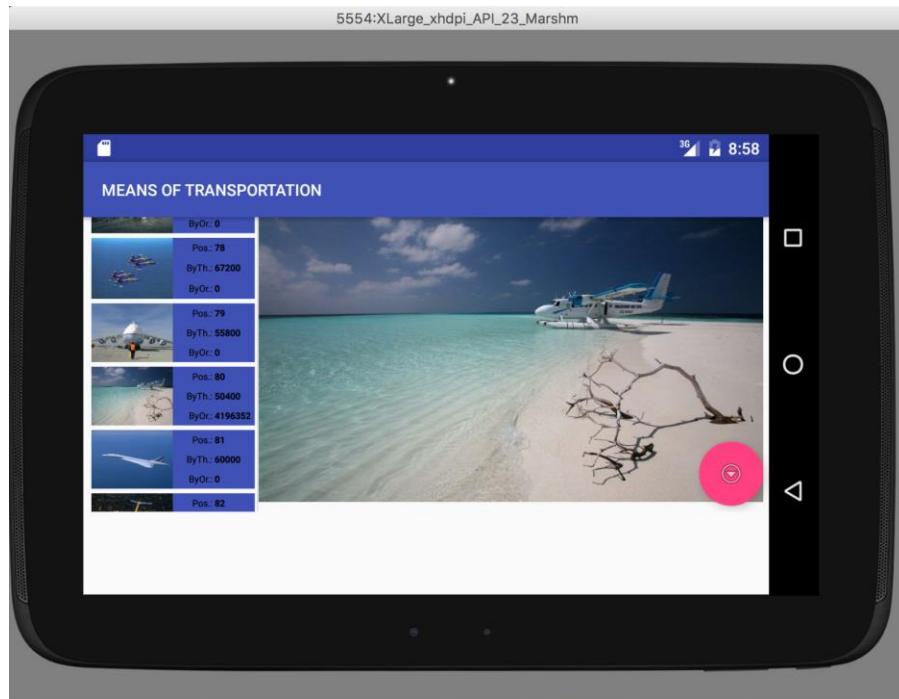


Fig. 91 Visualización aplicación Android en modo multipanel

- En el resto de teléfonos (ver Fig. 92) se muestra una lista de medios de transporte en toda la pantalla y, si se selecciona algún *thumbnail*, se muestra en otra pantalla. Para volver a la lista hay que pulsar el botón de vuelta a la pantalla anterior del Sistema Operativo (SO) Android. El botón de refresco se muestra en el borde inferior derecho de la pantalla.

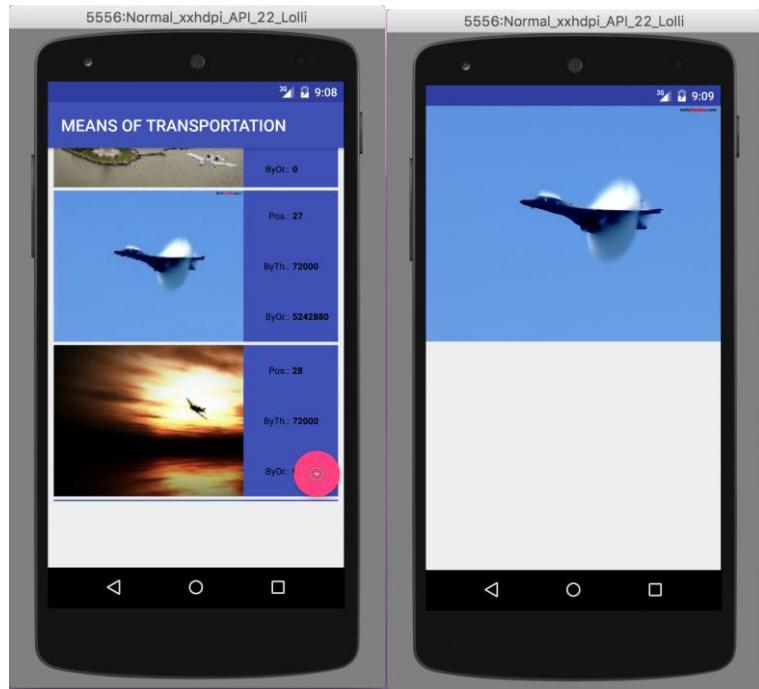


Fig. 92 Visualización de la aplicación Android en modo único panel

Adicionalmente, el cliente se ha diseñado lo suficientemente robusto como para que sea posible continuar ante errores tan comunes como los de conexión. En caso de fallos transitorios de este tipo, se muestra un mensaje descriptivo del error y se le ofrece al usuario la posibilidad de volver a intentarlo nuevamente (ver Fig. 93), para lo cual tendría que pulsar el botón de refresco indicado anteriormente. De esta forma se cumple con el requisito **RR1** de resistencia.

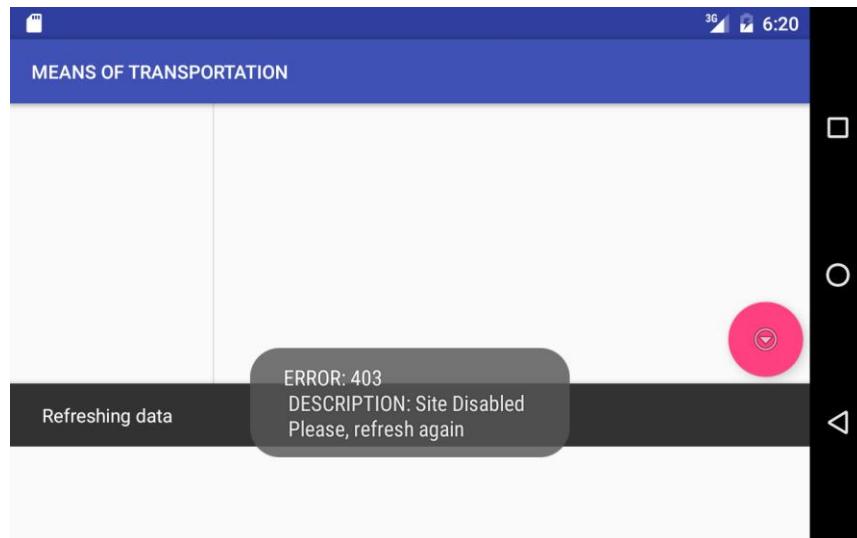


Fig. 93 Visualización mensaje error si el cliente REST no puede acceder al API RES

5 Administración del ciclo de vida de la aplicación con Visual Studio Team Services

El desarrollo, test y producción de la aplicación construida en el caso de estudio se ha realizado mediante ALM VSTS. En las siguientes secciones se describe cómo configurar el entorno inicial y el proceso diario de desarrollo.

5.1 Configuración inicial

En esta Sección se describen los pasos que hay que llevar a cabo para, desde el ALM VSTS, configurar de los entornos de desarrollo, test y producción en la suscripción de Azure en la que se va a desplegar la aplicación (ver Fig. 94).

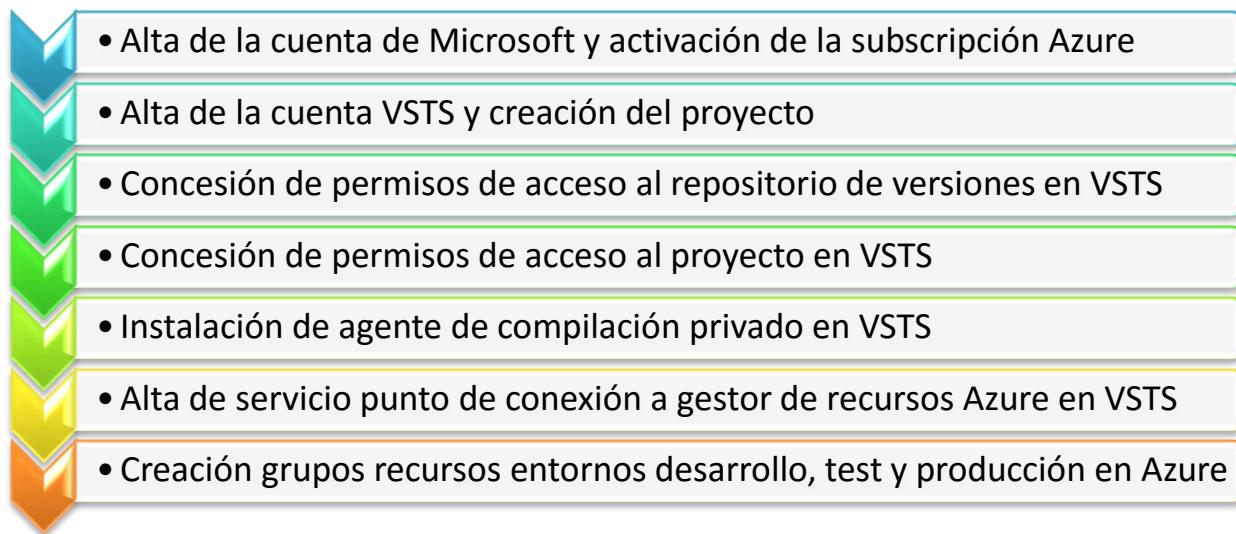


Fig. 94 Configuración entornos desarrollo, test y producción de la aplicación desde VSTS

5.1.1 Alta de cuenta de Microsoft y activación de suscripción a Microsoft Azure

Para desarrollar una aplicación en Microsoft Azure, lo primero que se ha de realizar es dar de alta una cuenta de Microsoft. Esta cuenta es sobre la que se va a activar la suscripción de

Microsoft Azure. Para desarrollar la aplicación del caso de estudio, se realizó dicha alta y activación. A continuación se describe cómo se hizo:

- Se da de alta una cuenta Microsoft desde "<https://signup.live.com>" En el caso del PFG se dio de alta la cuenta "alumnopfg2016@hotmail.com"
- Para activar la suscripción de Microsoft Azure se accede a "<http://www.microsoftazurepass.com>", se introduce el código de promoción y se le asocia a una cuenta Microsoft. En el caso del PFG se le asoció a la cuenta creada en el paso anterior "alumnoPFG2016@hotmail.com"(ver Fig. 95).

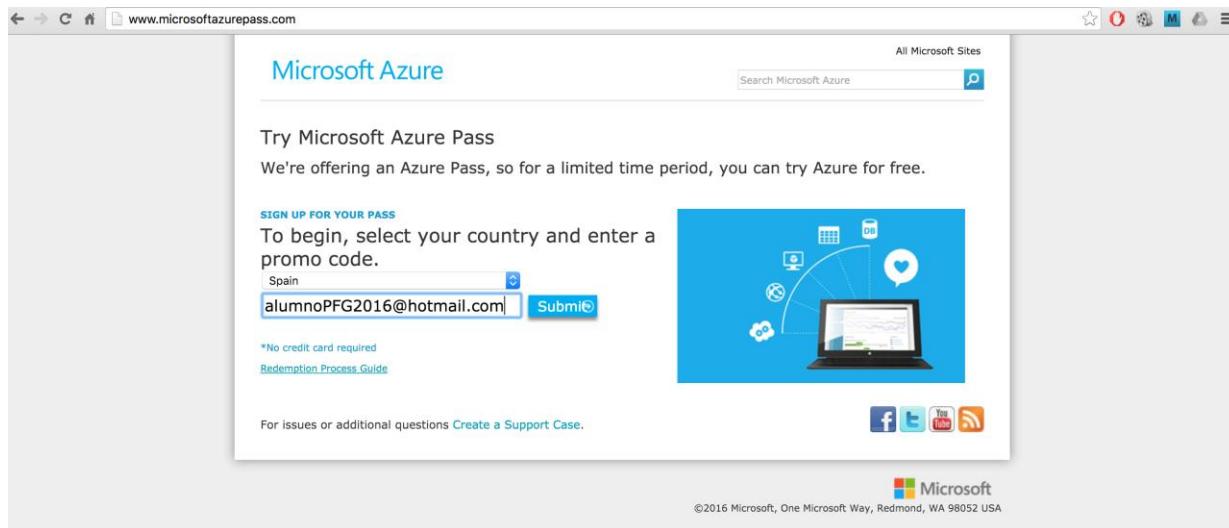


Fig. 95 Sitio web de activación de suscripciones de Azure

5.1.2 Alta de la cuenta VSTS y creación del proyecto

Como la gestión del ciclo de vida de la aplicación de este PFG se ha realizado utilizando VSTS, fue necesario darse de alta en VSTS creando una cuenta, para ya una vez dada de alta dicha cuenta, proceder a la creación del proyecto:

- Para crear una cuenta VSTS se accede a "<https://www.visualstudio.com>" utilizando una cuenta Microsoft. En el caso del PFG se introdujo la cuenta Microsoft "alumnoPFG2016@hotmail.com", y se pulsó "**Iniciar Sesión**"(ver Fig. 96):

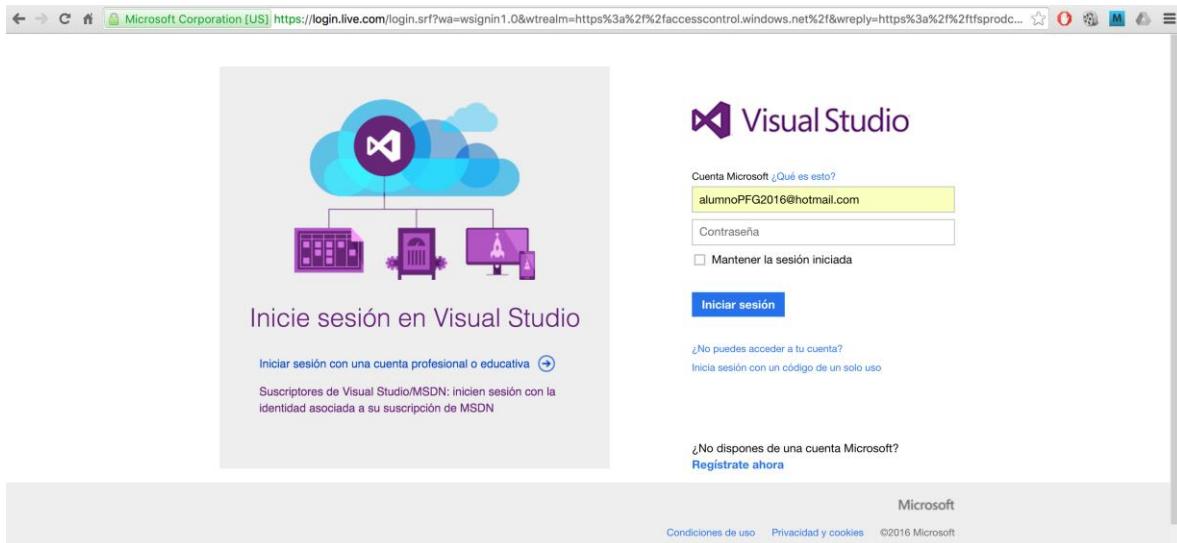


Fig. 96 Sitio Web registro VSTS

- Se pulsa "Crear una cuenta gratuita ahora" (ver Fig. 97).



Mi información

<div style="border: 1px solid #ccc; padding: 5px; width: 150px; height: 40px; background-color: #f0f0f0; display: flex; align-items: center; justify-content: center;"> AP </div> <p>alumno PFG2016 alumnopfg2016@hotmail.com Spain</p> <p>Editar información</p>	<p>Visual Studio Dev Essentials Use sus beneficios</p> <p>Cuentas <small>?</small></p> <p>https://alumnopfg2015.visualstudio.com/ (Owner) https://alumnopfg2019.visualstudio.com/ (Owner) https://alumnopfg2015.visualstudio.com/ (Eliminado) Restaurar https://alumnopfg2016.visualstudio.com/ (Owner) https://pfgallex2015.visualstudio.com/ (Owner) https://alexmorardriguez.visualstudio.com/ (Eliminado) Restaurar https://tsaccountperformacetest.visualstudio.com/ (Owner) https://oscar3377.visualstudio.com/</p> <p>Crear una cuenta gratuita ahora</p>
--	--

Fig. 97 Sitio Web creación cuenta VSTS

- Se introduce el nombre de la cuenta VSTS que se desea crear, se selecciona el sistema de control de versiones (puede elegirse entre *Team Foundation Version Control* y *Git*) y se pulsa "**Continue**". En el caso del PFG se introdujo "*alumnopfg2016*" como nombre principal de la URL y *Git* como sistema de control de versiones (ver Fig. 98).

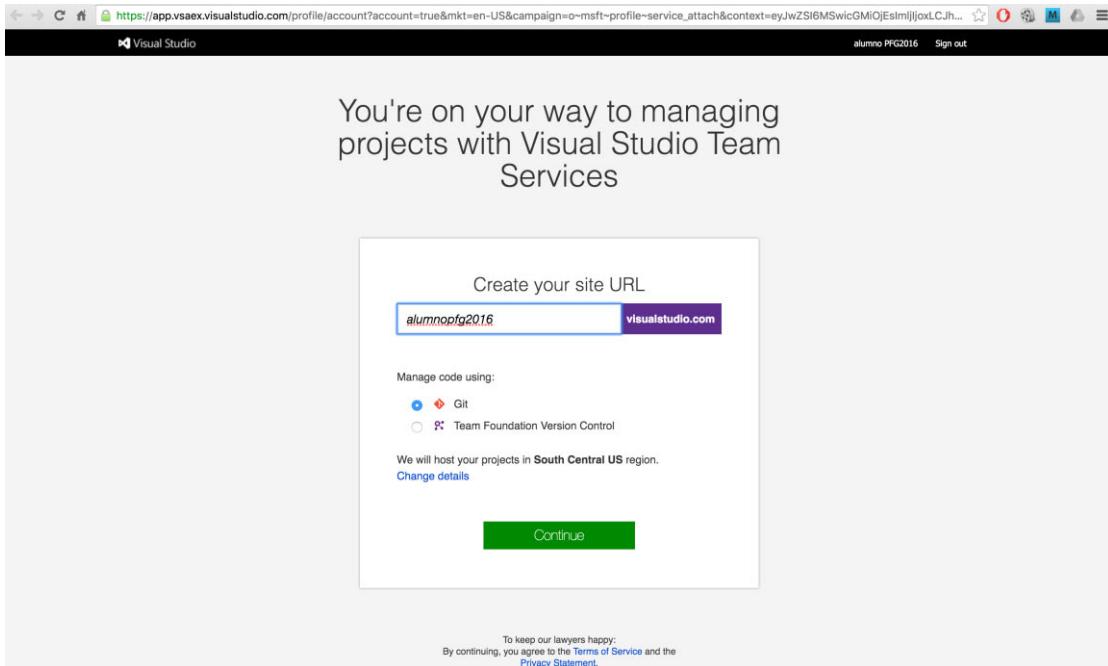


Fig. 98 Sitio Web opciones nueva cuenta VSTS

- Se crea un nuevo proyecto pulsando el enlace “*New*” del panel principal de nuestra cuenta VSTS. En el caso del caso de estudio del PFG se creó el proyecto “*PFGAlex2016*” (ver Fig. 99).

