Migración de ambientes de recursos Azure correspondiente a las integraciones de la compañía J.S. Held

Arbey de Jesús Villegas Carvajal

Aspirante al título de Ingeniero de Sistemas otorgado por UdeA

Asesores

Sandra Patricia Zabala Orrego, Especialista Gerencia de Proyectos, UdeA

Jonny Carmona Rodriguez, Enterprise Data Architect, J.S. Held

Universidad de Antioquia

Facultad de Ingeniería

Ingeniería de Sistemas

Seccional Oriente

2023

Agradecimientos

Agradezco a la profesora Sandra Zabala por todo el apoyo brindado durante la práctica empresarial, con su guía fue posible llevar a efecto las actividades efectivamente. Un agradecimiento especial a Jonny Carmona, mi asesor externo, que desde los inicios fue un soporte incondicional, un ejemplo de persona y profesional, y también un gran compañero del cual aprendo constantemente cuando tengo la bondad de interactuar con él.

TABLA DE CONTENIDO

RESUMEN 9
ABSTRACT 10
I. INTRODUCCIÓN 11
II. PLANTEAMIENTO DEL PROBLEMA 12
III. JUSTIFICACIÓN 13
IV. OBJETIVOS 14
V. MARCO TEÓRICO 16
Azure 16
Azure DevOps 19
Infraestructura como código 19
VI. METODOLOGÍA 21
Scrum 21
Código fuente 22
Convención de nombramiento para los recursos de Azure: 22
Tecnologías de desarrollo y uso 23
Infraestructura como Código (IaC): 24
Arquitectura base 27
Documentación 30
VII RESULTADOS 31
Migración de la infraestructura base 31
Migración de la integración D365 Worker 32
Integración con Box 34
Integración con HRSoft 36
VIII. CONCLUSIONES 38

IX. RECOMENDACIONES 39

REFERENCIAS 40

LISTA DE TABLAS

Tabla 1 Listado de subscripciones Azure 14
Tabla 2 Convención de nombramiento de recursos 23
Tabla 3 Componentes de la infrastructura base 31
Tabla 4 Descripción de los componentes de la integración D365 Worker 32
Tabla 5 Descripción de los componentes de la integración con Box 34
Tabla 6 Descripción de los componentes de la integración con HRSoft 35

LISTA DE FIGURAS

Fig. 1 Estrategia git feature branch 22
Fig. 2 Storage account ARM template con código Bicep 25
Fig. 3 Código de pipeline ADO para crear un servicio Key vault 26
Fig. 4 Patrón publisher/subscriber 27
Fig. 5 Aplicaciones contextualizadas 28
Fig. 6 Estructura de carpeta en el repositorio para las aplicaciones contextualizadas 29
Fig. 7 Infraestructura base 31
Fig. 8 Componentes de la integración D365 Worker 32
Fig. 9 Componentes de la integración con Box 33
Fig. 10 Diagrama de componentes de la integración con HRSoft 35

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

IEEE	Institute of Electrical and Electronics Engineers
ADO	Azure DevOps
ARM	Azure Resource Manager
CI/CD	Integración continua/despliegue continuo
UdeA	Universidad de Antioquia
API	Interfaz de programación de aplicación
SMB	Sever Message Block
IDE	Integrated Development Environment
DSL	Domain-specific language
CLI	Command-line interface

RESUMEN

Este informe final de la práctica del pregrado de Ingeniería de Sistemas pretende plasmar la implementación de esta en detalle, su metodología y los resultados obtenidos. Se describe conceptos de Azure que permiten comprender la esencia de esta plataforma de servicios en la nube. Dentro de los conceptos que se describen, se hace énfasis en los tipos de recursos utilizados en las integraciones. Muy importante resaltar que también se hace uso de prácticas de DevOps para el despliegue de las integraciones a producción. Luego se describe el estado actual de las integraciones y la necesidad de migrar de una subscripción a otra los recursos que hacen partes de estas. Además, se explica los estándares y prácticas que se usaron para la implementación de las integraciones y su migración. Por último, se detallará cada una de las integraciones migradas y los aspectos principales de estas como su diseño a gran nivel, modelo de datos, sistemas involucrados, flujo de datos y documentación.

ABSTRACT

This final report of the undergraduate practice in Systems Engineering aims to outline the detailed implementation of it, its methodology, and the results obtained. It describes Azure concepts that allow understanding the essence of this cloud service platform. Among the concepts described, there is an emphasis on the types of resources used in the integrations. It is essential to highlight that DevOps practices are also used for deploying integrations to production. Next, it describes the current state of the integrations and the need to migrate the resources that are part of them from one subscription to another. Additionally, it explains the standards and practices used for the implementation of the integrations and their migration. Finally, each of the migrated integrations and their main aspects, such as high-level design, data model, involved systems, data flow, and documentation, will be detailed.

I. INTRODUCCIÓN

J.S. Held es una compañía multinacional que ofrece servicios técnicos, legales, científicos y financieros orientados a recursos y valores de compañías en riesgo. La empresa lleva 49 años desde su fundación por Jeromy S. Held. La organización ha venido creciendo desde entonces fusionándose con otras compañías y haciendo presencia en los 5 continentes para ofrecer sus servicios especializados. Hoy hay más de 2.500 empleados trabajando en la compañía.

El equipo de servicios tecnológicos ha venido evolucionando en los últimos 4 años, donde se viene consolidando los equipos de mesa de ayuda, seguridad, aplicaciones, proyectos de tecnología, arquitectura empresarial, integraciones y desarrollo.

Dado que el equipo de IT y por lo tanto el de Integraciones & Desarrollo son relativamente nuevos, los estándares de desarrollo y buenas prácticas no fueron definidos desde el principio. Esto también tiene que ver con el hecho de que el objetivo de la compañía no es el desarrollo de software o servicios meramente tecnológicos, como se describió anteriormente. Por lo tanto, los desarrollos existentes no cumplen con los lineamientos de la industria de la ingeniería de software y el desarrollo para la nube Azure. En aras de mejorar, la compañía decidió configurar nuevas subscripciones de Azure (en este documento también se llaman ambientes) que dentro de su configuración tienen políticas de seguridad, y buenas prácticas que exigen que los recursos desplegados como parte de los desarrollos y la infraestructura sigan los estándares definidos.