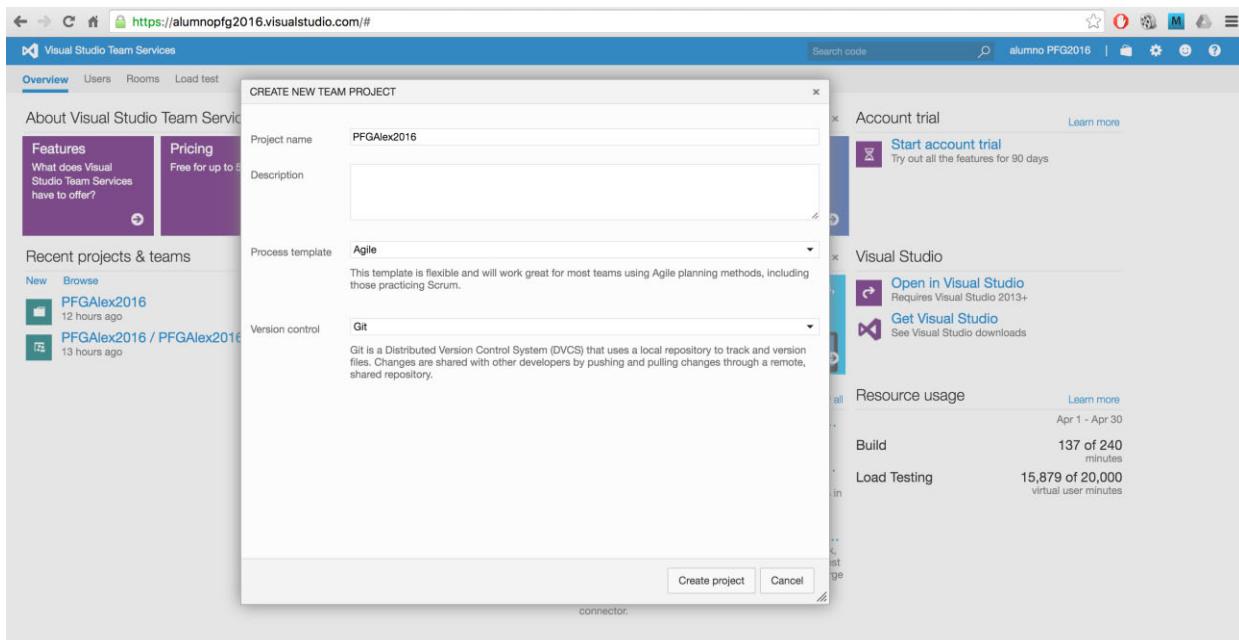


Fig. 99 Panel principal de la cuenta VSTS: creación de nuevo proyecto VSTS

5.1.3 Concesión de permisos de acceso al repositorio de versiones en VSTS

Una vez creado el proyecto en VSTS se procedió a crear los usuarios y las políticas de acceso al repositorio de control de versiones. Para poder dar de alta a un usuario como miembro del equipo VSTS, hay que darlo de alta como usuario del repositorio previamente. Para ello hay que seguir una serie de pasos que se describen a continuación:

- Se accede al panel de configuración de VSTS (ver Fig. 100)



Fig. 100 Botón de acceso al panel de configuración de VSTS

- A través de la pestaña "*Version Control*" y pulsando el botón "*Add User*" se añaden usuarios. En la Fig. 101 se puede ver como se ya se han añadido los usuarios "*alexmmorarodriguez@gmail.com*", "*Jennifer Pérez Benedí*" y "*yessica diaz*".

A screenshot of the VSTS Control Panel showing the "Version Control" section. Under "Repositories", "PFGAlex2016" is selected. In the "Security" tab of the "Security for PFGAlex2016 repository" dialog, the "Add user" button is highlighted. The right pane shows an "ACCESS CONTROL SUMMARY" table with permissions like "Administer", "Branch creation", etc., and a list of users: alexmorarodriguez@gmail.com, Jennifer Pérez Benedí, Project Collection Build Service (alumnoPFG2016), and yessica diaz.

Fig. 101 Botón para añadir usuarios al sistema de control de versiones en VSTS

- En la Fig. 102 se puede ver como se introduce el email de un nuevo usuario, "*verorodriguezblanco@gmail.com*", al que también se desea conceder permisos de acceso al repositorio.

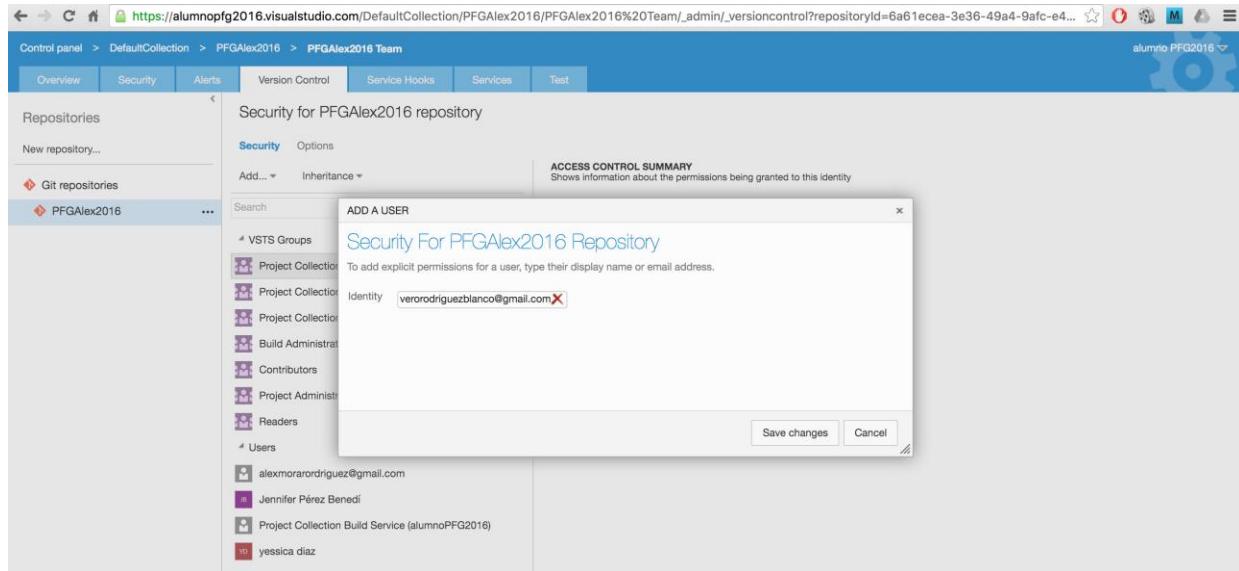


Fig. 102 Ventana para añadir usuario al sistema de control de versiones en VSTS

- Por defecto el usuario recién dado de alta no tiene permisos de ningún tipo en el repositorio. Por lo tanto, se deben ir ajustando uno a uno cada uno de los posibles permisos con los valores *Allow/Deny/Not set*” (ver Fig. 103). A todos los usuarios dados de alta se les conceden (*Allow*) todos los permisos sobre el repositorio disponibles (*Administer/Branch creation/Contribute/Exempt from policy enforcement/Note management/Rewrite and destroy history/Tag creation*):

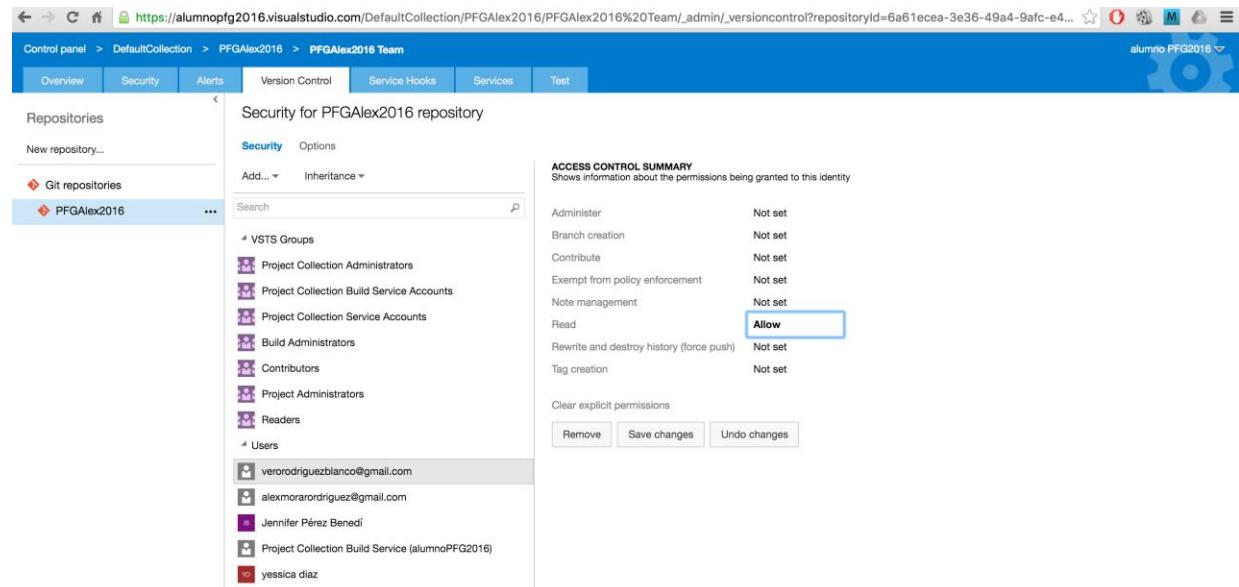


Fig. 103 Ajuste de los permisos de acceso al sistema de control de versiones en VSTS

- Al usuario recién dado de alta le llega un email informativo que se le han concedido permisos en el repositorio como el que se muestra a continuación (ver Fig. 104):

From VisualStudioTeamServices@microsoft.com ★
 Subject You have been invited to a Visual Studio Team Services project!
 To Vero ★
 Microsoft

Visual Studio

You've been invited!

Join alumno PFG2016 at <https://alumnopfg2016.visualstudio.com/> to plan, collaborate and ship great software.

Here are two ways to accept the invitation:

[Join](#) [Open in Visual Studio](#)

Visual Studio Team Services give teams access to a [wide set of developer services](#) we think you'll love like:

- Unlimited free private code repos
- Track bugs, work items, feedback and more
- Develop in any language
- Use [Visual Studio, Eclipse, or your own IDE](#)
- Continuous integration builds
- Enterprise-grade services, priced to be small-team friendly



And here are your details:

- Account URL - <https://alumnopfg2016.visualstudio.com/>
- Your sign-in address - verorodriguezblanco@gmail.com
- Not sure what to do? Contact alumno PFG2016 at alumnopfg2016@hotmail.com
- Need help? Check our [support options](#)

Happy Coding!
 The Visual Studio team

Fig. 104 Correo confirmación que llega a usuario dado de alta al repositorio en VSTS

5.1.4 Concesión de permisos de acceso al proyecto en VSTS

Para poder dar de alta a los usuarios que van a formar parte del equipo de proyecto y puedan gestionarlo a través de VSTS, previamente hay que darlos de alta en el repositorio de dicho proyecto (ver 5.1.3). Una vez se hecho esto, hay que seguir los pasos que se van a enumerar a continuación. A continuación se enumeran los pasos que se han seguido para conceder permisos de acceso al proyecto “PFGAlex2016” en VSTS a los usuarios “alexmorarodriguez@gmail.com”, “Jennifer Pérez Benedí”, “yessica diaz” y “verorodriguezblanco@gmail.com” (en la versión gratuita de VSTS solo se pueden dar de alta 5 usuarios):

- Desde la pestaña “Overview” del “Control Panel” del equipo de trabajo (Team Profile) definido en VSTS, en este caso “PFGAlex2016 Team”, se pulsa “Add User” para añadir a los usuarios, teniendo en cuenta que estos han tenido que ser dados de alta en el repositorio previamente (ver Sección 5.1.3). En el del equipo “PFGAlex2016 Team” del proyecto “PFGAlex2016” se dieron de alta los usuarios “alexmorarodriguez@gmail.com”, “Jennifer Pérez Benedí”, “yessica diaz” y “verorodriguezblanco@gmail.com” (ver Fig. 105).

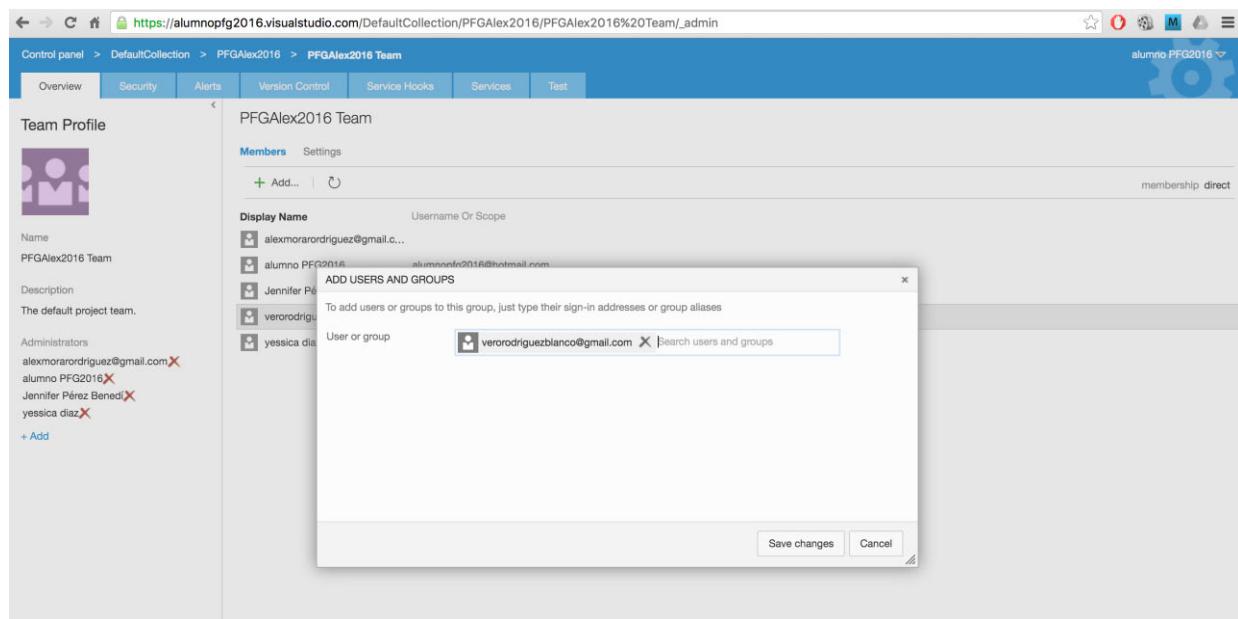


Fig. 105 Ventana de alta de miembros del equipo en un proyecto en VSTS

- Adicionalmente, se les pueden conceder permisos de administrador (ver Fig. 106). En el caso del “PFGAlex2016 Team” se dieron permisos de administrador a todos los usuarios (“alexmorarodriguez@gmail.com”, “Jennifer Pérez Benedí”, “yessica diaz” y “verorodriguezblanco@gmail.com”) para que pudieran tener un control y visualización completos del transcurso del desarrollo de la aplicación.

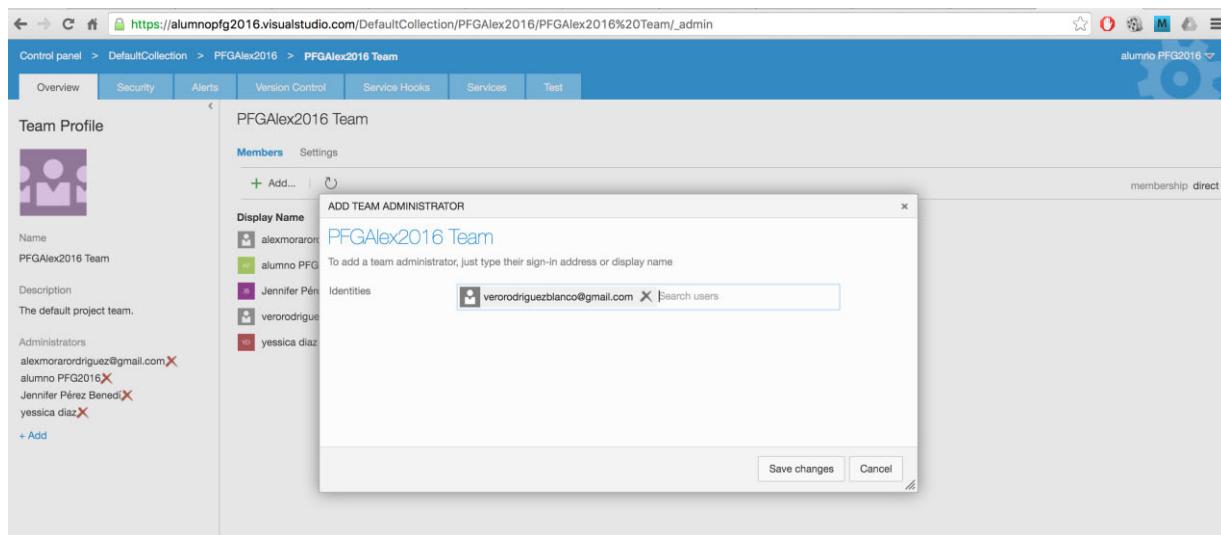


Fig. 106 Ventana de alta de usuario como administrador del equipo en VSTS

5.1.5 Instalación de un agente de compilación privado de VSTS en local

Entre los muchos servicios que proporciona VSTS se incluye el de compilación [47]. Con este servicio basado en tareas, se puede encolar compilaciones empleando para ello un agente. Los agentes de compilación son imprescindibles para implementar integración continua.

El agente de compilación puede ser privado (VSTS proporciona 1 agente privado gratis se ejecuta en nuestra máquina local y permite instalar software personalizado que no está disponible en el agente hospedado como por ejemplo la SDK de Android) u hospedado (el agente lo ejecuta *Microsoft* poniendo a disposición de los usuarios 240 minutos gratis al mes).

Este capítulo muestra como instalar un agente de compilación privado de VSTS en local. Es normal disponer de varios agentes de compilación para poder realizar compilaciones en paralelo tanto en la nube (con el agente hospedado) como en local (con el agente privado):

- Se accede al panel de configuración de VSTS (ver Fig. 107)



Fig. 107 Botón de acceso al panel de configuración de VSTS

- A través de la pestaña “Agent pools” se descarga el agente de compilación pulsando el botón “Download Agent” (ver Fig. 108).

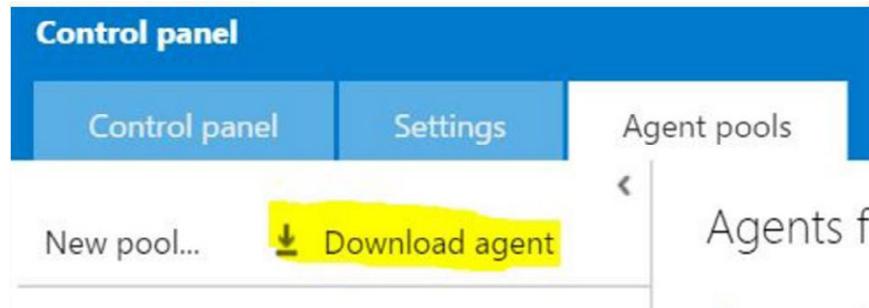


Fig. 108 Botón de descarga agente de compilación en pestaña Agent Pools VSTS

- Se descomprime el fichero descargado "*agent.zip*" que contiene los ficheros que se muestran en la Fig. 109.

Nombre	Fecha de modificación	Tipo	Tamaño
_diag	04/05/2016 20:16	Carpetas de archivos	
_work	04/05/2016 19:55	Carpetas de archivos	
agent	04/05/2016 19:19	Carpetas de archivos	
agent.old	15/04/2016 6:17	Carpetas de archivos	
tasks	20/04/2016 18:46	Carpetas de archivos	
ConfigureAgent.cmd	04/05/2016 17:18	Script de comandos ...	1 KB
keystoreAlex.jks	25/02/2016 22:44	Archivo JKS	3 KB
RunAgent.cmd	04/05/2016 17:18	Script de comandos ...	1 KB
settings.json	15/04/2016 5:59	Archivo JSON	1 KB

Fig. 109 Contenido del fichero "agent.zip"

- Se abre una ventana de comandos de Windows con privilegios de administrador en el PC donde se va a configurar el agente y se ejecuta "*ConfigureAgent.cmd*" y se completa la información solicitada según se muestra en la Fig. 110.

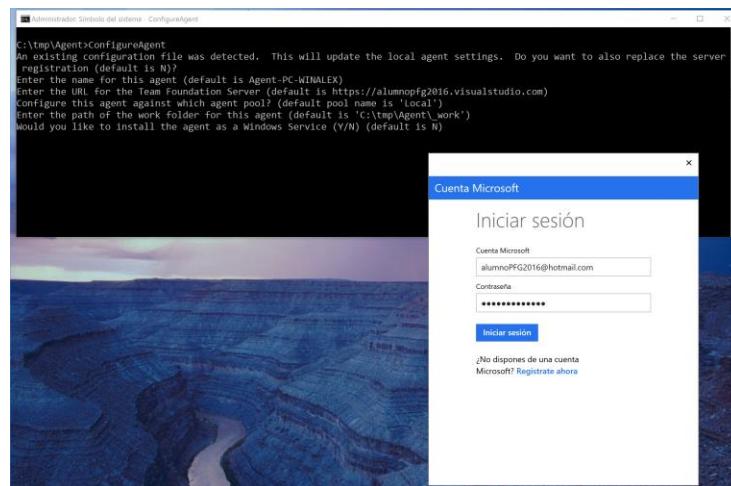


Fig. 110 Instalación agente de compilación

- Si la configuración ha ido bien, se podrá ver el nuevo agente de compilación en la pestaña “Agent pools” del panel de control de VSTS (ver Fig. 111).