Es este documento se describe de forma sucinta de las características y retos que se tuvieron con las integraciones anteriores que motivaron a la mejora en el proceso de desarrollo para Azure. Además, se plasma detalladamente la metodología seguida para realizar las migraciones y que son los estándares también definidos para nuevos desarrollos e integraciones. Adicionalmente, se hace énfasis en nuevas prácticas introducidas para el despliegue de los desarrollos a diferentes ambientes por medio de prácticas de DevOps. De igual manera, se describe las actividades realizadas como parte de la práctica para migrar las integraciones existentes y desarrollar nuevas con los nuevos estándares definidos.

II. PLANTEAMIENTO DEL PROBLEMA

Como parte del trabajo del equipo de integraciones y desarrollo, es necesario migrar y modernizar algunos sistemas de integración de unas subscripciones antiguas de Azure a otras, ya que los nuevos ambientes (subscripciones) tienen la configuración necesaria que cumple con los parámetros de seguridad y buenas prácticas requeridos por la compañía.

Las integraciones ya existentes fueron construidas directamente desde el portal de Azure, estableciendo el mismo lugar como ambiente de desarrollo y producto final de producción. Esto trae diferentes dificultades:

1. En la práctica no existe código fuente porque los recursos desplegados directamente desde el portal de Azure son manejados por la plataforma sin necesidad de código.
2. Cuando un desarrollador tiene que hacer un cambio a los recursos de Azure que pertenecen a una integración, tienen que ir directamente a afectar el recurso existente en el portal, que es el artefacto que funciona como componente activo del ambiente de producción.
3. Si por algún motivo (que no es tan probable, pero puede suceder) hay indisponibilidad de los recursos, o los recursos desaparecen de la subscripción de Azure por motivos humanos o de fallas en la nube, no habría una forma de recuperar los artefactos que representan el ambiente de producción. La única forma sería confiar en el conocimiento, memoria e intervención humana.

Simultáneamente, la documentación de los desarrollos existentes era nula, lo existente y desplegado en Azure era la “documentación”, lo que dificultaba tener un punto de referencia cuando se necesitaba consultar algún aspecto general de las integraciones.

Además, no existía metodología de desarrollo clara para que los desarrolladores tuvieran un marco de referencia para su día a día. No existía una arquitectura base ni estándares para construir las integraciones lo que se reflejaba en integraciones construidas a criterio de los desarrolladores.

III. JUSTIFICACIÓN

El desarrollo de software y todo lo que involucra su ciclo de vida es parte integral de la Ingeniería de Sistemas. En esta práctica se usa metodologías ágiles para el desarrollo de aplicaciones e integraciones, se tiene un marco de trabajo y arquitectura base para los desarrollos, también se definieron estándares de desarrollo lo cual permite que las integraciones sean consistentes, además se realizan despliegues haciendo uso de pipelines automatizados. Por lo anterior, este trabajo es relevante para la Ingeniería de Sistemas ya que se requiere adquirir y poner en práctica los conocimientos y la experiencia para llevar a efecto la implementación adecuada de sistemas de integración dentro de la compañía.

IV. OBJETIVOS

A. Objetivo general

Migrar los recursos de Azure Cloud del ambiente antiguo y no manejado hacia otro ambiente/subscripción Azure (ver la Tabla 1 para conocer el listado de subscripciones origen y destino de la migración), con el propósito de alojar los sistemas de integración en la región apropiada y con los estándares establecidos por la compañía.

Detalle	**Suscripción Azure**	**Descripción**
Antiguas suscripciones Azure	Pay-as-you-go	La suscripción más antigua de la compañía. Se tienen recursos mezclados de desarrollo y producción. La mayoría de los recursos se eliminarán.
	JSH - Azure DevTest subscription	Suscripción antigua de ambiente desarrollo y test.
	JSH - Azure subscription	Suscripción antigua de Producción.
Suscripciones Azure de destino de la migración	JSH- DevTest	Suscripción destino donde el equipo scrum publica y prueba las integraciones.
	JSH - ProdDMZ	Suscripción de producción donde se publican integraciones que se comunican con por lo menos con un sistema externo a la compañía.
	JSH - Production	Suscripción de producción donde se publican las integraciones que se comunican solo con sistemas internos de la compañía.

Tabla 1 Listado de subscripciones Azure

B. Objetivos específicos

- Utilizar los estándares definidos para el desarrollo de integraciones para la nube Azure.
- Crear código como infraestructura (IaC) para desplegar los recursos Azure a la nueva subscripción/ambiente de desarrollo y posteriormente producción.
- Crear pipelines para despliegue automático de los recursos a través de Azure DevOps.
- Documentar las integraciones a nivel técnico.

V. MARCO TEÓRICO

Azure

Azure es una plataforma de computación en la nube perteneciente a Microsoft. Azure posee una vasta oferta de servicios incluyendo almacenamiento, bases de datos, análisis, redes, inteligencia artificial, internet de las cosas (IoT) entre los más importantes. También, se puede clasificar los servicios de Azure como software como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS); que es como se clasifican los tipos de servicios en la computación en la nube en general.

Para el desarrollo de las integraciones se utilizan varios de los servicios Azure que facilitan el desarrollo de las aplicaciones. A continuación, se hace referencia a los más relevantes que son utilizados dentro del desarrollo de esta práctica y dentro de la compañía para el desarrollo de integraciones, y se explican con cierto nivel de detalle más adelante: Logic Apps, Functions apps, Key vault, Storage account, Service Bus, Application Insights, Monitor Log, API Management, y Data factory.

Logic Apps es una plataforma de integración y desarrollo basado en flujos de trabajo creados gráficamente (aunque por debajo, tienen una representación en código descriptivo). Son parte del conjunto de herramientas low-code ofrecido por Microsoft para la creación de aplicaciones, en especial integraciones. Los flujos de trabajo de Logic Apps se basan en disparadores y acciones. Los disparadores son eventos que inician la ejecución de los Logic Apps. Dentro de los diferentes disparadores que se tiene para iniciar la ejecución de un Logic App se encuentran: recurrente (regularmente cada cierta cantidad de tiempo), por un llamado HTTP, por una inserción o actualización de un registro en Azure SQL y muchos eventos disparadores más. Las acciones son operaciones dentro del flujo de trabajo las cuales ejecutan una tarea específica. Algunas acciones se conectan a sistemas o recursos Azure fuera del Logic App. Para que un Logic App se pueda conectar a otro sistema o recurso Azure, este hace uso de los conectores. Los conectores son elementos que permiten establecer una conexión entre el Logic App y otros sistemas. Supongamos que el Logic App necesita conectarse con Azure SQL para realizar una consulta a una tabla, para esto se requiere de un conector que se configura con autenticación y

ubicación del recurso, básicamente. Lo anterior es solo un ejemplo de conectores de los cuales existen muchos disponibles como para conectar con: SharePoint, Teams, Active Directory, HTTP, entre otros. Todas las acciones tienen una entrada (parámetros) que pueden recibir de la acción anterior o anteriores y una salida (output) que pueden entregar y tener disponible para las acciones siguientes.

Los Function app permiten crear y ejecutar funciones en la nube. Los Function app se pueden desarrollar en varios lenguajes de programación como C#, Java, JavaScript, TypeScript, PowerShell y Python. Los Function app se pueden desencadenar mediante diferentes tipos de eventos, conocidos como triggers o disparadores, así como los Logic Apps. Los triggers son los mecanismos que desencadenan la ejecución de una función. Algunos de los disparadores más comunes son: HTTP, Timer o recurrente, Blob Storage (se agrega o elimina un archivo), Queue Storage (se agregar un mensaje a la cola). Como parte de la estructura escencial de los Function app se tiene los bindings. Los bindings son una forma declarativa de conectar recursos externos a un Function app. Los bindings pueden ser conectados como entradas, salidas o ambos. Por ejemplo, se puede declarar un binding para que el Function app se conecte con un Storage Table (para de los servicios disponibles en un Storage Account). Si la conexión al Storage Table es configurada como de entrada, se pueden obtener datos de este. Si la conexión es configurada como de salida, se puede almacenar datos o realizar actualizaciones de los registros existentes.

Key Vault es un servicio para el almacenamiento de secretos y el acceso a estos de forma segura. Un secreto es todo aquello cuyo acceso se desea controlar de manera estricta y segura, como los tokens de API, las contraseñas, los certificados SSL o las claves criptográficas.

Storage account es uno de los servicios más utilizados cuando se crean aplicaciones en Azure por tener múltiples usos. Este servicio provee a los usuarios con cuatro componentes principales de almacenamiento: blobs, compartimiento de archivos, colas, y tablas. El componente de Blobs permite guardar una grande cantidad de datos no estructurados como archivos de texto e imágenes. Mientras que el componente de compartimiento de archivos es un servicio en la nube totalmente manejado que le permite a los usuarios almacenar y compartir archivos a través de SMB. Ahora bien, las colas (queues) facilitan el almacenamiento de mensajes que son producidos por

una aplicación y consumidos por otra de manera asíncrona. Por otro lado, las tablas son objetos que permiten almacenar datos no estructurados en un esquema clave valor.

El Service bus es otro servicio que actúa como intermediario de mensajes empresariales totalmente administrado con colas de mensajes, temas de publicación y subscripciones. Este es utilizado para desacoplar aplicaciones y servicios, los cuales publican y generan mensajes que asíncronamente pueden ser consumidos por otros servicios. Dentro del Service bus, las colas son utilizadas cuando se quiere que un mensaje sea consumido solo por una aplicación destino. Por el contrario, los temas de publicación (topics) son pertinentes cuando un mensaje es producido por una aplicación, pero se permite que una o varias aplicaciones puedan consumir un mismo mensaje por medio de subscripciones.

Application insights es una extensión de Azure Monitor que proporciona características de monitoreo de rendimiento de las aplicaciones. Con Application insights se recopilan métricas y datos de telemetría los cuales describen la salud de las aplicaciones. Application insights es utilizado en servicios como las aplicaciones web, Azure functions, Logic apps standard, entre otros.

Un Log Analytics Workspace es una unidad de almacenamiento donde se recopilan y almacenan datos de registro. Se utiliza para recuperar datos de varias fuentes como máquinas virtuales, el mismo Application insights y cualquier servicio de Azure que pueda generar logs. Es un componente esencial de Azure Monitor.

API Management es una plataforma robusta para la gestión del ciclo de vida de las API. API Management consta de una puerta de enlace de API, un plano de administración y un portal para desarrolladores. Estos componentes están alojados en Azure y son totalmente administrados por defecto. Dentro de los desafíos que ayuda a resolver están: abstrae de la complejidad de la arquitectura del backend de los consumidores de los API, expone de forma segura los servicios alojados dentro y fuera de la red de Azure, y habilita el descubrimiento y consumo de la API por parte de usuario internos y externos.

Azure DevOps

Azure DevOps es una plataforma que facilita la colaboración entre desarrolladores, gerentes de proyecto y colaboradores para la ejecución de proyectos de software. Azure DevOps proporciona características integradas a las que puede acceder a través de tu explorador web o cliente IDE. Azure DevOps provee un listado de servicios que se acoplan al flujo de trabajo de cada equipo de trabajo. Dentro de los servicios provistos encontramos: Boards, Repos, Pipelines, Test Plans y Artifacts.

Los Boards ofrecen un conjunto de herramientas ágiles para el trabajo de planificación y seguimiento, defectos de código y problemas mediante los métodos Kanban y Scrum. Ahora bien, los Repos proporcionan repositorios de Git o Control de versiones de Team Foundation (TFVC) para el control de código fuente. Por otro lado, los Pipelines ofrecen servicios de compilación y versión para admitir la integración y entrega continuas de las aplicaciones. Adicionalmente, los Test Plans proporciona varias herramientas para probar las aplicaciones, incluidas las pruebas manuales o exploratorias y las pruebas continuas. Por último, Artifacts permite a los equipos compartir paquetes como Maven, npm, NuGet, etc. desde orígenes públicos y privados e integrar el uso compartido de paquetes en las canalizaciones. En esta práctica se hace uso extensivo de la funcionalidades disponibles en Boards, Repos y Pipelines.

Infraestructura como código

Algunas disciplinas han salido a la luz recientemente como integración continua, desarrollo orientado a las pruebas, automatización de despliegues y más. El propósito ha sido automatizar la mayoría de las partes involucradas en el ciclo de vida del desarrollo de productos de software. Ahora también se busca que la infraestructura en la cual el software está soportado se construya y administre automáticamente.