Enabled	Name	Current Status	Requests	Capabilities	Name	Date Queued	Date Assigned	Date Started	
	Agent-PC-WINALEX	Idle	280	Build	Azure Installation	279	20/04/2016 18:59:13	20/04/2016 18:59:13	20/04/2016 1

Fig. 111 Agentes de compilación de VSTS

- Una vez configurado, para poder hacer uso del agente de compilación privado hay que ejecutar “agent\VsosAgent.exe”(ver Fig. 112). En el PFG es necesario tener el agente privado en ejecución en la máquina local donde se haya instalado para que se pueda construir el código escrito en Android (o cualquier otro lenguaje específico que esté instalado en local). Tenemos que utilizar un agente privado porque con el agente hospedado que nos proporciona Windows podemos construir en Java pero no es posible construir con Android.

```
C:\tmp\Agent>agent\VsoAgent
Agent: Starting
Authenticating to the server https://alumnopfg2016.visualstudio.com
Running the agent interactively: C:\tmp\Agent\agent\VsoAgent.exe
Registering the agent 'Agent-PC-WINALEX (Local)' with the server https://alumnopfg2016.visualstudio.com
Using SessionOwnerName 'PC-WINALEX'
Press Ctrl+C to quit...
```

Fig. 112 Ejecución de agente de compilación privado en consola de comandos Windows

5.1.6 Alta del servicio de punto de conexión a Microsoft Azure desde VSTS

Para poder hacer un despliegue de recursos a una suscripción de Azure, es necesario crear un punto de conexión al gestor de recursos de Microsoft Azure. Esto se realiza desde la pestaña *BUILD* de VSTS:

- Se accede al panel de configuración de VSTS (ver Fig. 113)



Fig. 113 Botón de acceso al panel de configuración de VSTS

- A través de la pestaña “Services” se pulsa “*New Service Point*” y se selecciona la opción “Azure Resource Manager” (ver Fig. 114)

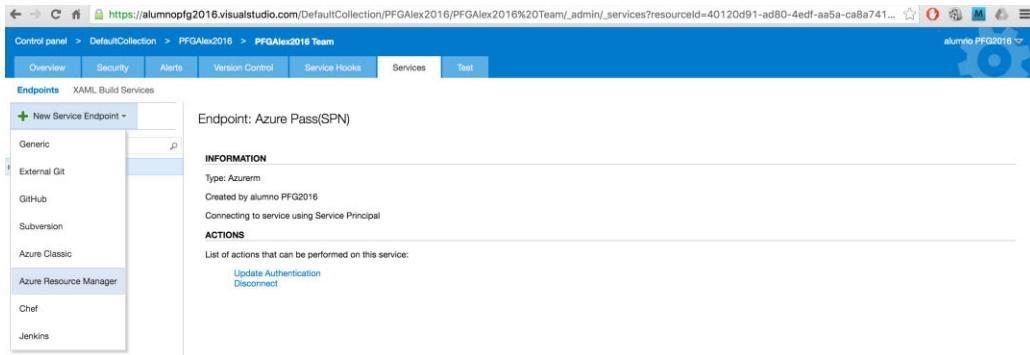


Fig. 114 Ventana de servicios de puntos de conexión de VSTS

- Se introduce la información solicitada acerca de la suscripción de Azure con la cual se desea establecer el servicio punto de conexión (ver Fig. 115). Para facilitar la obtención de dicha información se ejecuta el script **PowerShell** “*SNPCreation.ps1*” (se puede obtener pinchando el enlace²⁸ que aparece en la propia ventana. El script “*SNPCreacion.ps1*” se debe ejecutar desde una consola de PowerShell en Windows con permisos de administrador (ver Fig. 116).

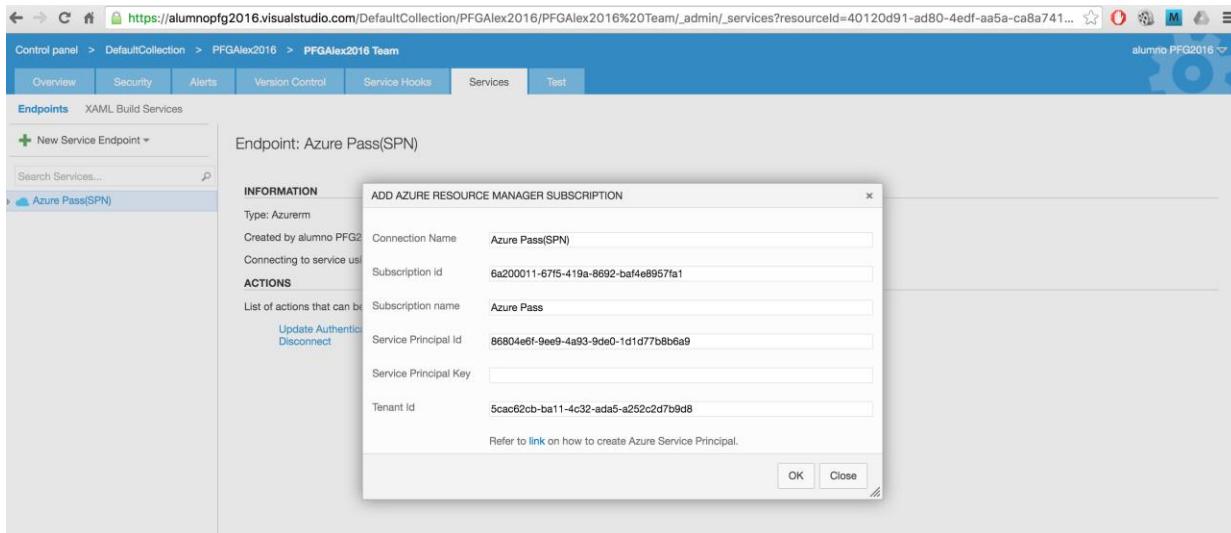


Fig. 115 Alta de punto de conexión a suscripción Azure en VSTS

²⁸ <https://blogs.msdn.microsoft.com/visualstudioalm/2015/10/04/automating-azure-resource-group-deployment-using-a-service-principal-in-visual-studio-online-buildrelease-management/>

```

PS C:\WINDOWS\system32> c:\Users\alejandro\Desktop\AzureInst\SNPCreation.ps1
cmdlet SNPCreation.ps1 en la posición 1 de la canalización de comandos
Proporcione valores para los parámetros siguientes:
(Escriba !? para obtener Ayuda.)
subscriptionName: Azure Pass
password: PFG2016alumno
Provide your credentials to access Azure subscription Azure Pass

Environment          : AzureCloud
Account             : alumnopfg2016@hotmail.com
TenantId            : 5cac62cb-ba11-4c32-ad45-a252c2d7b9d8
SubscriptionId      : 6a200011-67f5-419a-8692-baf4e8957fa1
CurrentStorageAccount :

Creating a new Application in AAD (App URI - http://vso.alejandro.560a037d-9c51-4b57-b3cd-889ec2ca516c)
Azure AAD Application creation completed successfully (Application Id: 86804e6f-9e9-4a93-9de0-1d1d77b8b6a9)
Creating a new SPN
SPN creation completed successfully (SPN Name: 86804e6f-9ee9-4a93-9de0-1d1d77b8b6a9)
Waiting for SPN creation to reflect in directory before Role assignment
Assigning role (owner) to SPN App (86804e6f-9ee9-4a93-9de0-1d1d77b8b6a9)

RoleAssignmentId   : /subscriptions/6a200011-67f5-419a-8692-baf4e8957fa1/providers/Microsoft.Authorization/roleAssignments/cb2cd625-cdc7-47c4-b81b-263594f90cda
Scope              : /subscriptions/6a200011-67f5-419a-8692-baf4e8957fa1
DisplayName        : vso.alejandro.560a037d-9c51-4b57-b3cd-889ec2ca516c
SignInName         :
RoleDefinitionName: Owner
RoleDefinitionId  : 8e3af657-a8ff-443c-a75c-2fe8c4bcbb635
ObjectId           : 73b83b14-c7d0-412d-b9e3-560f5afecc6
ObjectType         : ServicePrincipal

SPN role assignment completed successfully

Copy and Paste below values for service connection
*****
Connection Name: Azure Pass(SPN)
Subscription Id: 6a200011-67f5-419a-8692-baf4e8957fa1
Subscription Name: Azure Pass
Service Principal Id: 86804e6f-9ee9-4a93-9de0-1d1d77b8b6a9
Service Principal key: <Password that you typed in>
Tenant Id: 5cac62cb-ba11-4c32-ad45-a252c2d7b9d8
*****

```

Fig. 116 Resultado ejecución script "SNPCreacion.ps1" en consola PowerShell

- Si todo ha ido bien, el servicio punto de conexión aparecerá en la pestaña "*Services*" del "*Control Panel*" (ver Fig. 117).

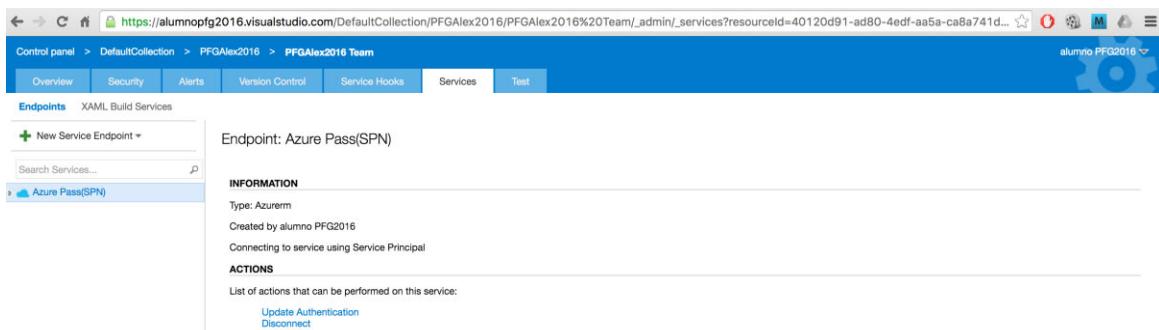


Fig. 117 Servicios puntos de conexión en la pestaña Services del Control Panel de VSTS

5.1.7 Creación de grupos recursos en Microsoft Azure

Una vez dados de alta las cuentas, usuarios, puntos de conexión con la suscripción de Azure, repositorios y concedidos sus permisos, se crean los grupos de recursos para desarrollo, test y producción en Microsoft Azure. Para ello, se han de seguir una serie de pasos:

- Desde la pestaña de **BUILD** de VSTS se pulsa “+” para crear una nueva definición de construcción. Se comprueba que dicha definición aparece bajo el epígrafe “*Build definitions*”.
- Se añade un paso de construcción pulsando “*Add build step...*” y, de entre las opciones de construcción disponibles, se selecciona “*Azure PowerShell script: PowershellProjects/AzureRM*” y se completa. Esta opción permite ejecutar un *script* PowerShell en una suscripción Azure.
- Tomando como ejemplo el PFG, se han configurado los siguientes parámetros tal y como muestra la Fig. 118:
 - Se indica el tipo de conexión con la suscripción de Azure. En el campo “*Azure Connection Type*” se selecciona “*Azure Resource Manager*” (la otra opción es “*Azure Clasic*”).
 - Se indica el nombre del punto de conexión que se ha creado a la suscripción Azure (ver 5.1.6), es decir, “*Azure Pass(SPN)*”.

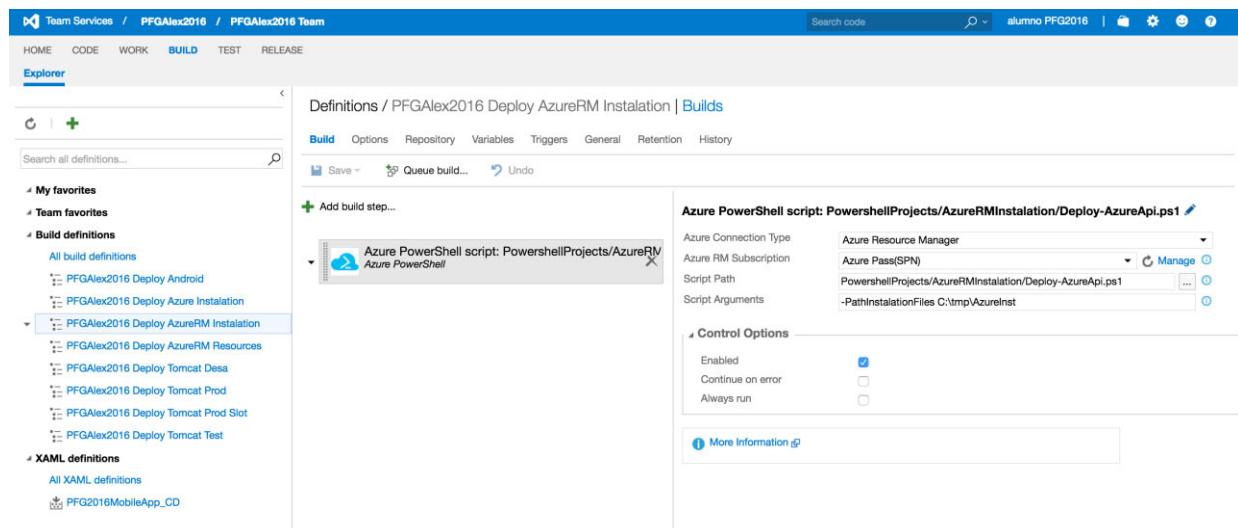


Fig. 118 Definición de la compilación que instala los entornos de trabajo en Azure

- Se indica la ubicación en el repositorio del *script* PowerShell. En el campo "Script Path" se especifica la plantilla "*Deploy-AzureApi.ps1*"²⁹. Esta plantilla contiene los cmdlets Azure PowerShell que crean todo el entorno de trabajo con los tres grupos de recursos Azure y sus correspondientes servicios (ver Fig. 119).

```

Param(
    [Parameter(Mandatory=$true)] [string] $PathInstallationFiles
)

function Deploy-AzureResourcesApi () {

Param(-- 
    [Parameter(Mandatory=$true)] [string] $ResourceGroupName,
    [Parameter(Mandatory=$true)] [string] $ResourceGroupLocation,
    [Parameter(Mandatory=$true)] [string] $ServicePlanName,
    [Parameter(Mandatory=$true)] [string] $ServicePlanSku,
    [Parameter(Mandatory=$true)] [string] $ServicePlanWorkerSize,
    [Parameter(Mandatory=$true)] [string] $ApiName,
    [Parameter(Mandatory=$true)] [string] $SlotName,
    [Parameter(Mandatory=$true)] [string] $StorageAccountName,
    [Parameter(Mandatory=$true)] [string] $StorageAccountType,
    [Parameter(Mandatory=$true)] [string] $TableName,
    [Parameter(Mandatory=$true)] [string] $ContainerOriginalName,
    [Parameter(Mandatory=$true)] [string] $ContainerThumbnailsName,
    [Parameter(Mandatory=$true)] [string] $AutomationAccountName,
    [Parameter(Mandatory=$true)] [string] $AutomationCertificateName,
    [Parameter(Mandatory=$true)] [string] $AutomationCredentialName,
    [Parameter(Mandatory=$true)] [string] $TemplateFile,
    [string] $CertificatePath,
    [Parameter(Mandatory=$true)] [string] $RunbookName,
    [Parameter(Mandatory=$true)] [string] $RunbookPath ,
    [Parameter(Mandatory=$true)]
    # [Parameter(Mandatory=$true)] [string] $Credential,
    #[Parameter(Mandatory=$true)] [string] $CredentialPassword,
    [Parameter(Mandatory=$true)] [string] $ScheduleName,
    [Parameter(Mandatory=$true)] [string] $ScheduleStartDay
)

$AzureRmResourceGroup=Get-AzureRmResourceGroup -Name $ResourceGroupName -Verbose
if ($AzureRmResourceGroup) {
    Remove-AzureRmResourceGroup -Name $ResourceGroupName -Verbose -Force
}

# Create or update the resource group using the specified template file and template parameters file
New-AzureRmResourceGroup -Name $ResourceGroupName -Location $ResourceGroupLocation -Verbose -Force -ErrorAction Stop

```

Fig. 119 Extracto *script* PowerShell "*Deploy-Azure.ps1*"

- Si todo ha ido bien, en el portal de Azure aparecen los tres grupos de recursos ("*ARMmetrans*", "*ARMmedtransdes*" y "*ARMmedtranstest*") con sus correspondientes recursos (ver Fig. 120):

²⁹ El fuente de la plantilla está accesible desde el repositorio, en particular de ruta *PowerShellProjects/AzureRMInstallation* de la rama master y en la misma ruta del CD que acompaña a esta memoria

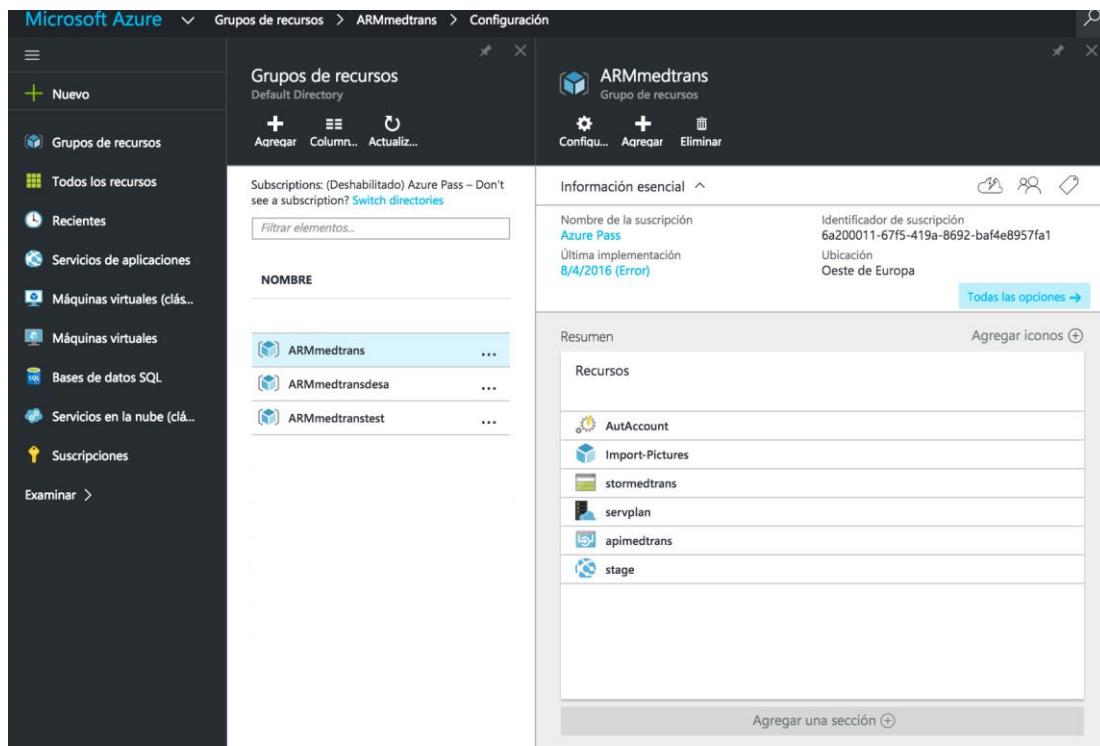


Fig. 120 Grupos de recursos para los entornos producción, desarrollo y test en Azure

5.2 Proceso diario de desarrollo

En esta Sección se describen las tareas habituales que se deben llevar a cabo para construir, copiar y desplegar las aplicaciones o para modificar recursos en los grupos de recursos de la suscripción de Azure (ver Fig. 121).



Fig. 121 Proceso diario de desarrollo

5.2.1 Construcción manual o automática de aplicaciones desde VSTS

En esta Sección se describen los pasos que hay que seguir para construir aplicaciones desde VSTS en base al contenido de un repositorio. La construcción puede ser manual o automática y, además, se puede utilizar un agente de compilación hospedado o privado:

- Si se va a utilizar un agente de compilación privado hay que tenerlo en ejecución en el equipo donde se haya instalado (ver Sección 5.1.5).
- Si se desea construir de forma automática hay que activar *Continuous Integration (CI)* en la pestaña *Triggers* (ver Fig. 122). Esto significa que se va a construir la aplicación cada vez que se detecte un cambio en los ficheros de código fuente de la rama del repositorio que se indique.

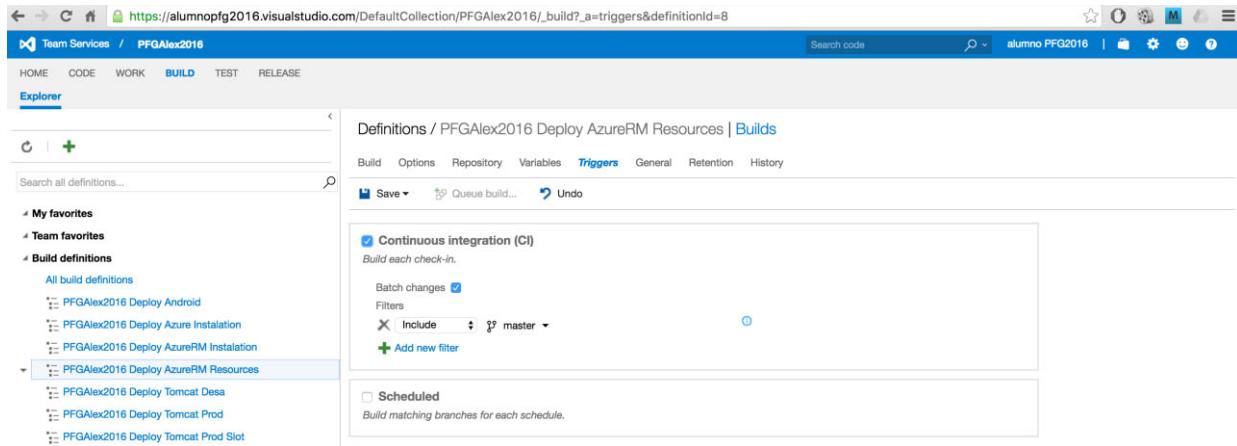


Fig. 122 Activación CI con cambios en la rama master del repositorio en VSTS

- Para la integración continua hay que definir cuál va a ser el agente de compilación por defecto (ver Fig. 123).

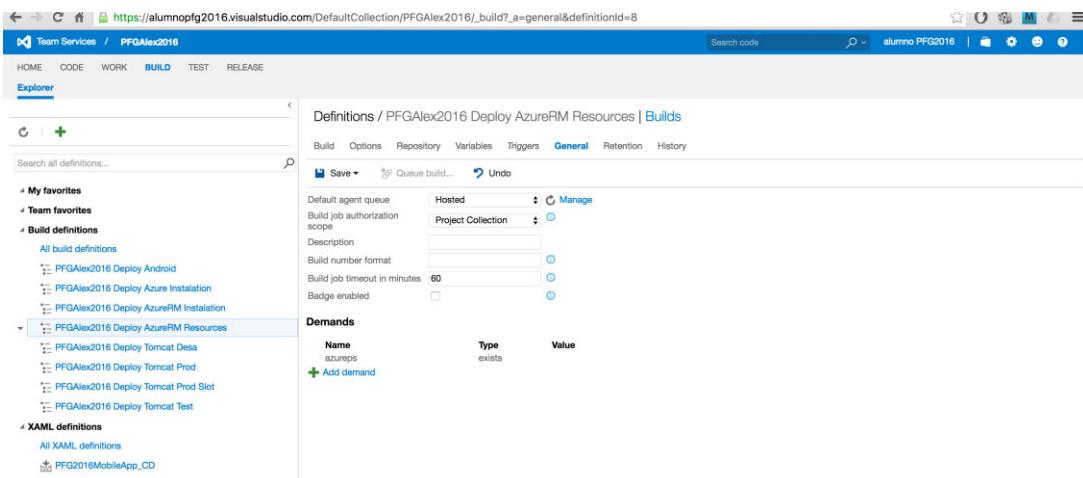


Fig. 123 Selección del agente de compilación por defecto en VSTS

- Para la forma manual hay que pulsar "*Queue build...*". En la ventana que aparece se selecciona como "*Queue*" el agente de compilación deseado (hospedado o privado), la rama del repositorio que se va a construir (*master*, ...) y se pulsa "*OK*"(ver Fig. 124).

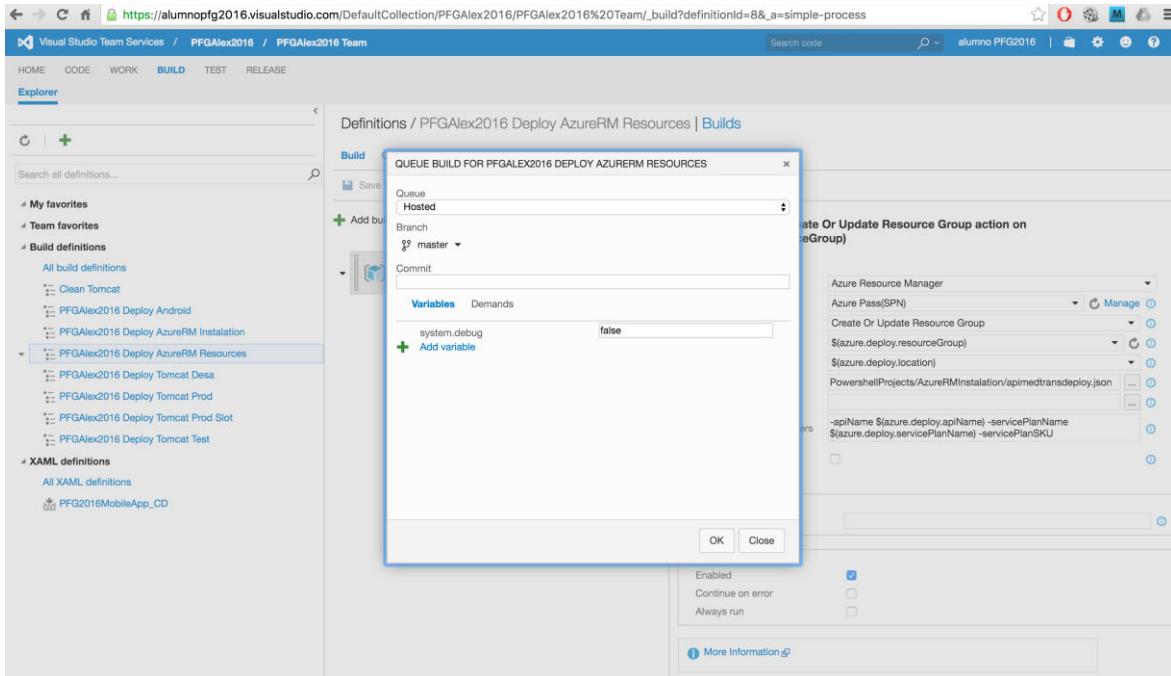


Fig. 124 Encolar construcción aplicación con agente hospedado

- En cuanto el agente de compilación comienza su trabajo se muestra una consola con el *log* (ver Fig. 125).

```

Build Succeeded
Build
Ran for 5.9 minutes (Hosted Agent), completed 0 seconds ago

Console Logs
Executing the following commandline:
C:\IRVMS\services\Vms\TaskAgentProvisioner\Tools\agents\1.98.1\agent\worker\vsoWorker.exe /name:Worker-6f53d8a1-25a1-4237-8dc9-6baff4ee1fbc /queue:Queue new build... /Download all logs as zip
Starting: Get sources
Syncing repository: PFGAlex2016 (Git)
Starting clone
Checking out 505949fd0a99f634e85a95fdb9309e75c0dbbf49 to C:\a\1\s
Checking out branch refs/heads/master for repository PFGAlex2016 at commit 505949fd0a99f634e85a95fdb9309e75c0dbbf49
Running tasks
Starting task: Azure Deployment:Create Or Update Resource Group action on $(azure.deploy.resourceGroup)
Executing the powershell script: C:\IRVMS\services\Vms\TaskAgentProvisioner\Tools\agent\1.98.1\tasks\AzureResourceGroupDeployment\1.0.70\Deploy.ps1
Looking for Azure PowerShell module at C:\Program Files\Microsoft SDKs\Azure\PowerShell\Azure\PowerShell\Azure.psd1
AzurePSModuleVersion: 1.0.2
Get-ServiceEndpoint -Name 40120d91-ad80-4edf-aa5a-ca8a741dad7e -Context Microsoft.TeamFoundation.DistributedTask.Agent.Worker.Common.TaskContext
tenantId=6a20001-67f5-419a-8692-baf4e0957fa1
subscriptionId=6a20001-67f5-419a-8692-baf4e0957fa1
azureSubscriptionName= Azure Pass
Add-AzureRmAccount -ServicePrincipal -Tenant **** -Credential System.Management.Automation.PSCredential
Select-AzureRmSubscription -SubscriptionId 6a20001-67f5-419a-8692-baf4e0957fa1 -tenantId ****
Created resource group 'ARMmedtranstest'
Creating resource group deployment with name apimedtranstest-deploy-20160419-1055
Creating resource group deployment with name ARMmedtranstest
Successfully created resource group deployment with name 'ARMmedtranstest'
Finishing task: AzureResourceGroupDeployment
Finishing Build
Worker Worker-6f53d8a1-25a1-4237-8dc9-6baff4ee1fbc finished running job 6f53d8a1-25a1-4237-8dc9-6baff4ee1fbc
  
```

Fig. 125 Consola VSTS con el resultado de la construcción de la aplicación

5.2.2 Creación recursos de grupo de recursos en Microsoft Azure desde VSTS

En esta Sección se describen los pasos que hay que seguir para crear los recursos en un grupo de recursos Azure desde VSTS. Para ello se utiliza un agente hospedado de VSTS al que se le encarga la tarea que cree un grupo de recursos Microsoft Azure en base a una plantilla JSON, donde se conecta gracias a un servicio punto de conexión a la suscripción Microsoft Azure (ver Sección 5.1.6):

- Desde la pestaña de BUILD de VSTS se pulsa “+” para definir una nueva definición de construcción. Se comprueba que dicha definición aparece bajo el epígrafe “*Build definitions*”.
- Se añade un paso de construcción pulsando “*Add build step...*” y, de las opciones de construcción disponibles, se selecciona “*Azure Resource Group Deployment*” y se completa.
- Tomando como ejemplo el PFG, se han configurado los siguientes parámetros tal y como muestra la Fig. 126:
 - Se indica el tipo de conexión a la suscripción de Azure. En el campo “*Azure Connection Type*” se selecciona “*Azure Resource Manager*” (la otra opción es “*Azure Clasic*”).
 - Se indica el nombre del punto de conexión que se ha creado a la suscripción Azure (ver Sección 5.1.6), es decir, “*Azure Pass(SPN)*”.

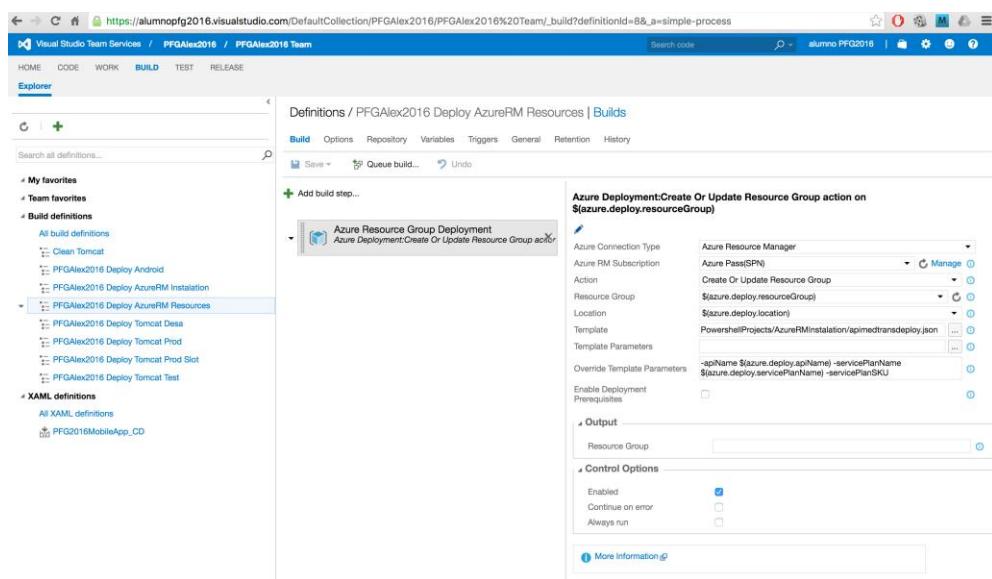


Fig. 126 Definición de la compilación que crea un grupo de recursos Azure desde VSTS

- Se indica la ubicación en el repositorio de la plantilla JSON. En el campo "*Template*" se especifica la plantilla "*apimedtransdeploy.json*"³⁰. Esta plantilla contiene la especificación del despliegue de los recursos (plan de servicio de aplicaciones, servicio de aplicaciones, servicio de automatización y servicio de almacenamiento) que se desea efectuar en un determinado grupo de recursos de una suscripción Azure. En el PFG se determina la suscripción Azure (en "*Azure RM Subscription*") es "*Azure Pass(SPN)*", que el grupo de recursos (en "*Resource Group*") es "*ARMmedtranstest*" y que la localización (en "*Location*") es "*West Europe*" (ver Fig. 127).

```

"$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
"contentVersion": "1.0.0.0",
"parameters": {
  "apiName": {
    "type": "string",
    "defaultValue": "apimedtrans"
  },
  "slotName": {
    "type": "string",
    "defaultValue": "stage"
  },
  "instancesCount": {
    "type": "int",
    "maxValue": 10,
    "defaultValue": 1
  },
  "servicePlanName": {
    "type": "string",
    "minLength": 1,
    "defaultValue": "ServPlan"
  },
  "servicePlanSKU": {
    "type": "string",
    "allowedValues": [
      "Free",
      "Shared",
      "Basic",
      "Standard"
    ],
    "defaultValue": "Free"
  },
  "servicePlanWorkerSize": {
    "type": "string",
    "allowedValues": [
      "Small",
      "Medium",
      "Large"
    ],
    "defaultValue": "Medium"
  }
}
  
```

Fig. 127 Extracto script JSON "apimedtrans.json"

³⁰ El fuente de la plantilla está accesible desde el repositorio, en particular de ruta *PowerShellProjects/AzureRMInstalation* de la rama master y en la misma ruta del CD que acompaña a esta memoria

- La plantilla "apimedtransdeploy.json" acepta varios parámetros, tales como el nombre del servicio de aplicación (en este caso el API app), etc. que se especifican en "Override Template Parameters". Los parámetros de la plantilla y la mayoría de los campos de la ventana se pueden declarar como variables en la pestaña "**Variables**"(ver Fig. 128).

Name	Value	Allow at Queue Time
system.collectionid	ab33b9c8-4022-4f07-9e72-d365233fc95d	<input type="checkbox"/>
system.teamProject	PFGAlex2016	<input type="checkbox"/>
system.definitionid	8	<input type="checkbox"/>
system.debug	false	<input checked="" type="checkbox"/>
azure.deploy.apiName	apimedtranstest	<input type="checkbox"/>
azure.deploy.storageAccountName	stormedtranstest	<input type="checkbox"/>
azure.deploy.servicePlanSKU	Standard	<input type="checkbox"/>
azure.deploy.servicePlanName	servmedtranstest	<input type="checkbox"/>
azure.deploy.resourceGroup	ARMmedtranstest	<input type="checkbox"/>
azure.deploy.location	West Europe	<input type="checkbox"/>

Fig. 128 Variables de la compilación despliegue recursos de grupo recursos Azure

La ejecución de esta definición de construcción que crea el grupo de recursos en Azure puede ser manual o automática (cada vez que se produzca un cambio en los fuentes de la rama seleccionada del repositorio si se ha activado Integración Continua). Para más detalles, ver Sección 5.2.1.

Si todo ha ido bien, en el portal de Azure aparece el grupo de recursos con sus correspondientes recursos (ver Fig. 129).

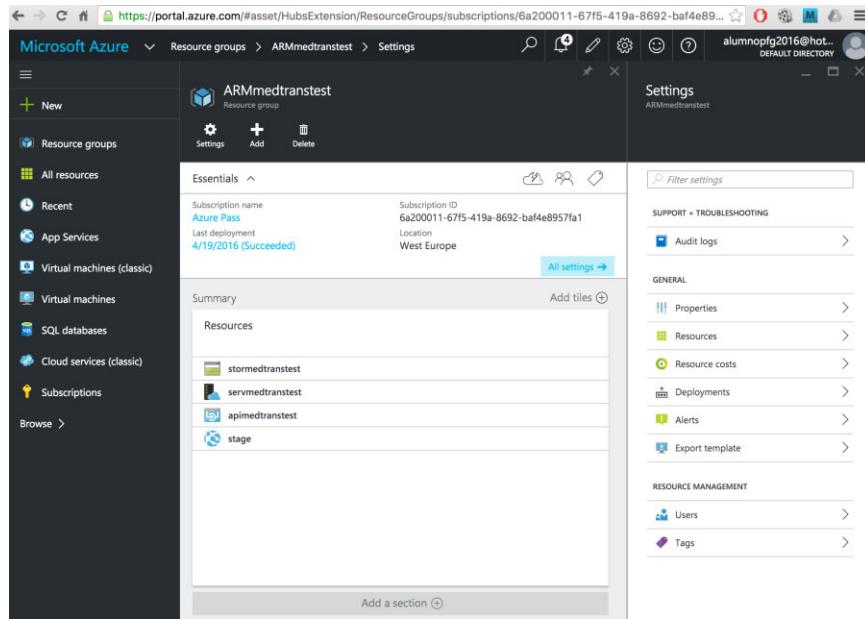


Fig. 129 Aspecto y grupo recursos creado desde VSTS en plataforma Microsoft Azure

5.2.3 Construcción y copia de un API en Microsoft Azure desde VSTS

En esta sección se describen los pasos que hay que seguir para, desde VSTS, construir y copiar en el servidor Microsoft Azure un API. Como el API ha sido desarrollado en Java, se pueden utilizar los agentes de compilación hospedados. A este agente se le encargan las tareas de compilación con Maven (ver Sección 4.1.1) y se copia en el servidor correspondiente con *cURL*³¹ (es una herramienta de línea de comando y biblioteca para transferencia de datos utilizando sintaxis URL y soportando protocolos como FTP, HTTP, HTTPS, etc.). El resultado de la construcción es un fichero WA, es decir, un fichero que se utiliza para distribuir una colección de JSPs, Servlets Java, Clases Java, ficheros XML, páginas estáticas HTML y otros recursos que junto constituyen una aplicación Web. Este WAR se copia en la carpeta **"*/site/wwwroot/webapps*"** con el nombre **"root.war"** para que Tomcat³² lo despliegue automáticamente:

³¹ <https://curl.haxx.se/>

³² <http://tomcat.apache.org/>

- Desde la pestaña de BUILD de VSTS se pulsa “+” para definir una nueva definición de construcción. Se comprueba que dicha definición aparece bajo el epígrafe “**Build definitions**”. Como ejemplo, se va a definir la construcción “**PFGAlex2016 Deploy Tomcat Prod**” del PFG.
- Se añade un paso de construcción pulsando “**Add build step...**” y, de las opciones de construcción disponibles, se selecciona “**Maven**” y se completa. En el caso de PFG se ha completado como se muestra en Fig. 130.
 - Se indica la ubicación en el repositorio del fichero de configuración de Maven. En el campo “*Maven POM*” se especifica el fichero “*pom.xml*”³³ (ver 4.1.1).
 - También se puede ver en la Fig. 130 que se ha especificado “**-X package war:war**” como “**Goals**”. El ciclo de vida de Maven está compuesto de las tareas *compile*, *test*, *package*, *install* y *deploy*. Este orden es importante ya que lleva a cabo las tareas previas hasta que llega a la objetivo. Por tanto, compilara y ejecutará las pruebas antes de empaquetar y generar el fichero WAR.