La infraestructura como código (IaC) es un enfoque hacia la automatización de la infraestructura basado en prácticas de desarrollo de software. Se enfatiza en rutinas consistentes y repetibles para el aprovisionamiento y modificaciones de sistemas y su configuración. Se realizan cambios al código y luego, se usa automatización para probar y aplicar esos cambios a los sistemas.

Para implementar Infraestructura como código en este proyecto se usan plantillas ARM y Biceps. Toda la gestión de la infraestructura de Azure esta hecha a través de Azure Resource Manager (ARM). ARM es un API que se puede utilizar para listar, crear, actualizar y borrar todos los recursos de una subscripción de Azure.

El API ARM puede ser utilizado para manejar la infraestructura de forma declarativa usando las plantillas ARM. Las plantillas ARM están escritas en formato JSON o BICEP. Normalmente, las plantillas ARM están escritas en formato JSON, pero una de las desventajas es que el contenido se puede volver extenso y con una estructura compleja. Para solucionar esto, Microsoft introdujo un leguaje específico del dominio (DSL) como alternativa para crear plantillas ARM: código Bicep.

Bicep es un lenguaje que usa sintaxis declarativa para implementar recursos de Azure. Bicep es una abstracción transparente del código JSON de plantillas de ARM, y no pierde ninguna de las funcionalidades de las plantillas JSON. Durante la implementación, la CLI de Bicep convierte un archivo de Bicep en un archivo JSON de plantilla de ARM.

VI. METODOLOGÍA

Scrum

Para la ejecución del proyecto se usa metodología Scrum con las siguientes características. Los sprints son de 15 días. Al principio del sprint se realiza la reunión de planeación. Al final se realiza la reunión de revisión y retrospectiva. Todos los días se hace reunión de sincronización (standup meeting).

La metodología Scrum es una forma de gestionar proyectos de software que se basa en la entrega de valor en ciclos cortos y adaptativos. En la compañía se sigue esta metodología con las siguientes características:

- Los sprints son de 15 días. Un sprint es un período de tiempo fijo en el que se desarrollan y entregan las funcionalidades acordadas.
- Al principio del sprint se realiza la reunión de planeación. En esta reunión se define el objetivo del sprint, se seleccionan las tareas del backlog (lista de requerimientos priorizados) y se asignan a los miembros del equipo.
- Al final del sprint se realiza la reunión de revisión y retrospectiva. En la revisión se muestra el resultado del sprint y se recibe su retroalimentación. En la retrospectiva se analiza el desempeño del equipo y se identifican las mejoras a implementar en el siguiente sprint.
- Todos los días se hace reunión de sincronización (standup meeting). Esta es una reunión breve y diaria en la que cada miembro del equipo informa lo que hizo el día anterior, lo que va a hacer hoy y los obstáculos que enfrenta.

Código fuente

El código fuente está alojado en Azure DevOps con el motor git. Existe una estrategia para las ramificaciones en el repositorio de código fuente, esta se llama git feature branch workflow como se puede ver en la Fig. 1. A continuación, se describe el proceso:

- Cuando hay una nueva funcionalidad o corrección de error, el desarrollador debe crear una rama feature o bug.
- El nombre de las ramas feature se estructura así: *feature/<número de ticket>*.
- El nombre de las ramas que corresponden a correcciones de errores se nombra de la siguiente manera: *bug/<número del ticket de error>*.

Cuando el desarrollo se ha terminado, se debe crear un Pull Request apuntando a la rama principal (main en el mayor de los casos). Por lo menos un code reviewer debe aprobar el pull request antes de que el código pueda ser integrado en la rama principal.

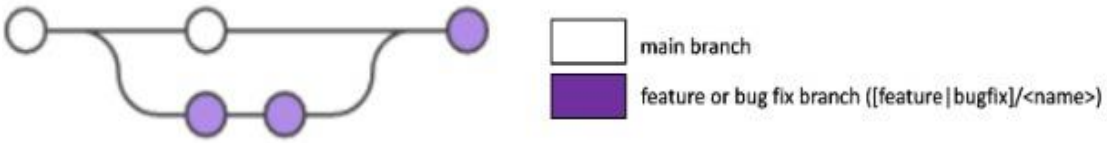

Fig. 1 Estrategia git feature branch

Convención de nombramiento para los recursos de Azure:

Dado que se requiere crear recursos Azure para soportar la implementación de las integraciones, se define una convención para el nombramiento de estos. Existe una diferencia en el nombramiento dependiendo del ambiente y el número de instancia. A continuación, se detalla en la Tabla 2:

Ambiente: Desarrollo (dev), Producción (prod), DMZ (dmz). (En el Objetivo se detalla los ambientes).

Recurso	Nombramiento	Ejemplo
Resource group	rg-<ambiente>-<integracion>-<##>	rg-prod-boxintegration-01
Key vault	kv-<ambiente>-<integracion>-<##>	kv-dev-d365worker-01
Storage account	st<ambiente><integracion><##>	stprodboxintegration01
Logic app	logic-<ambiente>-<integracion>-<##>	logic-dmz-bamboointegration-01
Function app	fun-<ambiente>-integración>-<##>	fun-dev-percipio-01
Application insights	appi--<ambiente>-integración>-<##>	appi-prod-d365worker-01
API Management	apim-<ambiente>-integración>-<##>	apim-dmz-integrationsbase-01
Service Bus	sb-<ambiente>-integración>-<##>	sb-dev-integrationsbase-01
Monitor Log	log-<ambiente>-integración>-<##>	log-prod-integrationsbase-01

Tabla 2 Convención de nombramiento de recursos

Tecnologías de desarrollo y uso

Visual Studio / VS code

- Crear logic apps por medio de ARM templates.
- Editar YAML files para los pipelines Azure DevOps.
- Desplegar plantillas ARM en el ambiente de desarrollo.

Azure DevOps

- Control de código fuente.
- Tableros para manejo de proyectos ágiles.
- Automatización de despliegue con pipelines.

Azure

- Alojamiento de servicios cloud.
- Resource groups.
- Logic apps.

- Service bus.
- Storage account.
- Key vault.
- API management.
- Function apps.

Infraestructura como Código (IaC):

El desarrollo de los servicios Azure se realiza con plantillas ARM (Azure Resource Manager). Esta tecnología permite describir la infraestructura de las integraciones de forma declarativa. Las plantillas luego pueden ser usadas con Azure DevOps pipelines para realizar los despliegues respectivos.