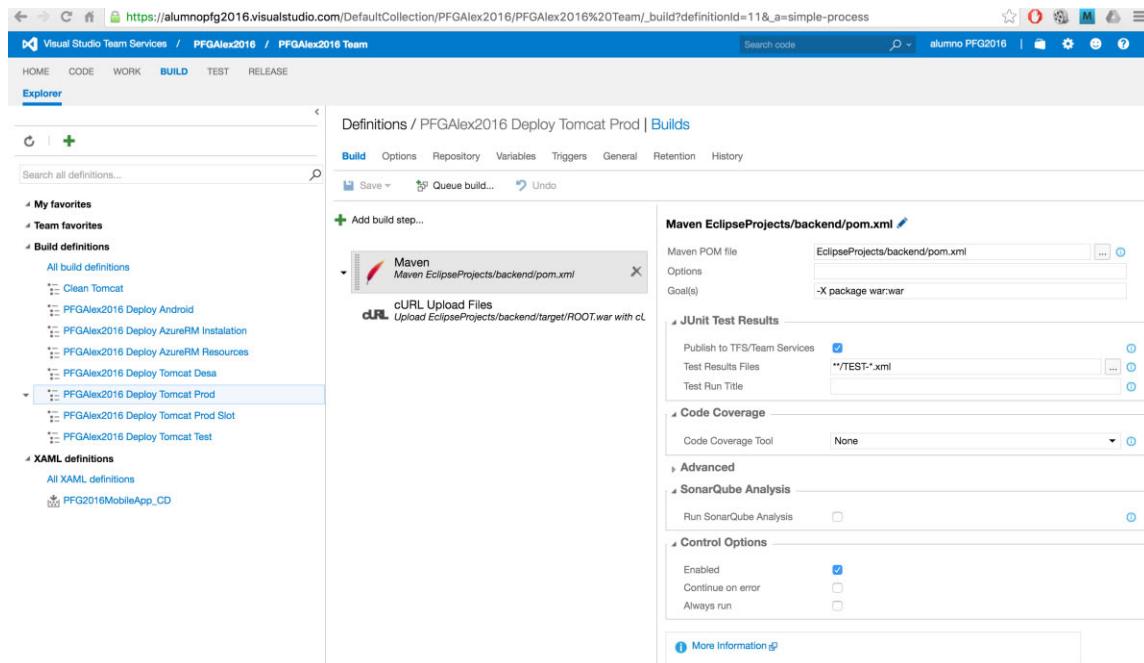


Fig. 130 Definición compilación despliegue API: 1er. paso compilación con Maven

³³ El fuente de la plantilla está accesible desde el repositorio, en particular de ruta *EclipseProjects/backend* de la rama master y en la misma ruta del CD que acompaña a esta memoria

- Se añade otro paso de construcción pulsando "*Add build step...*" y, de las opciones de construcción disponibles, se selecciona "**cURL**". En el caso de PFG se ha completado como se muestra en Fig. 131. Este paso copia el WAR generado en el paso anterior a la URL que se le indique. La mayoría de los campos de la ventana se pueden declarar como variables en la pestaña "**Variables**" (ver Fig. 132). Para el PFG, se especifica "`ftp://waws-prod-am2-065.ftp.azurewebsites.windows.net`" como URL y "`apimedtrans\amrodriguez`" como "**Username**".

The screenshot shows the "Build" tab of a build definition named "PFGAlex2016 Deploy Tomcat Prod". On the left, the navigation pane shows various build definitions, including "PFGAlex2016 Deploy Tomcat Prod" which is selected. In the main area, under the "Build steps" section, there is a "cURL Upload Files" step. The configuration for this step includes:

- File:** EclipseProjects/backend/target/ROOT.war
- Username:** \${azure.ftp.userName}
- Password:** \${azure.ftp.password}
- URL:** \${azure.ftp.url}
- Optional Arguments:** -Q "+CWD site/wwwroot/webapps"

The "Advanced" section contains "Control Options" with checkboxes for "Enabled" (checked), "Continue on error" (unchecked), and "Always run" (unchecked).

Fig. 131 Definición copia API con *cURL*

The screenshot shows the "Variables" tab of the same build definition. It lists predefined variables and allows adding new ones. The variables listed are:

Name	Value	Allow at Queue Time
system.collectionId	ab33b9c8-4022-4f07-9e72-d365233fc95d	<input type="checkbox"/>
system.teamProject	PFGAlex2016	<input type="checkbox"/>
system.definitionId	11	<input type="checkbox"/>
system.debug	false	<input checked="" type="checkbox"/>
azure.ftp.url	ftp://waws-prod-am2-065.ftp.azurewebsites.windows.	<input type="checkbox"/>
azure.ftp.userName	apimedtrans\amrodriguez	<input type="checkbox"/>
azure.ftp.password	*****	<input type="checkbox"/>

A "Add variable" button is visible at the bottom left of the table.

Fig. 132 Variables para la copia API

- La ejecución de esta definición de construcción que copia el WAR en el servidor donde está ejecutándose el contenedor Web Tomcat puede ser manual o automática (cada vez que se produzca un cambio en los fuentes de la rama seleccionada del repositorio si se ha activado integración continua). Para más detalles, ver 5.2.1.
- Si todo ha ido bien, en el portal de Microsoft Azure aparece el API "**"apimedtrans"**" (ver Fig. 133).

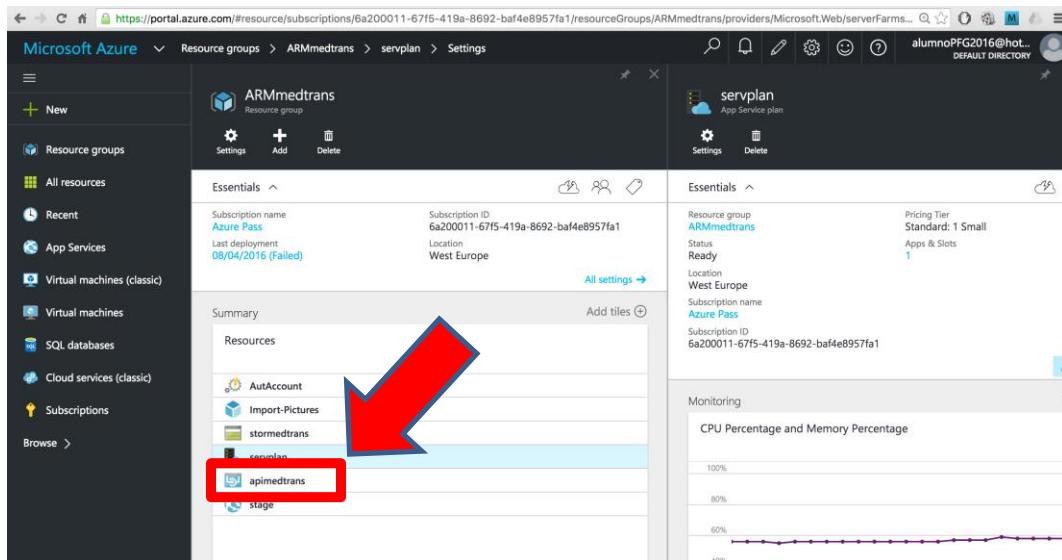


Fig. 133 API desplegado en Azure desde VSTS

5.2.4 Construcción aplicación Android desde VSTS

A la hora de construir la aplicación Android del caso de estudio desde VSTS, se ha utilizado un agente privado de VSTS, al que se le encargan las tareas de compilación, test e instalación de la aplicación en un AVD con Gradle (ver Sección 4.2.1) y después genera un APK³⁴ firmado y alineado:

- Desde la pestaña de BUILD de VSTS se pulsa "+" para definir una nueva definición de construcción. Se comprueba que dicha definición aparece bajo el epígrafe "*Build definitions*". Como ejemplo, se va a definir la construcción "**PFGAlex2016 Deploy Android**" del PFG.

³⁴ APK (*Android Application Package*): este archivo es un paquete para el sistema operativo Android. Se usa para distribuir e instalar componentes empaquetados para la plataforma Android para teléfonos inteligentes y tabletas.

- Se añade un paso de construcción pulsando “Add build step...” y, de las opciones de construcción disponibles, se selecciona “Android Build” y se completa. En el caso de PFG se ha completado como se muestra en Fig. 134:

- Se indica la ubicación en el repositorio del fichero de configuración de Gradle. En el campo “Location of Gradle Wrapper” se especifica “gradlew.bat”³⁵ (ver Sección 4.2.1).
- Para que el APK se instale en el AVD hay que especificar como “**Gradle Arguments**”:

```
installDebug && adb shell am start -n
"net.azurewebsites.apimedtrans/net.azurewebsites.apimedtrans.ItemListActivity"
-a android.intent.action.MAIN -c android.intent.category.LAUNCHER
```

- Finalmente, se especifica “Nexus_10_API23” como “AVD”. De esta forma, el agente privado después de compilar y pasar los test de la aplicación, instala la aplicación en este dispositivo virtual (que tiene que estar en ejecución en la máquina donde se está ejecutando el agente privado).

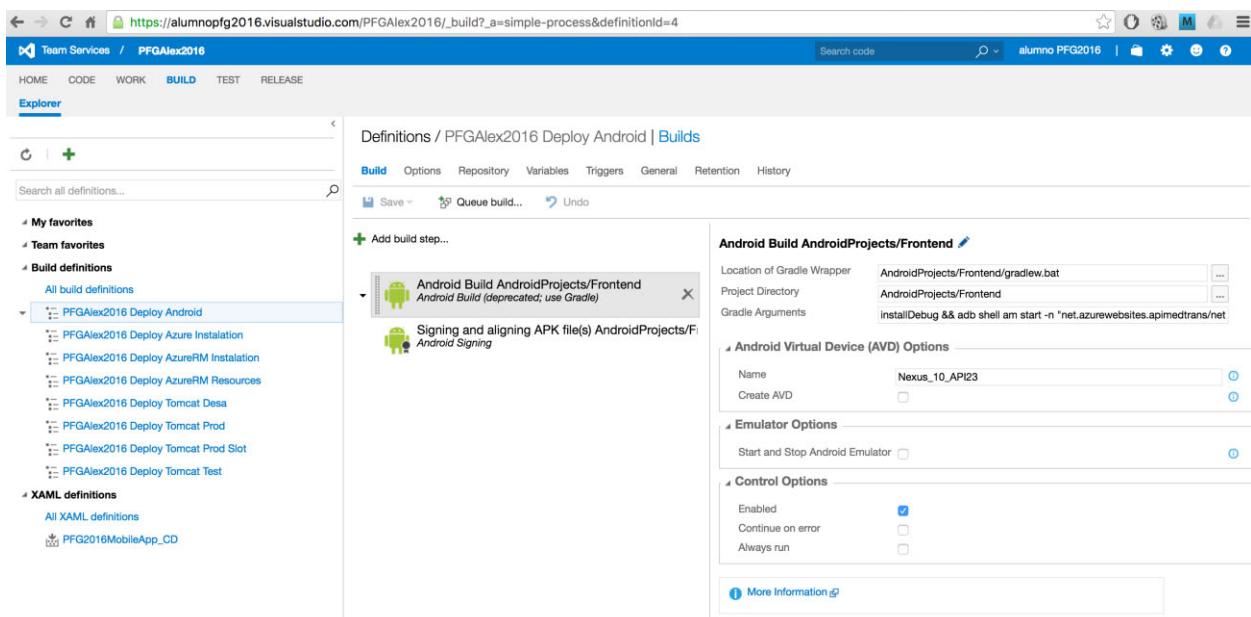


Fig. 134 Definición compilación construcción Android: 1er. paso compilación Gradle

³⁵ El fuente de la plantilla está accesible desde el repositorio, en particular de ruta *AndroidProjects/frontend* de la rama master y en la misma ruta del CD que acompaña a esta memoria

- Se añade otro paso de construcción pulsando “*Add build step...*” y, de las opciones de construcción disponibles, se selecciona “*Signing and Aligning APK Files*”. Este paso firma y alinea el APK generado en el paso anterior. Por una parte, las aplicaciones se firman por varios motivos: como medida de seguridad, como requisito de garantía, para poder distribuir e instalar nuestra aplicación sin problemas, para que de esta forma sólo nosotros podamos modificar y actualizar nuestra aplicación y porque es un requisito que nos pide el Android Market para subir nuestras aplicaciones. Por otra parte, las APK se alinean para reducir la cantidad de RAM que consume y que consiste en asegurarse que todos los datos no comprimidos empiecen por un determinado alineamiento de bytes relativo al comienzo del fichero. En el caso de PFG se ha completado como se muestra en Fig. 135.

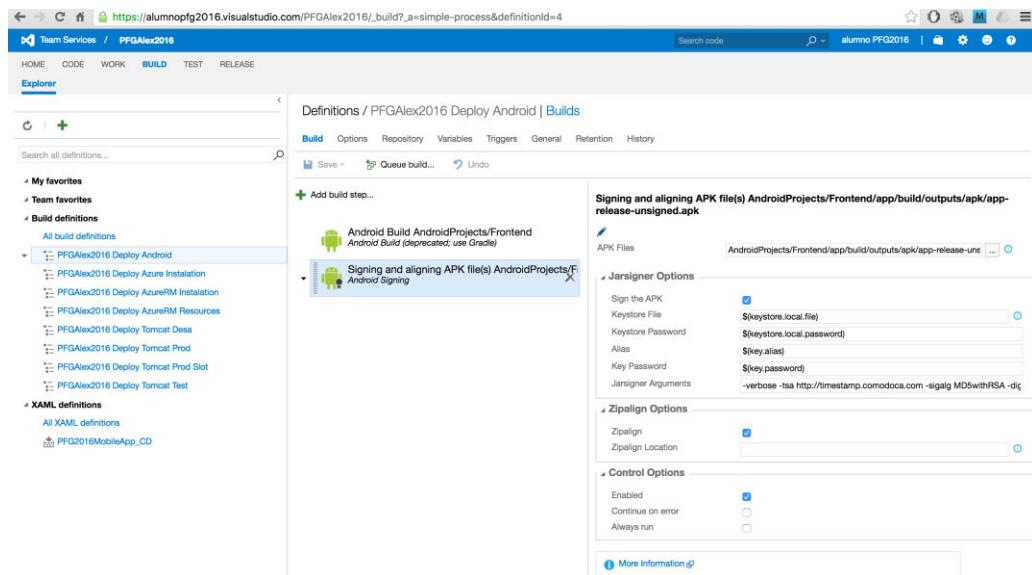


Fig. 135 Definición compilación construcción Android: 2º paso firma

- La ejecución de esta definición de construcción que instala el APK en el AVD donde está ejecutándose el agente privado puede ser manual o automática (cada vez que se produzca un cambio en los fuentes de la rama seleccionada del repositorio si se ha activado Integración Continua). Para más detalles, ver Sección 5.2.1.
- Como se va a utilizar un agente de compilación privado, hay tenerlo en ejecución en el equipo donde se haya instalado (ver Sección 5.1.5). En este caso además hay que tener en ejecución el AVD que se haya especificado.
- Si todo ha ido bien, la aplicación aparecerá instalada en el AVD para poder ser probada. En la Fig. 136 se puede ver el agente privado (ventana de comandos de Windows) y el AVD con la aplicación instalada y en ejecución.

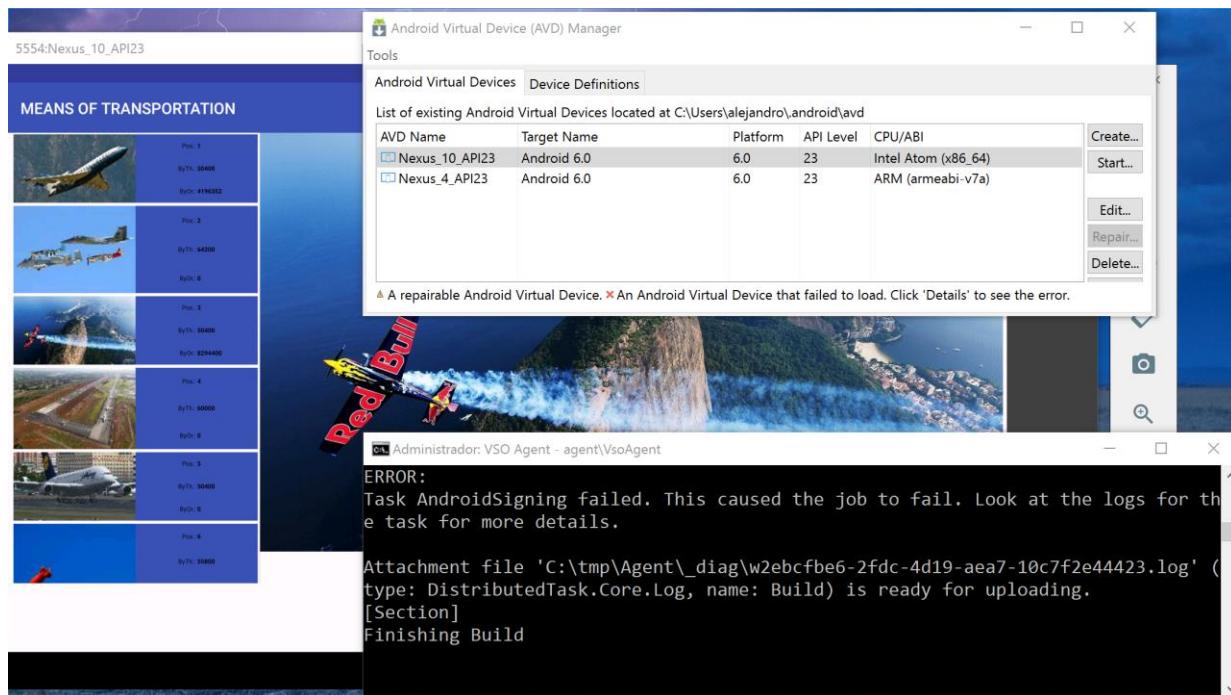


Fig. 136 Android construido con agente privado e instalado en AVD desde VSTS

5.2.5 Ejecución de test de carga Apache JMeter en VSTS

En esta Sección se describen los pasos que hay que seguir para diseñar y lanzar un test de carga. Estos test de carga se suelen ejecutar en el entorno de test como condición previa al despliegue de la aplicación en producción. En este PFG no se ha definido ningún requisito específico a este respecto.

Para diseñar el test se utiliza la herramienta Apache JMeter³⁶ mientras que su ejecución se realiza desde VSTS. Por otra parte, todas las cuentas VSTS tienen 20.000 minutos de usuario virtual gratis para lanzar test cada mes. Por consiguiente, si el test de carga consta de 250 usuarios virtuales concurrentes, seremos capaces de ejecutar este test un total de 60 minutos al mes.

En este PFG se quiere ver cómo responde la aplicación cuando 100 usuarios ejecutan concurrente y repetidamente durante 1 minuto la petición HTTP "`http://mediotransporte/rangefrom/FIRST_ELEMENT`".

³⁶ <http://jmeter.apache.org/>

- En primer lugar, se diseña un test de carga utilizando la aplicación *JMeter*. En primer lugar se da de alta un “*Thread Group*” como se muestra en la Fig. 137. Este test en particular va a lanzar 100 hilos de ejecución.

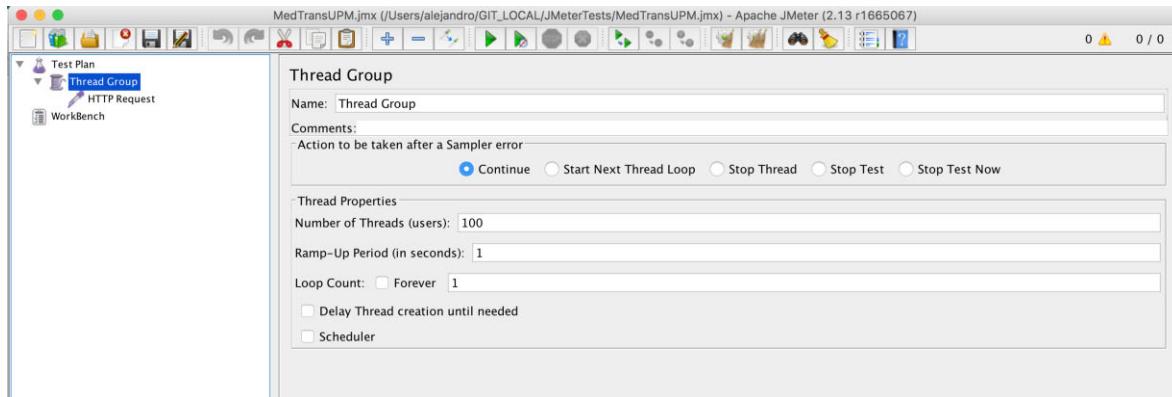


Fig. 137 Configuración test de carga con JMeter: Thread Group

- En segundo lugar, se da de alta un “*HTTP Request*” asociado a “*HTTP Group*” como se muestra en la Fig. 138. Este test en particular va a lanzar la petición “`http://mediotransporte/rangefrom/FIRST_ELEMENT`”. También se pueden dar de alta parámetros. En este caso se crea el parámetro *nTokens* y se le da el valor de 5.

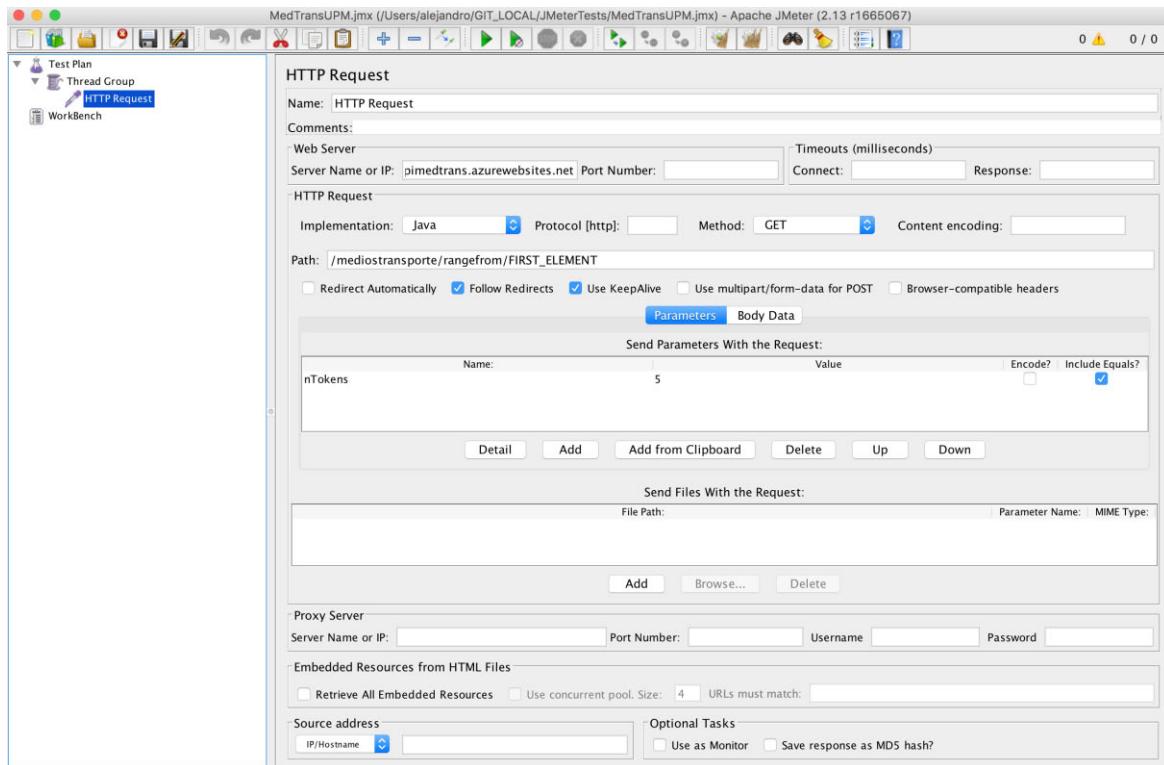


Fig. 138 Configuración test de carga con JMeter: HTTP Request

- Guardamos el test *JMeter* con el nombre “*MedTransUPM.jmx*” y lo cargamos en VSTS. Para ello se selecciona “*New*” y se selecciona “*Apache JMeter test*” (ver Fig. 139).

The screenshot shows the Visual Studio Team Services interface under the TEST tab. In the left sidebar, 'Load tests' is selected, and 'MedTransUPM.jmx' is highlighted. The main area displays a table of load test runs for 'MedTransUPM.jmx'. The columns include Run Id, Load Test, Run Type, Time Completed, Duration(sec), User Count, Requested By, and Location. The table lists 19 runs from 17/04/2016, mostly with 100 users and 60 seconds duration, located in East US 2 (Virginia). A message at the top says, "We've added support for running Apache JMeter based tests in the cloud. Here's how to get started."

Run Id	Load Test	Run Type	Time Completed	Duration(sec)	User Count	Requested By	Location
19	MedTransUPM.jmx	Apache JMeter	17/04/2016	60	100	alumno PFG2016	East US 2 (Virginia)
35	MedTransUPM.jmx	Apache JMeter	17/04/2016	60	-	alumno PFG2016	
33	MedTransUPM.jmx	Apache JMeter	17/04/2016	60	-	alumno PFG2016	
32	MedTransUPM.jmx	Apache JMeter	17/04/2016	60	-	alumno PFG2016	
31	MedTransUPM.jmx	Apache JMeter	17/04/2016	60	-	alumno PFG2016	
30	MedTransUPM.jmx	Apache JMeter	17/04/2016	15	100	alumno PFG2016	East US 2 (Virginia)
29	MedTransUPM.jmx	Apache JMeter	17/04/2016	15	100	alumno PFG2016	East US 2 (Virginia)
28	MedTransUPM.jmx	Apache JMeter	17/04/2016	15	3	alumno PFG2016	East US 2 (Virginia)
27	MedTransUPM.jmx	Apache JMeter	17/04/2016	60	-	alumno PFG2016	East US 2 (Virginia)
26	MedTransUPM.jmx	Apache JMeter	17/04/2016	21	3	alumno PFG2016	East US 2 (Virginia)
25	MedTransUPM.jmx	Apache JMeter	17/04/2016	15	3	alumno PFG2016	East US 2 (Virginia)
24	MedTransUPM.jmx	Apache JMeter	17/04/2016	15	2	alumno PFG2016	East US 2 (Virginia)
23	MedTransUPM.jmx	Apache JMeter	17/04/2016	15	1	alumno PFG2016	East US 2 (Virginia)
22	MedTransUPM.jmx	Apache JMeter	17/04/2016	60	5	alumno PFG2016	East US 2 (Virginia)

Fig. 139 Alta del test de carga Apache JMeter desde la pestaña TEST de VSTS

- Se ejecuta el test “*MedTransUPM.jmx*” y se configura tal y como se ve en la Fig. 140.

The screenshot shows the configuration of the 'Apache JMeter test' in VSTS. The 'Test script' field contains 'MedTransUPM.jmx'. The 'Supporting files' field has 'Choose Files' and 'No file chosen'. The 'Number of agents' is set to 1. The 'Load duration' is set to 1 minute. The 'Load location' is set to East US 2 (Virginia). At the bottom, there is a 'Run Test' button.

Fig. 140 Ejecución del test de caga Apache JMeter desde la pestaña TEST de VSTS

- Cuando el test ha acabado, se puede ver un resumen del rendimiento de la aplicación en la pestaña “*Summary*” (ver Fig. 141). Aquí se muestran las principales métricas tales como:
 - Tiempo de respuesta medio:** es el tiempo medio que le lleva a la aplicación responder a una petición. Es una de las métricas clave para medir el rendimiento de la aplicación. La mayoría de las aplicaciones se comportan bien con un único usuario accediendo a la aplicación pero su rendimiento suele empeorar cuando la aplicación

está bajo carga. Esto puede suceder porque los recursos (CPU, SQL, etc.) están al límite de su capacidad teniendo como consecuencia un mayor tiempo de respuesta.

- **Carga de usuario:** es el pico de carga de usuario conseguido para el test de carga.
- **Peticiones por segundo:** es una métrica de rendimiento que mide el número de peticiones hechas al servidor por segundo. El RPS (*Requests Per Second*) depende del tamaño de las respuestas y de las peticiones, del tiempo de respuesta, de la carga de usuario, etc.
- **Peticiones fallidas:** es el número de peticiones que han fallado bien porque la aplicación no responde o bien por errores de conectividad. Llegado este punto la aplicación puede que empiece a bajar su rendimiento y la aplicación pare de aceptar nuevas peticiones.

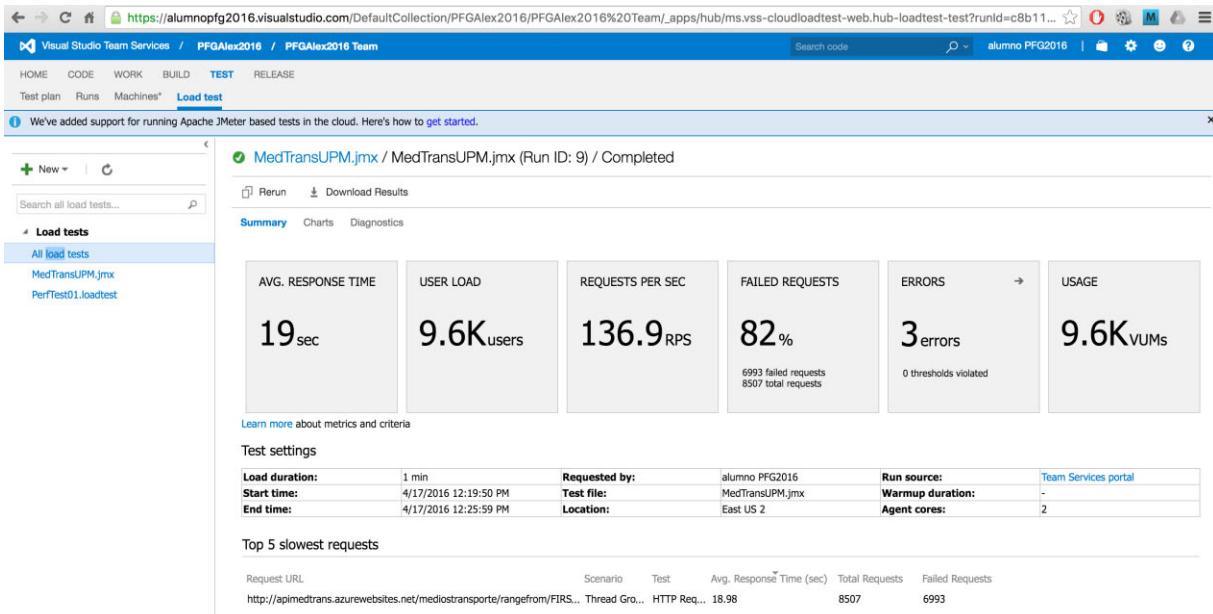


Fig. 141 Resultados del test de carga desde la pestaña TEST de VSTS

- En la pestaña “*Charts*” se puede consultar una representación gráfica del resultado de los tests a lo largo del tiempo (ver Fig. 142).

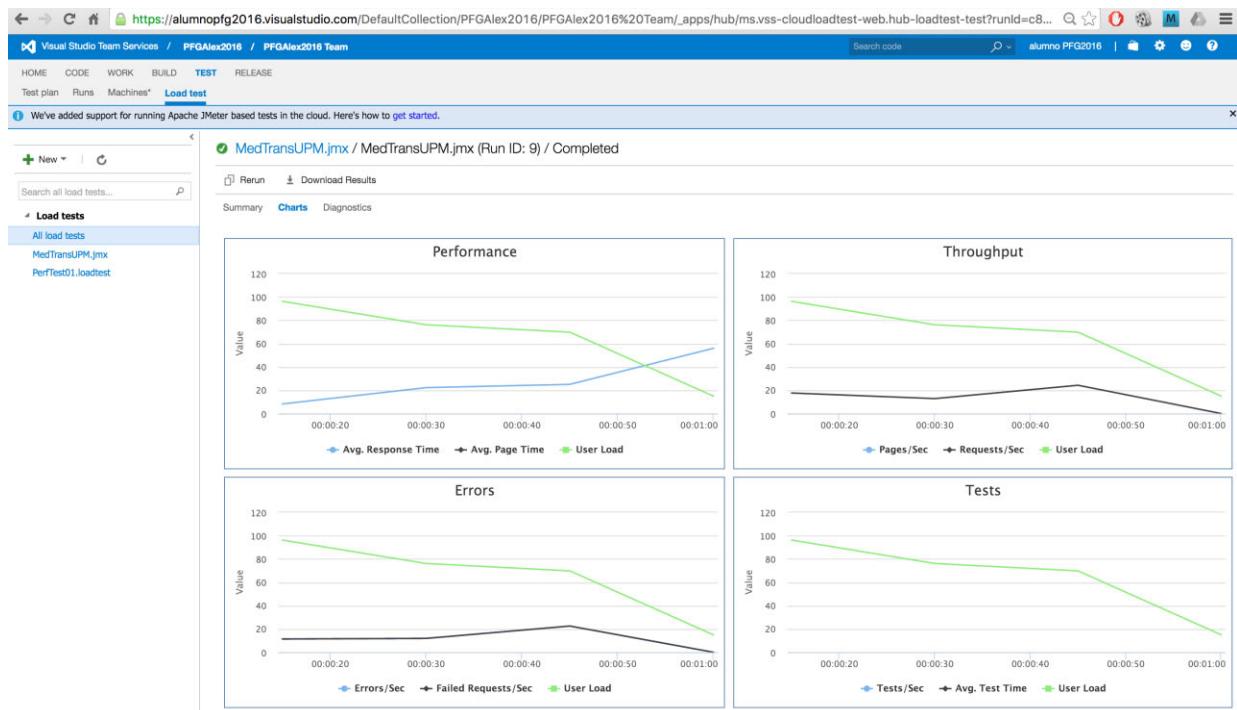


Fig. 142 Representación gráfica resultados test carga en pestaña TEST de VSTS

6 Conclusiones y Trabajos Futuros

Este capítulo presenta y analiza las principales contribuciones de este PFG. También presenta los trabajos futuros que se pueden llevar a cabo para continuar este estudio y extender los resultados que ya han sido obtenidos.

6.1 Conclusiones

El objetivo general del PFG consistía en conocer los servicios avanzados de la PaaS Microsoft Azure y los servicios de administración del ciclo de vida de la aplicación con VSTS, así como el desarrollo de aplicaciones basado en DevOps. Este objetivo se ha alcanzado gracias a un estudio exhaustivo y puesta en práctica de dichos servicios y enfoque de desarrollo a través del desarrollo de un caso de estudio en DevOps. Concretamente, se ha desarrollado una aplicación para la gestión de fotos en la nube, compuesta por un API REST en Java y un cliente Android.

El desarrollo se ha realizado en apenas 4 meses y con un coste mensual inferior a los 80 Euros/Mes (prescindiendo del espacio de implementación o ensayo, el coste bajaría a alrededor de 10 Euros/Mes). Como resultado de este desarrollo, a día de hoy el API REST se aloja en Microsoft Azure y permite a los clientes Android acceder a un servicio de almacenamiento escalable manualmente. Por otra parte, la aplicación se ha realizado administrando su ciclo de vida con VSTS: desarrollo ágil de software; flujo de trabajo (desarrollar, desplegar, monitorizar y aprender) y buenas prácticas (automatizar todo, control de versiones e integración continua) DevOps e integración con Microsoft Azure (posibilita integración y entrega continua desde VSTS).

Personalmente, partiendo de la premisa que no conocía Microsoft Azure ni otras plataformas *Cloud* ni metodologías ágiles ni DevOps, el haber conseguido los objetivos planteados se convierte en uno de los principales avales de la plataforma de aplicaciones Microsoft Azure, de la plataforma de administración del ciclo de vida de aplicaciones VSTS y de la perfecta simbiosis entre ambas. Por ello puedo afirmar que la curva de aprendizaje es asumible, pero se ha de tener en cuenta que se puede ver dificultada tanto por el hecho que cada día van surgiendo servicios nuevos o migrados de la vieja plataforma, como por el hecho de que existe una cantidad de documentación abrumadora al respecto (Internet y libros). También se puede afirmar que el desarrollo ágil de software y DevOps incrementan la velocidad de desarrollo y su puesta en producción.

Finalmente, quisiera resaltar que cuando una empresa solicita gente con experiencia, no se refiere a gente que lleva toda su vida haciendo lo mismo, sino que se refiere a gente que ha afrontado numerosos retos laborales de diversa índole. Pues bien, en este sentido, considero que este proyecto ha incrementado mi experiencia. Ha supuesto un gran reto personal en

todos los sentidos. En el mismo se abordan tecnologías y metodologías que o bien eran nuevas para mí o bien tenía escasos conocimientos. Afortunadamente, además de contar con todo el apoyo de mis tutoras, existe Internet y los *freaks* de *stackoverflow*. Está claro que sin Internet la tecnología informática no crecería al ritmo que lo hace y, sin los *freaks*, el proyecto estaría parado por semanas al primer *bug de turno*...

6.2 Trabajos Futuros

Se estima que la administración del ciclo de vida de la aplicación con VSTS ha sido idónea y ha favorecido la consecución de los objetivos planteados y favorecería cualesquiera otros que se planteen. En cualquier caso, en este proyecto **no se han explotado los servicios ofertados por VSTS relacionados con test automatizados y con la liberación de versiones** los cuales permitirían aplicar la **integración continua también a los entornos de test y producción** y no solo al entorno de desarrollo. Por tanto, trabajos futuros deberían ir en la línea de aplicar DevOps puro, es decir, no solo una parte sino todas las buenas prácticas que recomienda y no solo al entorno de desarrollo sino a los entornos de desarrollo y producción también.

También se podrían realizar mejoras con respecto a los atributos de calidad tanto para el servicio API REST como para el cliente. Las mejoras sugeridas pueden consistir en la utilización de algún servicio adicional de Microsoft Azure o en la implementación de algún patrón *cloud*.

- **Rendimiento y disponibilidad:** para mejorar el rendimiento y disponibilidad en caso incrementos de carga ser recomienda implementar el **patrón Queue-Based Load Leveling** (ver Fig. 143).

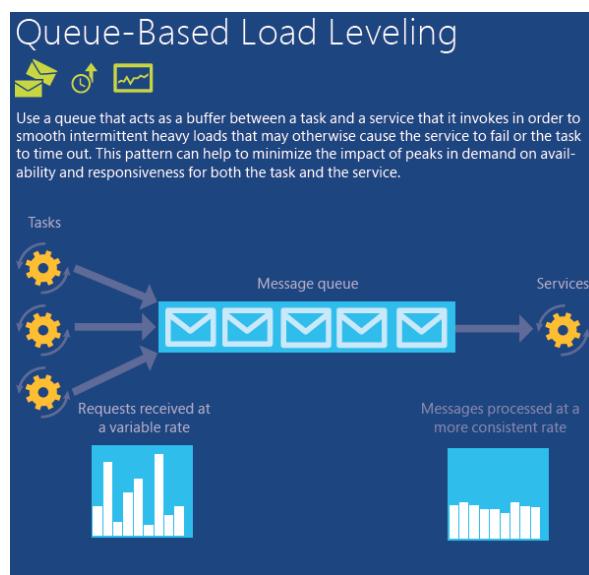


Fig. 143 Patrón Colas [13]

Otra sugerencia para mejorar rendimiento y disponibilidad y evitar que el servicio API REST se convierta en un cuello de botella es **utilizar en el cliente la librería *Android client SDK for Mobile Apps*** y acceder directamente a los servicios de la cuenta de almacenamiento de Microsoft Azure.

Una última sugerencia para mejorar rendimiento y disponibilidad consiste en implementar el **patrón Caché-aside en el servidor** (ver Fig. 144) para reforzar la caché que se implementó en el cliente.