Para la creación de plantillas ARM se puede usar formato JSON o lenguaje Biceps. El lenguaje Biceps es el más utilizado en nuestros desarrollos y es el recomendado por Microsoft por encima de JSON ya que es menos verboso. Al final Bicep se convierte en JSON internamente al momento de ser usado. En la Fig. 2 se muestra un ejemplo de código Biceps para la creación de un Storage account.

```
@description('Storage account name.')
param storageAccountName string

@description('Asset related tag.')
param tags object

param location string = resourceGroup().location

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-09-01' = {
  name: storageAccountName
  location: location
  kind: 'StorageV2'
  tags: tags
  properties:{
    allowBlobPublicAccess: false
    minimumTlsVersion: 'TLS1_2'
  }
  sku: {
    name: 'Standard_LRS'
  }
}
```

Fig. 2 Storage account ARM template con código Bicep

Despliegues

Los despliegues se ejecutan por medio de Azure DevOps pipelines. Los pipelines son representados en archivos YAML que son parte del repositorio de código fuente. Los archivos YAML contienen las instrucciones para realizar los despliegues a los diferentes ambientes (subscripciones Azure). Básicamente, los pipelines llaman los archivos correspondientes a las plantillas ARM con sus respectivos parámetros por medio de Azure CLI. A continuación, en la Fig. 3 se muestra una porción de código con un ejemplo de código YAML para pipelines Azure DevOps en el cual se crea un servicio Key vault en el ambiente de desarrollo:

```yaml
stages:
- stage: dev
  jobs:
  - job: dev
    variables:
    - name: var-resource-group-name
      value: $(vgvar-rg-mds-dev)
    - name: var-keyvault-name
      value: $(vgvar-mds-keyvault-dev)
    - name: var-template-param-file
      value: '$(var-template-param-path)/arm-create-az-kv.parameters.dev.json'
    steps:
    - checkout: self
      clean: true
    - checkout: basetemplates
      clean: true
    - task: AzureCLI@2
      displayName: 'Create key vault for master data service integration'
      inputs:
        azureSubscription: 'sc-masterdataservice-dev'
        scriptType: ps
        scriptLocation: inlineScript
        inlineScript: |
          az --version
          az deployment group create --resource-group $(var-resource-group-name)
          --template-file $(var-template-file) --parameters $(var-template-param-file)
          --parameters keyVaultName=$(var-keyvault-name) adminSecurityGroupId=$
          (var-sec-az-integrations-admin-objectid)
```

Fig. 3 Código de pipeline ADO para crear un servicio Key vault

Arquitectura base

Para la construcción y migración de las integraciones se utiliza dos patrones arquitectónicos principales: publisher/subscriber y self-contained applications.

Publisher/subscriber

Habilita a las integraciones para propagar mensaje a múltiples consumidores de manera asíncrona, sin acoplar los componentes que envían los mensajes con los que los consumen. En la Fig. 4 se visualiza los componentes que hacen parte de este patrón.

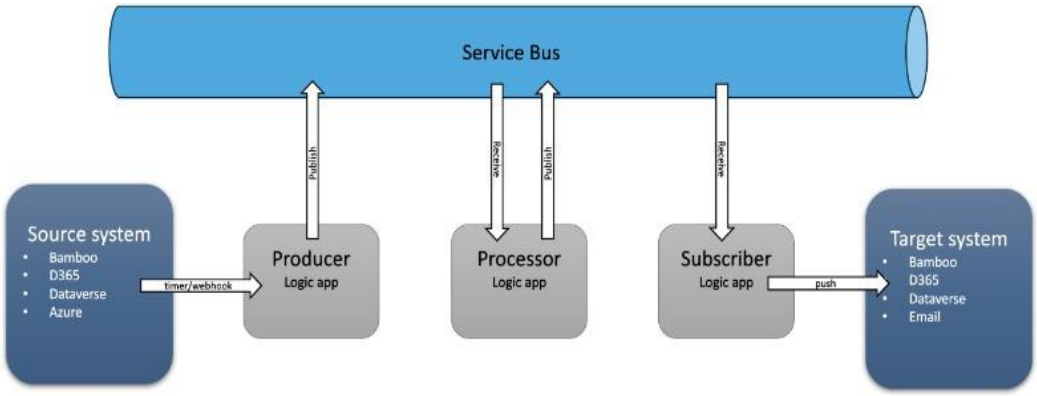

Fig. 4 Patrón publisher/subscriber

Producer/publisher

- Se comunica con el sistema origen de donde inicia la integración.
- Extrae los datos del sistema origen.
- Crea el mensaje inicial que representa los datos que se van a sincronizar al sistema destino.
- El mensaje debe contener la mayoría de los atributos de la entidad origen.
- Publica el mensaje a la cola correspondiente al sistema orquestador, en este caso Azure Service Bus.

Processor

- Escucha la cola de mensajes en donde el Producer/Publisher publica los mensajes.

- Transforma y filtra los mensajes provenientes de la cola del Service Bus.
- Obtiene datos de mapeo y completa el mensaje para complementarlo con los datos necesarios que el sistema destino requiere.
- Publica el mensaje transformado a otra cola del service bus.

Subscriber

- Escucha la cola de mensaje en donde el Processor publica los mensajes.
- Actualiza/inserta/elimina datos del sistema destino basado en el mensaje obtenido.

Aplicaciones contextualizadas

Unas de las dificultades de la organización de los recursos Azure en el pasado fue que se guardaban los recursos de una sola integración en múltiples lugares o grupo de recursos. Con el patrón de aplicaciones contextualizadas, todos los recursos pertenecientes a una integración son agrupados de forma inteligente. Todos los servicios y recursos necesarios para la implementación de la integración están ubicados en el mismo grupo de recursos. El mismo enfoque es aplicado en el repositorio de código fuente, documentación y despliegue como se puede ver en la Fig. 5.

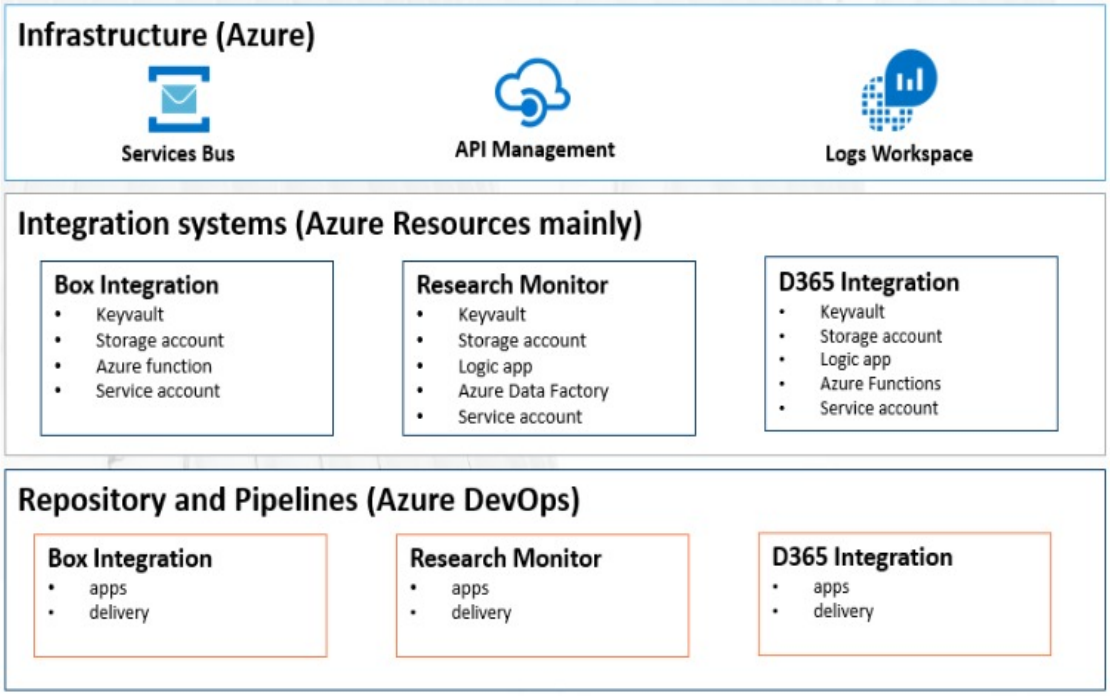

Fig. 5 Aplicaciones contextualizadas

A continuación, en la Fig.6 se describe la estructura de carpetas usada para los repositorios de código fuente para las aplicaciones contextualizadas:

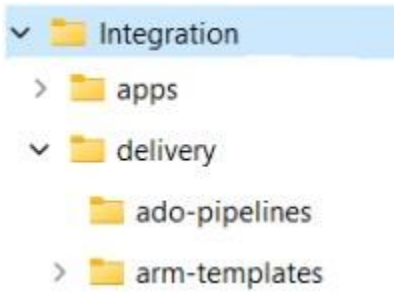

Fig. 6 Estructura de carpeta en el repositorio para las aplicaciones contextualizadas

- La carpeta **apps** contiene todo el código fuente para los Functions apps, Logic apps, Web apps, Data factory, y todo el código que representa las funcionalidades desarrolladas.
- La carpeta **delivery** contiene el Código como Infraestructura (IaC) para realizar escaneo de código estático, construirlo y desplegarlo como servicios Azure por medio de estrategias CI/CD (continuous integration/continuous delivery).
    - La carpeta **arm-templates** contiene los archivos y parámetros para crear los servicios Azure usando plantillas ARM.

- La carpeta **ado-pipelines** contiene los archivos YAML para crear los pipelines en Azure DevOps.

Documentación

Cada integración tiene su documentación creada en Word que es diligenciada desde el inicio del ciclo de vida y se debe actualizar conforme haya cambios durante este tiempo. La documentación es almacenada en SharePoint en la carpeta correspondiente a cada integración. Los diagramas de componentes y actividades de la integración son creados con Visio y almacenados en el mismo lugar.

Los apartes que generalmente contiene la documentación de cada integración son:

- Propósito
- Contactos (stakeholders, dueños del producto, desarolladores, arquitecto)
- Componentes que hacen parte de la integración.
- Proceso de sincronización.
- Validaciones.
- Entidades involucradas en cada sistema origen y destino.
- Flujo de datos y transformaciones.

VII RESULTADOS

A continuación, se describen las integraciones que se han implementado y migrado como parte de este proyecto.

Migración de la infraestructura base

La infraestructura base migrada contiene los servicios Azure que se usan transversalmente por múltiples integraciones. Como se ve en la Fig. 7, los servicios que hacen parte de la infraestructura base son: Service Bus, API Management y Logs Analytics Workspace.

Fig. 7 Infraestructura base

En la Tabla 3 se describe lo servicios, nombramiento y uso de los componentes de la infraestructura base.

Tabla 3 Componentes de la infrastructura base

Servicio	**Nombre**	**Uso**
Service Bus	sb-<dev|prod>-infrastructurebase-01	Orquestador de mensajería.
API Management	apim-<dev|prod>-infrastructurebase-01	Se publican los API a sistemas externos.
Logs Workspace	log-<dev|prod>-infrastructurebase-01	Almacenamiento administrado de logs provenientes de los servicios Azure pertenecientes a las integraciones.

Migración de la integración D365 Worker

El propósito de esta integración es sincronizar los datos de los empleados del sistema de recursos humanos (Bamboo HR) al ERP de la compañía (Dynamics 365 Finance & Operations). En el diagrama de la Fig. 8 se muestran los diferentes componentes que hacen parte de la integración.

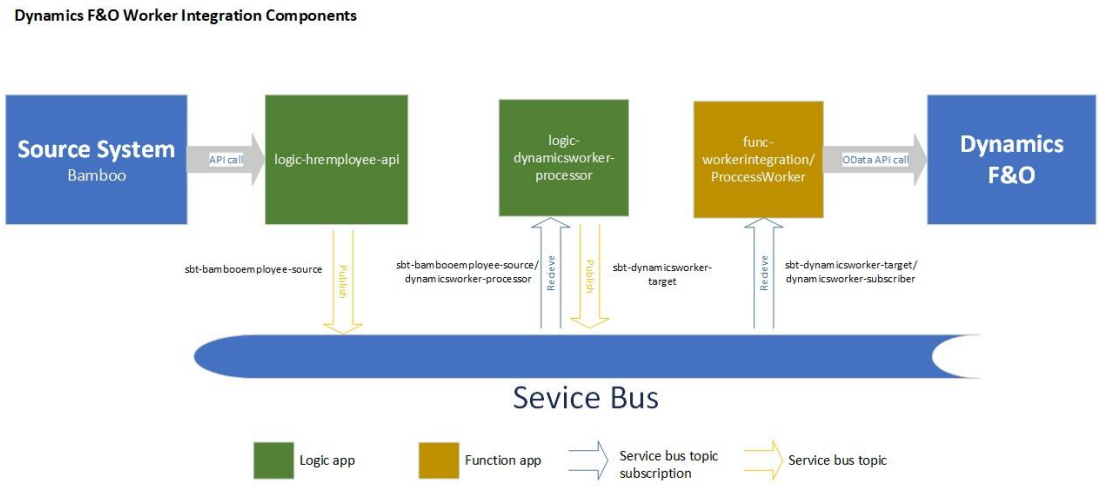

Fig. 8 Componentes de la integración D365 Worker

La Tabla 4 describe de forma sucinta los servicios y sistemas que hacen parte de la integración D365 Worker en donde se hace uso del patrón Publisher/subscriber para la implementación de la solución.