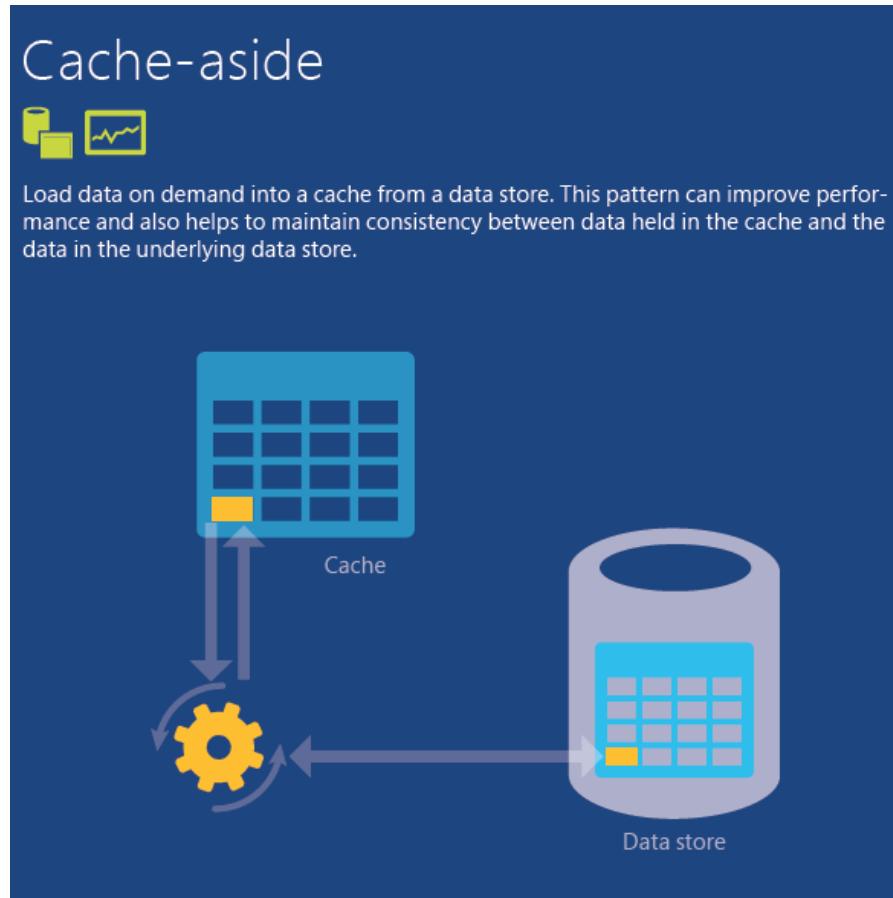


Fig. 144 Patrón Caché [13]

- **Monitorización:** para detectar problemas y tratar de mejorar la aplicación, se recomienda monitorizarla mediante el servicio ***Application Insights*** proporcionado por la plataforma Azure.
- **Mercado objetivo:** para mejorar las posibilidades de negocio es necesario ampliar los mercados objetivo de la aplicación, en particular se recomienda desarrollar versiones del frontend en **Windows Phone y iOS**, además de la ya existente en Android.

- **Seguridad:** gestionar los accesos al API mediante, por ejemplo, **Azure Active Directory (AD)**.
- **Resistencia:** para evitar que la estabilidad de la aplicación se vea afectada negativamente al producirse fallos temporales en las comunicaciones a través de Internet, se recomienda implementar el **patrón Retry** (ver Fig. 145) tanto en cliente como en servidor.

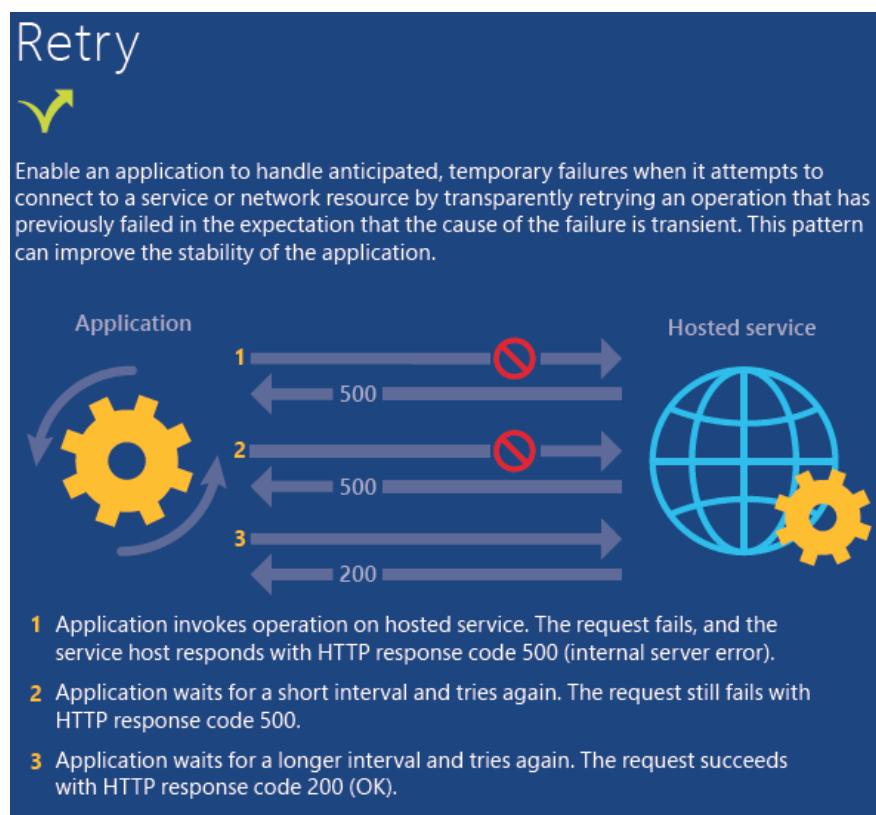


Fig. 145 Patrón Reintentar [13]

Bibliografía

- [1] Z. Mahmood, R. Puttini y T. Erl, *Cloud Computing: Concepts, Technology & Architecture*, Prentice-Hall, 2013.
- [2] L. Wang, R. Ranjan, J. Chen y B. Benatallah, *Cloud Computing: Methodology, Systems and Applications*, CRC Press, 2012.
- [3] N. Herskowitz, «Microsoft – the only vendor named a leader in Gartner Magic Quadrants for IaaS, Application PaaS, and Cloud Storage,» 22 Mayo 2015. [En línea]. Available: <http://azure.microsoft.com/blog/2015/05/22/microsoft-the-only-vendor-named-a-leader-in-gartner-magic-quadrants-for-iaas-application-paas-cloud-storage-and-hybrid/>. [Último acceso: 10 Junio 2015].
- [4] «Administración del ciclo de vida de las aplicaciones,» [En línea]. Available: <https://www.visualstudio.com/es-es/features/app-lifecycle-management-vs.aspx>. [Último acceso: 01 05 2016].
- [5] H. Bai, *Zen of Cloud. Learning Cloud Computing by Examples on Microsoft Azure*, Boca Raton: CRC Press, 2015.
- [6] M. Kavis, *Architecting the cloud - Design decisions for cloud computing service models*, Wiley, 2014.
- [7] R. Buyya, C. Vecchiola y T. Selvi, *Mastering Cloud Computing*, Elsevier, 2013.
- [8] M. Collier y R. Shahan, *Fundamentals of Azure*, Redmond, Whasington: Microsoft Corporation, 2015.
- [9] P. Mell y T. Grance, «NIST Cloud Computing Program,» 11 2011. [En línea]. Available: <http://www.nist.gov/itl/cloud/>. [Último acceso: 23 04 2016].
- [10] L. Bass, P. Clements y R. Kazman, *Software Architecture in Practice*, Third Edition, Septiembre: 25, 2012.
- [11] M. Masse, *REST API Design Rulebook*, O'Reilly Media, Inc., 2011.
- [12] K. Burns, *Windows Azure Websites Succintly*, Morrisville: Syncfusion, 2014.
- [13] «Microsoft Azure: Cloud Design Patterns,» [En línea]. Available: <https://azure.microsoft.com/es-es/documentation/infographics/cloud-design-patterns/>.
- [14] A. Homer, J. Sharp, L. Brader, M. Marumoto y S. Trent, *Cloud Design Patterns*, Microsoft Corporation, 2014.
- [15] «What is Windows Azure?,» [En línea]. Available: https://www.youtube.com/watch?v=poDRw_Xi3Aw.

- [16] «WHAT IS APPLICATION LIFECYCLE MANAGEMENT?», 2010. [En línea]. Available: http://www.davidchappell.com/writing/white_papers/What_is_ALM_v2.0--Chappell.pdf. [Último acceso: 2016 4 30].
- [17] «How to install and configure Azure PowerShell», 06 01 2016. [En línea]. Available: <https://azure.microsoft.com/en-us/documentation/articles/powershell-install-configure/>. [Último acceso: 22 04 2016].
- [18] «Azure subscription and service limits, quotas, and constraints», 19 4 2016. [En línea]. Available: <https://azure.microsoft.com/en-us/documentation/articles/azure-subscription-service-limits/#app-service-limits>. [Último acceso: 22 4 2016].
- [19] «DEV205x Architecting Microsoft Azure Solutions», [En línea]. Available: <https://www.edx.org/course>.
- [20] «Azure Storage Pricing», 2016. [En línea]. Available: <https://azure.microsoft.com/en-gb/pricing/details/storage/>. [Último acceso: 22 04 2016].
- [21] «Use Azure Resource Explorer to view and modify resources», 04 03 2016. [En línea]. Available: <https://azure.microsoft.com/en-gb/documentation/articles/resource-manager-resource-explorer/>. [Último acceso: 22 04 2016].
- [22] «Set up staging environments for web apps in Azure App Service», 09 03 2016. [En línea]. Available: <https://azure.microsoft.com/en-gb/documentation/articles/web-sites-staged-publishing/>. [Último acceso: 22 04 2016].
- [23] J. Annuzi, L. Darcey y S. Conder, *Introduction to Android. Android Development Fourth Edition*, Addison-Wesley, 2014.
- [24] «Wikipedia: Android», [En línea]. Available: [https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system)).
- [25] «Android vs iOS market share in 2015», [En línea]. Available: <https://deviceatlas.com/blog/android-vs-ios-market-share-2015>.
- [26] «Android: Platform Versions», [En línea]. Available: <https://developer.android.com/about/dashboards/index.html>.
- [27] «Android developpers blog», 7 4 2016. [En línea]. Available: <http://android-developers.blogspot.com.es/2016/04/android-studio-2-0.html>.
- [28] J. Rasmussen, *The Agile Samurai: How Agile Masters Deliver Great Software*, The Pragmatic Bookshelf, 2010.
- [29] «Manifiesto ágil», [En línea]. Available: <http://www.agilemanifesto.org/iso/es/>.
- [30] «Principios del manifiesto ágil», [En línea]. Available: <http://www.agilemanifesto.org/iso/es/principles.html>.
- [31] M. Ali Babar, A. W. Brown y I. Mistrik, *Agile Software Architecture*, Morgan Kaufmann, 2014.

- [32] «Wikimedia: Scrum,» [En línea]. Available: https://commons.wikimedia.org/wiki/File:Ficha_scrum.jpg.
- [33] «Use DevOps to Turn IT into a Strategic Weapon,» [En línea]. Available: <http://dev2ops.org/2012/09/use-devops-to-turn-it-into-a-strategic-weapon/>.
- [34] «Entre Dev y Ops: cosas sobre DevOps (2): En qué difiere de Agile,» [En línea]. Available: <http://www.entredevyops.es/posts/11-cosas-necesitas-saber-devops-2.html>.
- [35] S. Sharma, «Mobile DevOps - Trends and Challenges,» [En línea]. Available: <http://www.slideshare.net/sanjeev-sharma/dev-ops-for-mobile-modevtablet>.
- [36] T. Dykstra, R. Anderson y M. Watson, *Building Real-World Cloud Apps with Windows Azure*, Microsoft Corporation, 2014.
- [37] «Visual Studio,» 2016. [En línea]. Available: <https://www.visualstudio.com/>. [Último acceso: 30 04 2016].
- [38] M. Fowler, «Martin Fowler. Continuous Integration,» 2000. [En línea]. Available: <http://martinfowler.com/articles/continuousIntegration.html>. [Último acceso: 06 2016].
- [39] A. M. Paredes, «¿Qué es la Arquitectura de Software?,» 01 06 2015. [En línea]. Available: <https://elfrasco.github.io/2015/06/01/Que-es-la-Arquitectura-de-Software.html>.
- [40] «Arquitectura ágil,» 08 06 2015. [En línea]. Available: <https://elfrasco.github.io/2015/06/08/Arquitectura-Agil.html>.
- [41] «The Java EE 6 Tutorial. Building RESTful Web Services with JAX-RS,» 2016. [En línea]. Available: <http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html>. [Último acceso: 06 2016].
- [42] «Jersey. RESTful Web services in Java,» 2016. [En línea]. Available: <https://jersey.java.net/>. [Último acceso: 06 2016].
- [43] «Swagger,» 2016. [En línea]. Available: <http://swagger.io/>. [Último acceso: 06 2016].
- [44] «Swagger-Core Annotations 1.5.x,» [En línea]. Available: <https://github.com/swagger-api/swagger-core/wiki/Annotations-1.5.X>.
- [45] «Swagger UI,» [En línea]. Available: <http://swagger.io/swagger-ui/>. [Último acceso: 25 04 2016].
- [46] «Retrofit,» [En línea]. Available: <http://square.github.io/retrofit/>.
- [47] «Servicios adicionales VSTS,» [En línea]. Available: <https://azure.microsoft.com/es-es/pricing/details/visual-studio-team-services/>.
- [48] «Cuadrante Mágico de Gartner,» 6 01 2015. [En línea]. Available: <http://revista.helpdesktic.com/cuadrante-magico-de-gartner/>. [Último acceso: 25 05 2016].
- [49] «2016 será el año de maduración del Cloud Computing,» 15 03 2016. [En línea]. Available: <http://www.corporateit.cl/index.php/2016/03/15/2016-sera-el-ano-del-cloud-computing/>.

[Último acceso: 05 05 2016].

- [50] «Microsoft Investing Rs.1,400 Crore For Data Centers In India,» [En línea]. Available: <http://mspoweruser.com/microsoft-investing-rs-1400-crore-for-data-centers-in-india/>.
- [51] «State of Agile Survey 2011,» [En línea]. Available: https://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf.
- [52] «Agencia Española de Protección de Datos,» [En línea]. Available: https://www.agpd.es/portalwebAGPD/canaldocumentacion/publicaciones/common/Guias/GUIA_Cloud.pdf.
- [53] L. Bass, I. Weber y L. Zhu, *A Software Architect's Perspective*, Addison-Wesley Professional, 2015.

Glosario de términos

Active Directory: servicio de directorio de Microsoft en redes de ordenadores, que permite relacionar usuarios, permisos y políticas de acceso con los componentes de dichas redes.

Alinear APK: acción sobre el APK que reduce la cantidad de RAM que consume y que consiste en asegurarse que todos los datos no comprimidos empiecen por un determinado alineamiento de bytes relativo al comienzo del fichero. Se realiza utilizando el programa *zipalign*.

ALM (Aplicacion LifeCycle Management): es la gestión del ciclo de vida del producto de los programas de computadora, dividiéndola en tres áreas distintas (ver Fig. 146): gobierno (abarcá las decisiones y gestión de proyecto de una aplicación), desarrollo (proceso iterativo de crear la aplicación) y operaciones (trabajo requerido para ejecutar y gestionar la aplicación). Es decir, engloba no solo el ciclo de desarrollo del software (gestión de requisitos, arquitectura del software, programación, pruebas, mantenimiento), sino también integración continua, gestión de proyecto y gestión de versiones [16].

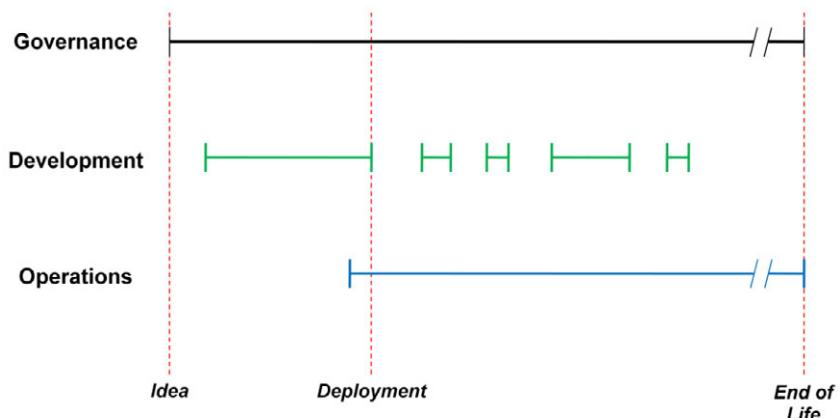


Fig. 146 Aspectos fundamentales de ALM [16]

API (Application Program Interface): un API expone un conjunto de servicios y funciones para facilitar las interacciones entre programas de ordenador y permitirles intercambiar información. Los programas cliente utilizan APIs para comunicarse con servicios Web. El estilo arquitectónico REST es generalmente aplicado al diseño de APIs en los servicios Web modernos.

APK (Android Application Package): un archivo con extensión apk es un paquete para el sistema operativo Android. Este formato es una variante del formato JAR de Java y se usa para distribuir e instalar componentes empaquetados para la plataforma Android para teléfonos

inteligentes y tabletas. Un archivo apk normalmente contiene: AndroidManifest.xml, classes.dex, resources.arsc, res (carpeta), META-INF (carpeta), lib (carpeta).

Alta disponibilidad: arquitectura que permite mantener en funcionamiento un servicio a pesar de las incidencias que puedan ocurrir. Suele ser una configuración software en clúster con una máquina activa y otra pasiva que entrará en funcionamiento si la activa deja de funcionar.

Apache JMeter: es un proyecto de Apache que implementa una herramienta para ejecutar pruebas de carga con el fin de analizar y medir el desempeño de una variedad de servicios, con énfasis en aplicaciones *Web*. JMeter puede ser usado también como una herramienta de pruebas unitarias para conexiones de bases de datos con JDBC, conexiones FTP, LDAP, Servicios *Web*, JMS, HTTP y conexiones TCP genéricas.

AVD (Android Virtual Device): es un configurador de emuladores que permite modelar un dispositivo real definiendo las opciones hardware y software que se van a emular en el emulador de Android. Tras instalar Android Studio o el SDK Android³⁷, la forma más fácil de crear un AVD es utilizar el AVD Manager que se localizado en el directorio <SDK>/tools/ o en Android Studio.

Backup: es una copia de un conjunto de ficheros o de datos, con el fin de poder restaurarlos en un futuro en caso de que los datos o ficheros originales se corrompan o se pierdan.

Backend / Frontend: *frontend* es la parte del software que interactúa con los usuarios y el *backend* es la parte que procesa la entrada desde el *frontend*. La separación del sistema en *frontends* y *backends* es un tipo de abstracción que ayuda a mantener las diferentes partes del sistema separadas. La idea general es que el *frontend* sea el responsable de recolectar los datos de entrada del usuario, que pueden ser de muchas y variadas formas, y los transforma ajustándolos a las especificaciones que demanda el *backend* para poder procesarlos, devolviendo generalmente una respuesta que el *frontend* recibe y expone al usuario de una forma entendible para este. La conexión del *frontend* y el *backend* es un tipo de interfaz.

BLOB / blob (Binary Large Object): es generalmente una imagen o un fichero de sonido de gran tamaño.

³⁷ <https://developer.android.com/studio/index.html#downloads>

Caché: es un componente hardware o software que almacena datos para que futuras peticiones a dichos datos se puedan obtener más rápidamente. Los datos almacenados en una caché suelen ser el resultado de una consulta previa.

CLI (Command Line Interface): el interfaz de la línea de comandos es un método que permite a los usuarios dar instrucciones a algún programa informático por medio de una línea de texto simple. Debe notarse que los conceptos de CLI, shell y emulador de terminal no son lo mismo, aunque suelen utilizarse como sinónimos.

Cloud: ver “nube”.

Cloud Computing: ver “computación en la nube”.

Cloud Server: servidor capaz de ofrecer sus servicios en la nube.

Clúster: grupo de máquinas que comparten algún hardware (por ejemplo un *array* de discos) y que se comportan como si fueran la misma máquina. De un clúster se puede esperar alto rendimiento, alta disponibilidad, balanceo de carga y escalabilidad.