Tabla 4 Descripción de los componentes de la integración D365 Worker

Servicio	**Nombre**	**Descripción**
Sistema de recursos humanos	Bamboo HR	Sistema origen donde los datos de los empleados son administrados.
ERP	D365 F&O	Sistema destino donde se crean las entidades Workers, Employment, Position y Project Resource.

Logic app	logic-<dev|prod>-hremployee-01	Extrae los datos de empleados del sistema origen por medio de API cada 2 horas y publica los mensajes con datos de empleados en el Service Bus.
Logic app	logic-<dev|prod>-dynamicsworker-processor-01	Transforma el mensaje de empleados con datos adicionales para adaptarlo conforme el sistema destino lo necesita.
Function app	fun-<dev|prod>-workerintegration-01	Function app que se encarga de comunicarse con D365 por medio de un API OData para sincronizar los datos de empleados y afectar las entidades mencionadas en la fila ERP.

Integración con Box

Box es una plataforma para el almacenamiento e intercambio de archivos en la nube. La integración con Box permite crear automáticamente la carpeta personal de los nuevos empleados en la plataforma Box. Los diferentes componentes de la integración y su interacción son reflejados en la Fig. 9.

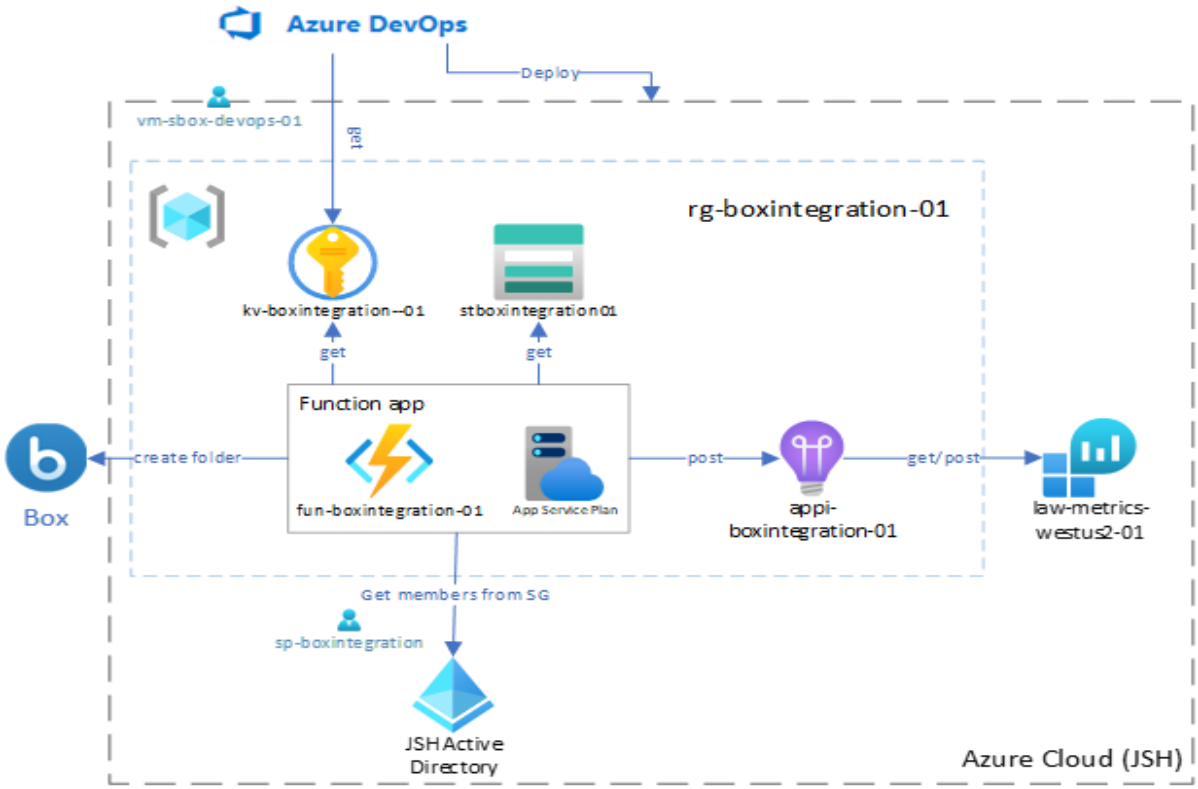

Fig. 9 Componentes de la integración con Box

Esta integración fue implementada siguiendo el patrón de aplicaciones contextualizadas y los componentes que hacen parte de ella son descritos en la Tabla 5.

Tabla 5 Descripción de los componentes de la integración con Box

Nombre del recurso	Tipo de recurso	Descripción
Active Directory	Fuente	Plataforma de gestión de identidades, recursos y aplicaciones de la organización. Contiene las cuentas de usuario a las cuales

		se les crea la carpeta personal en Box. Sistema origen.
Box	Destino	Plataforma en la nube para almacenar y compartir archivos. Sistema donde se crean las carpetas personales. Sistema destino.
appi-<env>-boxintegration-01	Application Insights	Consume los datos de la telemetría generados por el Function App.
fun-<env>-boxintegration-01	Function App	Contiene la lógica para crear la carpeta personal de los nuevos empleados en Box.
kv-<env>-boxintegration-01	Key vault	Servicio donde se almacena las credenciales del usuario de Sistema.
mi-<env>-boxintegration-01	Managed Identity	Cuenta de Sistema manejada para comunicar los servicios de Azure sin necesidad de contraseña.
st<env>boxintegration01	Storage Account	Es parte de la infraestructura necesario para que el Function app opera. Hay una tabla llamada cache que se utiliza para almacenar la posición del último evento procesado por la integración.