Computación en la nube: paradigma que permite ofrecer servicios de computación a través de Internet.

Coste y beneficio: representa la relación entre los beneficios económicos del sistema y los costes necesarios para su desarrollo e implantación.

CPD (Centro de Proceso de Datos): sala que aglutina los grandes sistemas informáticos y equipos de telecomunicaciones de una gran empresa, y que suelen estar protegidos por altas medidas de seguridad.

Cuadrante Mágico de Gartner: su objetivo es proporcionar un análisis cualitativo de un mercado en lo que respecta a su dirección, madurez y participantes (ver Fig. 147Fig. 147 Descripción Cuadrante Mágico de Gartner). En el eje X de dicho cuadrante, Gartner define el elemento *“completeness of vision”* (Integridad de visión) y viene a representar el conocimiento de los proveedores sobre cómo se puede aprovechar el momento actual del mercado para generar valor tanto para sus clientes como para ellos mismos. El eje Y trata sobre *“ability to execute”* (Capacidad de ejecutar) y trata de medir el producto en sí (sus funcionalidades, calidad...), las formas de licenciamiento y el costo del producto además de lo rápido que puede responder el proveedor a un cambio en las tendencias del mercado, desde el punto de vista de actualizar sus productos y ofrecer las nuevas funcionalidades que demanden sus clientes. Los dos ejes dividen el cuadrante en cuatro sectores: *leaders*, *visionaries*, *challengers* y *niche players*. Ser líder significa haber puntuado alto en los dos ejes de medida [48].

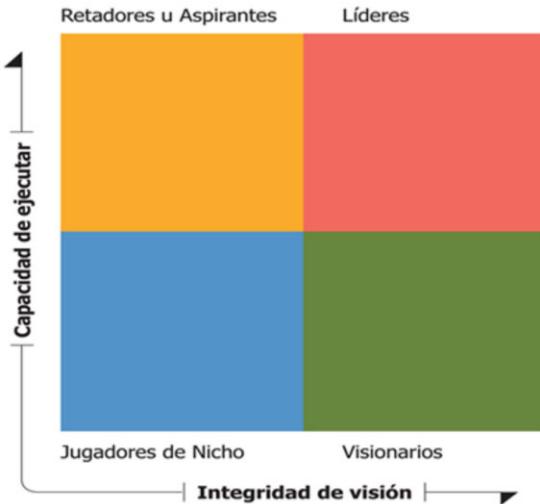


Fig. 147 Descripción Cuadrante Mágico de Gartner [48]

Datacenter: ver "CPD".

Disponibilidad: define la proporción de tiempo que el sistema es funcional y está trabajando. Se ve afectada por errores del sistema, problemas de infraestructura, ataques malignos y cargas del sistema. Habitualmente se mide como un porcentaje del tiempo de uso. Las aplicaciones cloud generalmente proporcionan a sus usuarios un SLA para que diseñen e implementen las aplicaciones en una forma que maximice la disponibilidad.

Durabilidad: ver "persistencia".

DVCS (Distributed Version Control System): sistema de control de versiones que permite a muchos desarrolladores software trabajar en un proyecto de forma local sin compartir red. Uno de los más populares es Git.

Escalabilidad: es la capacidad de un sistema para manejar incrementos de carga sin impactar en el rendimiento aumentando los recursos disponibles si es necesario. Las aplicaciones cloud suelen tener cargas variables y picos en actividad. Predecirlos, especialmente en un escenario de tenencia múltiple, es casi imposible. En su lugar, las aplicaciones deben ser capaces de escalar tanto para añadir recursos como para decrementar recursos en función de los picos de demanda. La escalabilidad no tiene solo que ver con instancias de computación sino con otros elementos tales como almacenamiento de datos, etc.

Espacio de implementación: también conocido como *staging slot* o ranura de ensayo. Habilita un mecanismo de paso a producción con menos riesgos y más rápido ya que la aplicación se despliega primero a este espacio y una vez que ha arrancado y lo ha hecho correctamente, simplemente se intercambia su espacio con el espacio de producción.

ETSIISI (Escuela Técnica Superior de Sistemas Informáticos): es un centro de la Universidad Politécnica de Madrid que imparte los estudios necesarios para obtención de los títulos de Grado en Ingeniería de Software y en Ingeniería de Computadores. La Escuela Universitaria de Informática, junto con la Escuela Universitaria de Ingeniería Técnica de Telecomunicación, la Escuela Universitaria de Topografía y el Centro Superior de Diseño de Moda de Madrid, integran el complejo universitario del Campus Sur, situado en el km 7 de la Carretera de Valencia.⁴

Firmar APK: Android requiere que todas las aplicaciones estén firmadas digitalmente con un certificado antes de que puedan ser instaladas. Android utiliza este certificado para identificar al autor de una app. El certificado no tiene que estar firmado por una autoridad certificadora. Las aplicaciones app frecuentemente utilizan certificados auto firmados, en cuyo caso, el desarrollador de la aplicación guarda la clave privada del certificado.

Framework: se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. Los objetivos principales que persigue un *framework* son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Frontend: ver "backend".

Gartner: es una empresa de consultoría dedicada de manera exclusiva a investigar la industria de las TI, analizar las tendencias del mercado y elaborar el ranking de soluciones tecnológicas para facilitar la selección de soluciones y productos, basados en una metodología de trabajo propia y un equipo de trabajo con una vasta experiencia y distribuido en todo el planeta.

Git: sistema de control de versiones de distribuido (DVCS)

Gradle: es una herramienta para automatizar la construcción de proyectos en cualquier lenguaje, en lo que se refiere a las tareas de compilación, testing, empaquetado y el despliegue de los mismos.

Grid Computing: forma de computación distribuida, a través de la cual una super computadora virtual compuesta de un grupo de computadoras que se encuentran conectadas a la red libremente, trabajan en conjunto para realizar tareas muy complejas.

Gson: es una biblioteca de código abierto para el lenguaje de programación Java que permite la serialización y deserialización entre objetos Java y su representación en notación JSON.

GUI (Graphical User Interface): es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información

y acciones disponibles en la interfaz. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador. Habitualmente las acciones se realizan mediante manipulación directa, para facilitar la interacción del usuario con la computadora. Surge como evolución de las interfaces de línea de comandos que se usaban para operar los primeros sistemas operativos y es pieza fundamental en un entorno gráfico. Como ejemplos de interfaz gráfica de usuario, cabe citar los entornos de escritorio Windows, el X-Window de GNU/Linux o el de Mac OS X, Aqua.

Historia de Usuario (HU): es una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos (acompañadas de las discusiones con los usuarios y las pruebas de validación). Cada historia de usuario debe ser limitada, ésta debería poderse escribir sobre una nota adhesiva pequeña. Deberían ser escritas por los clientes. Las historias de usuario son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes.

Host (anfitrión): término usado en informática para referirse a las computadoras conectadas a una red, que proveen y utilizan servicios de ella. Los usuarios deben utilizar anfitriones para tener acceso a la red.

HTTP (Hypertext Transfer Protocol): protocolo de marcado de dato usado para estructurar y dar formato al contenido de las páginas *Web*.

HTTPS (Hypertext Transfer Protocol Secure): cuando por debajo del HTTP se coloca una capa de seguridad SSL/TLS, la cual permite la encriptación de las comunicaciones.

Interoperabilidad: es la capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información que se ha intercambiado.

IT (Information Technology): ver "TI".

JAR (Java ARchive): es un tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java. Los archivos JAR están comprimidos con el formato ZIP y cambiada su extensión a .jar.

JAX-RS (Java API for RESTful web Services): es una especificación y API del lenguaje de programación Java que proporciona soporte en la creación de servicios *Web* de acuerdo con el estilo arquitectónico REST. JAX-RS usa anotaciones, introducidas en Java SE 5, para simplificar el desarrollo y despliegue de los clientes y puntos finales de los servicios *Web*. A partir de la versión 1.1 en adelante, JAX-RS es una parte oficial de Java EE 6. Una característica notable de ser parte oficial de Java EE es que no se requiere configuración para comenzar a

usar JAX-RS. Para los entornos que no son Java EE 6 se requiere una entrada en el descriptor de despliegue web.xml.

Jersey: es un framework de código abierto para el desarrollo de servicios *Web RESTful* en Java que implementa la referencia JAX-RS.

JSON (*JavaScript Object Notation*): es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente. Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON. En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval(), lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador *Web*.

LAN (*Local Area Network*): red que interconecta ordenadores en un área limitada, como por ejemplo un hogar u oficina.

Log: se trata de un registro o fichero que guarda todas las acciones o eventos que se van presentando durante la ejecución de un programa informático.

Mainframe (Computadora central): es una computadora grande, potente y costosa, usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos; por ejemplo, para el procesamiento de transacciones bancarias.

Marshalling: ver "serialización".

Máquina virtual: implementación software de una máquina física donde se puede instalar un sistema operativo igual o diferente del que tiene el equipo físico.

Maven: es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant, pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias con otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Mercado objetivo: representa la existencia de un mercado objetivo para el que refleja la necesidad de un producto o servicio.

Modelo de Proceso Software: existen varios modelos de desarrollo de software o metodologías tales como el modelo en cascada, V, incremental, ágil, iterativo y espiral.

Modelo Incremental: el modelo incremental es una aproximación intuitiva al modelo en cascada. Múltiples ciclos de desarrollo se suceden, convirtiendo el ciclo de vida en un ciclo multi-cascada. Los ciclos se dividen en iteraciones más pequeñas y más fácilmente gestionadas. Las iteraciones pasan a través de las etapas de requisitos, diseño, implementación y pruebas. Durante la primera iteración, se genera una versión operativa del software.

Modelo ágil: es un modelo incremental donde el software se desarrolla en ciclos rápidos e incrementales. El desarrollo es resultado de pequeñas versiones incrementales que se basan en la funcionalidad construida previamente y, es probado cuidadosamente para asegurar la calidad del software. Este modelo es el preferido para aplicaciones en las que el tiempo es crítico.

Multitenancy: ver “tenencia múltiple”.

Multitenencia: ver “tenencia múltiple”.

NIST (National Institute of Standards and Technology): es una agencia de la Administración de Tecnología del Departamento de Comercio de los Estados Unidos. La misión de este instituto es promover la innovación y la competencia industrial en Estados Unidos mediante avances en metrología, normas y tecnología de forma que mejoren la estabilidad económica y la calidad de vida.

Nube: metáfora para Internet, por ser así representada en los diagramas de red.

On-premises software: software que está instalado y se ejecuta en los ordenadores del edificio de la persona u organización que lo utiliza en vez de en un servidor remoto o en la nube.

Off-premises software: también llamado SaaS o *Cloud Computing*.

Paradigma: conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento.

Partition tolerance: ver “tolerancia a particiones”.

Persistencia: también conocido como durabilidad. Se refiere a la característica acerca del estado que sobrevive el proceso que lo crea. En la práctica se consigue almacenando dicho estado como datos en un almacenamiento de datos en un ordenador.

POJO (Plain Old Java Object): es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

Pooling (fondo): es una técnica de gestión de recursos que consiste en la agrupación de recursos de manera que se posibilite una utilización eficiente de los mismos.

PowerShell: es una plataforma de automatización y lenguaje de script para Windows Server y Windows que permite simplificar la gestión de los sistemas. A diferencia de otros intérpretes de comandos, PowerShell emplea el poder del framework .NET.

Product backlog: comprende una lista ordenada de requisitos que es mantenida para un producto por un equipo ágil. Consta de las características principales de la aplicación, errores, requisitos no funcionales, etc., es decir, cualquier cosa que se deba hacer para entregar un producto viable. El dueño del producto ordena el *product backlog* en base al riesgo, valor de negocio, dependencias y fechas de entrega.

Ranura de ensayo: ver “espacio de implementación”.

Redundancia: se dice de la cualidad que tiene un mismo dato cuando es almacenado en distintos lugares o cuando dos o más equipos hardware están repetidos y desempeñan la misma función.

Rendimiento: es una indicación del grado de reacción de un sistema a ejecutar cualquier acción dentro de un intervalo de tiempo.

Repositorio: un repositorio, depósito o archivo es un sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

Resistencia: es la habilidad de un sistema para manejar y recuperarse de los errores elegantemente. La naturaleza de hospedaje en la nube, donde las aplicaciones son mutitenedor, utilizan servicios compartidos de la plataforma, compiten por recursos y ancho de banda, se comunican en Internet y se ejecutan en *hardware* genérico significa que existe una alta probabilidad que sucedan errores tanto permanentes como transitorios. Detectar fallos y recuperarse rápida y eficientemente es necesario para conseguir que la aplicación sea resistente.

REST (REpresentational State Transfer): estilo arquitectónico que se aplica al diseño de APIs en los servicios *Web* modernos. Una *Web API* que se ajusta al estilo arquitectónico REST es una REST API. En este caso, el servicio *Web* se denomina RESTful.

Seguridad: es la capacidad del sistema de prevenir acciones no permitidas ya sean maliciosas o accidentales, y prevenir la divulgación o pérdida de información. Las aplicaciones cloud tienen especial riesgo ya que están expuestas a Internet, fuera de los límites de confianza una red local, frecuentemente están abiertas al público y pueden servir a usuarios de poca confianza. Las aplicaciones se deben diseñar y desplegar de una forma que se las proteja de ataques maliciosos, se restrinja el acceso solo a usuarios autorizados y se protejan los datos sensibles.

Serialización: la serialización (*marshalling*) consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria)

con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML o JSON, entre otros. La serie de bytes o el formato pueden ser usados para crear un nuevo objeto que es idéntico en todo al original, incluido su estado interno (por tanto, el nuevo objeto es un clon del original). La serialización es un mecanismo ampliamente usado para transportar objetos a través de una red, para hacer persistente un objeto en un archivo o base de datos, o para distribuir objetos idénticos a varias aplicaciones o localizaciones.

Servicio Web: son sistemas software diseñados para soportar una interacción interoperable máquina a máquina sobre una red. Los servicios *Web* suelen ser APIs *Web* que pueden ser accedidas dentro de una red (principalmente Internet) y son ejecutados en el sistema que los aloja.

Shell: el *shell* o intérprete de órdenes o intérprete de comandos es el programa informático que provee una interfaz de usuario para acceder a los servicios del sistema operativo. Dependiendo del tipo de interfaz que empleen, los shells pueden ser: de líneas texto (CLI, Command-Line Interface, interfaz de línea de comandos), gráficos (GUI, Graphical User Interface, interfaz gráfica de usuario), de lenguaje natural (NUI, Natural User Interface, interfaz natural de usuario).

SLA (Service Level Agreement): un acuerdo de nivel de servicio es un contrato escrito entre un proveedor de servicio y su cliente con objeto de fijar el nivel acordado para la calidad de dicho servicio. El SLA es una herramienta que ayuda a ambas partes a llegar a un consenso en términos del nivel de calidad del servicio, en aspectos tales como tiempo de respuesta, disponibilidad horaria, documentación disponible, personal asignado al servicio, etc.

SSL/TLS (Secure Socket Layer/Transport Layer Security): protocolos criptográficos de seguridad que permiten cifrar mensajes.

Staging slot: ver “espacio de implementación”.

Startup: son empresas que tratan de montar un nuevo negocio y que generalmente están apoyadas en la tecnología. Generalmente son empresas asociadas a la innovación, al desarrollo de tecnologías, al diseño *Web* o desarrollo *Web*; son empresas de capital riesgo.

Tenencia múltiple: también conocido como multitenencia o multipropiedad. Se trata de un principio de arquitectura de software en la cual una sola instancia de la aplicación se ejecuta en el servidor, pero sirviendo a múltiples clientes o tenedores. Este modelo es diferente de las arquitecturas con múltiples instancias en donde cada cliente o tenedor tiene su propia instancia de la aplicación. Con una arquitectura de tenencia múltiple, la aplicación puede particionar virtualmente sus datos y su configuración para que cada cliente tenga una instancia virtual adaptada a sus requerimientos. Algunos expertos consideran la tenencia múltiple como un factor decisivo del paradigma de computación en la nube ya que permite compartir los recursos y los costos al momento de la ejecución de la aplicación. Por otra parte, la implementación de

este tipo de arquitecturas es más compleja, ya que hay muchas cosas a tener en cuenta. Por ejemplo, la seguridad de los datos es esencial con el fin de que los usuarios de una instancia no puedan acceder a los datos de otra instancia.

Teorema CAP: también llamado Teorema de Brewer, enuncia que es imposible para un sistema de cómputo distribuido garantizar simultáneamente la consistencia o *Consistency* (la garantía que todos los nodos vean la misma información al mismo tiempo), la disponibilidad o *Availability* (la garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente) y la tolerancia al particionado o *Partition Tolerance* (la garantía que el sistema sigue funcionando a pesar de que haya sido partido por fallo de red). Según el teorema, un sistema puede tener no más de dos de estas tres características simultáneamente.

TI (Tecnología de la Información): término usado para referirse a las tecnologías que ayudan a producir, modificar, almacenar y/o comunicar información como, por ejemplo, el hardware, el software, el escáner, la impresora, el modem y el cd rom.

TIC (Tecnología de la Información y de la Comunicación): término usado para referirse al conjunto de tecnologías o sistemas compuesto por la integración de las TI (Tecnologías de la Información) y las TC (Tecnologías de la Comunicación) como, por ejemplo, televisión, radio, cine, periódicos, la fibra óptica, comunicación vía satélite, móviles y redes de comunicación a través de la Internet.

Thumbnail: imagen en miniatura.

Transacciones ACID (*Atomicity, Consistency, Isolation, Durability*): este tipo de transacciones en base de datos (conjunto de órdenes que se ejecutan de forma atómica) aseguran que una transacción es completa y consistente. En un entorno on-premises es lo esperado, no así en un entorno en la nube.

Transacciones BASE (*Basically Available, Soft State, Eventually Consistent*): este tipo de transacciones tiene muy en cuenta que los recursos pueden fallar y solo puede garantizar que los datos al final serán consistentes aunque no sabemos cuándo exactamente. BASE es frecuentemente utilizado en entornos volátiles donde los nodos pueden fallar o los sistemas necesitan trabajar tanto si el usuario está conectado a una red o no. Las arquitecturas *cloud* son tolerantes a particiones y se basan en este tipo de transacciones.

Tolerancia a particiones en la nube: también conocido como *partition tolerance*. Significa que si una instancia de un recurso de computación no puede completar una tarea, se llama a otra instancia para finalizar el trabajo. Al final, las discrepancias desaparecerán y todo quedará consistente.

TTM (Time To Market): representa el tiempo que se necesita para desarrollar un producto y lanzarlo al mercado.

Ubicuidad: un servicio posee esta cualidad cuando es accesible desde cualquier lugar.

UI (*User Interface*): GUI es un subconjunto del interfaz de usuario (UI). UI puede incluir interfaces no gráficos como CLI que no son considerados GUI.

URI (*Uniform Resource Identifier*): el identificador de recursos uniforme es una cadena de caracteres utilizada para identificar un recurso. Esta identificación posibilita la interacción con representaciones del recurso en la red, generalmente www, utilizando protocolos específicos. La forma más común de URI es URL, generalmente referenciada como dirección *Web*.

URL (*Uniform Resource Locator*): el localizador de recursos uniforme es una cadena que identifica un recurso en Internet. Es la cadena que se escribe en los navegadores cuando se quiere navegar hacia un sitio *Web*.

Usabilidad: es la facilidad con la que un usuario puede aprender a operar, preparar datos de entrada, interpretar las salidas de un sistema o componente.

Virtualización: abstracción de los recursos de una computadora, donde se crea una capa entre el hardware de la máquina física (*host*) y el sistema operativo de la máquina virtual (*guest*) siendo un medio para crear una versión virtual de un dispositivo o recurso, como un servidor, un dispositivo de almacenamiento, una red, etc.

WAN (*Wide Area Network*): red que interconecta ordenadores en área bastante amplia, normalmente utilizada por empresas y administraciones públicas para crear una red con nodos situados en diferentes localizaciones distantes entre sí.

WAR (*Web application ARchive*): es un fichero JAR que se utiliza para distribuir una colección de JSPs, Servlets Java, Clases Java, ficheros XML, páginas estáticas HTML y otros recursos que junto constituyen una aplicación *Web*.