Integración con HRSoft

HRSoft es una plataforma para la gestión de compensación y bonificación de la compañía. La integración con HRSoft pretende sincronizar la información de empleados desde Bamboo hacia HRSoft. Dentro de la información que se sincroniza está: Información básica, información de salario y bonificación. En la Fig. 10 se puede ver como el flujo de datos va desde Bamboo HR, pasa a una base de datos SQL Server por medio de un Logic app, para que después los archivos planos sean creados y enviados a un servidor SFTP por medio de Data Factory.

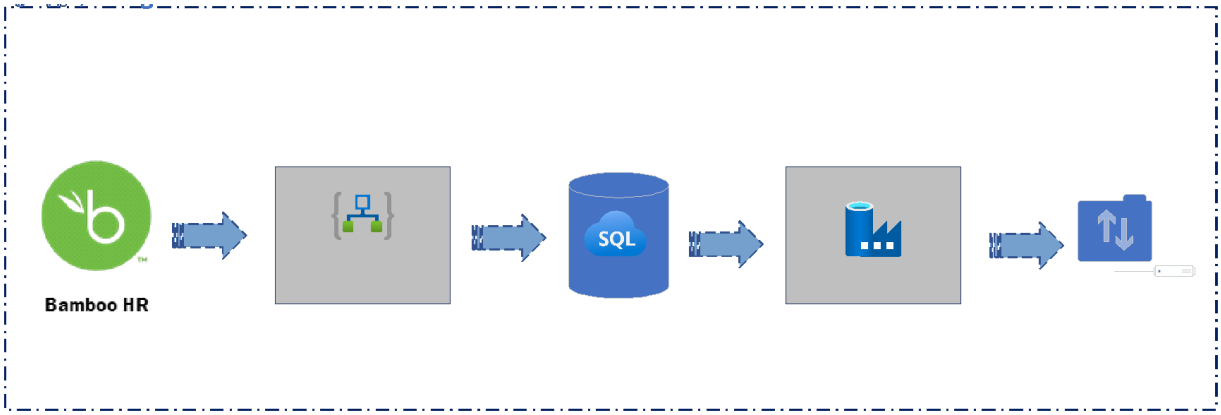

Fig. 10 Diagrama de componentes de la integración con HRSoft

En la Tabla 6 se describen los servicios que hacen parte de la integración y que son indispensables para que la esta tenga un funcionamiento óptimo. La integración usa el patrón de Aplicaciones Contextualizadas que fue descrito anteriormente en el marco teórico.

Tabla 6 Descripción de los componentes de la integración con HRSoft

Servicio	**Nombre**	**Descripción**
Sistema de Recursos Humanos	Bamboo	Sistema origen donde los datos de los empleados son administrados.
Sistema de gestión de compensación	HRSoft	Sistema destino donde se administra la compensación y bonificación de los empleados dentro de la compañía.
Logic app	logic-hrcompensation-tomds	Logic app que transfiere los datos de salario y bonificación desde Bamboo a la base de datos de integraciones.
Data Factory	df-hrsoftintgr	Data Factory en el cual se crea flujos ETL para crear los archivos planos necesarios

		para que HRSoft tenga los datos del sistema de recursos humanos.
SFTP Server	HRSoft SFTP server	SFTP server donde se almacenan los archivos planos que HRSoft utilizará para la alimentación de los datos que necesita para su funcionamiento.

VIII. CONCLUSIONES

La compañía comienza a adoptar prácticas modernas de desarrollo de software sin ser su eje de negocio la implementación de sistema informáticos. Se reconoce que se necesitan incluir prácticas de la industria de software en los procesos, así no sea el núcleo de la compañía, cuando se desarrollan proyectos de software.

Microsoft ofrece un ecosistema de servicios para manejar el ciclo de vida de las aplicaciones de forma consistente. En el proyecto se usa Office para la creación de diseños y documentación, SharePoint para el almacenamiento de estos, Azure DevOps para la gestión del proyecto, repositorio de código fuente y despliegues automatizados; y Azure para el alojamiento, funcionamiento y monitoreo de las integraciones.

Azure es una plataforma en la nube muy robusta que permite crear aplicaciones modernas con servicios existentes de forma consistente y, además, facilita la implementación de diferentes tecnologías, como IaC, donde se utiliza su estructura base como lo es ARM.

La Infraestructura como código se convierte en el mejor aliado para crear, modificar y configurar recursos Azure de manera confiable por las tecnologías detrás de la plataforma como los API ARM y la facilidad de crear plantillas ARM, con el propósito de desplegar los servicios Azure de manera consistentes a los diferentes ambientes.

IX. RECOMENDACIONES

El equipo de Integraciones has evidenciado las ventajas de la IaC en Azure como lo son:

- Consistencia en los despliegues de los recursos de Azure.
- Las plantillas ARM contienen buenas prácticas y seguridad que pueden ser aprovechadas para el despliegue de nuevos recursos de forma repetida.
- Las platillas ARM son reutilizables.
- Documentación de los despliegues por medio del código Bicep.
- Despliegue de recursos de forma automatizada por medio de Azure DevOps pipelines.

Se sugiere que la compañía continue fortaleciendo esta práctica y la pueda utilizar en los demás equipos de IT que hacen uso de Azure para la operación y para soportar al negocio.

REFERENCIAS

[1] Rossberg, J. (2019). Agile project management with azure DevOps: Concepts, templates, and metrics. Apress.

[2] Wikipedia contributors. (2023, October 5). Microsoft Azure. In Wikipedia, The Free Encyclopedia. Retrieved 21:09, October 8, 2023, from https://en.wikipedia.org/w/index.php?title=Microsoft_Azure&oldid=1178748902

[3] Morris, K. (2020). Infrastructure as code. O'Reilly Media.

[4] Rawat, S., Narain, A., Rawat, S., & Narain, A. (2019). Introduction to Azure Data Factory. Understanding Azure Data Factory: Operationalizing Big Data and Advanced Analytics Solutions, 13-56.

[5] Rendón, D. (2022). Building Applications with Azure Resource Manager (ARM): Leverage IaC to Vastly Improve the Life Cycle of Your Applications. Apress.


IEEE
Advancing Technology
for Humanity


UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3


JS|HELD