



# Manual de Despliegue CI/CD Appian–GitHub

*Operaciones Appian*



**Organización:** VR Group / BiceVida  
**Producto:** Pipeline CICD Appian & GitHub  
**Responsable:** Ángel Barroyeta  
**Consultor y Desarrollador:** Maximiliano Tombolini  
**Fecha:** Octubre 2025  
**Versión del documento:** 0.4.1

# Índice

Glosario de términos y abreviaciones	3
Resumen Ejecutivo	5
Alcance y Audiencia	6
1. Flujo general del pipeline	8
2. Entornos, roles y permisos	9
3. Incorporación de aplicaciones Appian al sistema CI/CD	11
4. Inputs del workflow (UI de Actions)	15
5. Flujo A — Aplicación base	18
6. Flujo B — Package + ICF	21
7. Flujo C — Package + SQL / Plugins	25
8. Criterios de aplicación de los flujos A, B y C	28
9. Checklist post-despliegue	30
10.Arquitectura de alto nivel del sistema CI/CD	34
10.1. Visión general . . . . .	34
10.2. Componentes principales . . . . .	34
10.2.1. Repositorios <i>wrapper</i> . . . . .	34
10.2.2. Repositorio <code>appian-cicd-core</code> . . . . .	35
10.2.3. Integración con Appian Cloud . . . . .	35
10.3. Flujos soportados y relación entre componentes . . . . .	35
11.Estructura de repos y artefactos	36

<b>12.Seguridad y secretos</b>	<b>36</b>
<b>13.Rollback y recuperación</b>	<b>36</b>
<b>14.Troubleshooting / FAQ</b>	<b>36</b>
<b>15.Control de cambios del documento</b>	<b>36</b>

## Glosario de términos y abreviaciones

Término / Sigla	Descripción
<b>Appian</b>	Plataforma low-code utilizada para desarrollar y desplegar aplicaciones empresariales.
<b>CI/CD</b>	Integración y Despliegue Continuos. Conjunto de prácticas que automatiza la entrega de artefactos entre entornos (DEV, QA, PROD).
<b>Wrapper</b>	Repositorio GitHub específico de una aplicación Appian. Orquesta los workflows de despliegue y resuelve secretos o variables propias.
<b>Core</b>	Repositorio central del sistema CI/CD. Contiene los workflows reutilizables y la lógica común.
<b>GitHub Actions</b>	Plataforma de automatización de GitHub que ejecuta los workflows (.yml) definidos en los repositorios wrapper.
<b>Workflow</b>	Ejecución orquestada en GitHub Actions que agrupa uno o más <i>jobs</i> para exportar, procesar, aprobar e importar artefactos.
<b>Deploy App</b>	Workflow utilizado para promover una aplicación Appian completa entre entornos (por ejemplo, DEV→QA→PROD). Incluye todos los objetos estándar de la aplicación.
<b>Deploy Package</b>	Workflow utilizado para promover un paquete específico exportado desde Appian Designer. Se usa en entregas incrementales, scripts SQL o plug-ins incluidos en el package.
<b>Deployment</b>	Ejecución de un flujo de promoción Appian iniciada desde GitHub (por ejemplo, dev-to-qa o qa-to-prod).
<b>Release</b>	Versión empaquetada en GitHub que se genera automáticamente cuando un despliegue a <b>PROD</b> finaliza con éxito e incluye los artefactos resultantes.
<b>Flujos A/B/C</b>	Modalidades de despliegue: A (base), B (con overrides/ICF), C (package con SQL o plug-ins).
<b>ICF</b>	<i>Import Customization File</i> . Archivo que permite sobrescribir valores dependientes de entorno (URLs, credenciales, IDs).
<b>ICF_JSON_OVERRIDES</b>	Secreto del repositorio (QA y/o PROD) con los valores concretos a aplicar sobre el ICF base.
<b>Artifact</b>	Archivo generado por el pipeline: .zip exportado desde Appian, logs, metadatos, plantillas de ICF o scripts SQL.
<b>Secrets</b>	Variables seguras en GitHub que almacenan credenciales o configuraciones sensibles utilizadas por los workflows.
<b>Environment Gate</b>	Aprobación manual en GitHub asociada a los entornos <b>qa</b> o <b>prod</b> , que detiene la promoción hasta que un revisor la aprueba.
<b>Idempotencia</b>	Propiedad deseable de los scripts SQL para que su re-ejecución no genere errores ni efectos adicionales.

<b>Término / Sigla</b>	<b>Descripción (continuación)</b>
<b>Rollback</b>	Recuperación manual a un estado anterior (por ejemplo, re-importando una aplicación o paquete previo, o usando el release generado).

## Resumen Ejecutivo

El presente manual tiene como objetivo guiar al equipo de **BiceVida** en el uso y operación del sistema de **Despliegue Automatizado CI/CD Appian–GitHub**, diseñado e implementado por **VR Group**. Este sistema corresponde a una arquitectura de entrega continua integrada entre Appian y GitHub, orientada a optimizar la trazabilidad, seguridad y eficiencia de los despliegues de aplicaciones.

El sistema permite realizar despliegues controlados y auditables entre los entornos **DEV**, **QA** y **PROD**, interactuando directamente con la [Appian Deployment REST API](#). A través de **GitHub Actions**, el usuario puede seleccionar el tipo de despliegue y ejecutar los pasos correspondientes de forma automatizada, sin requerir intervención manual dentro de Appian Designer.

La fase de **Continuous Integration (CI)** es gestionada de manera nativa por Appian, que se encarga de la inspección y validación de los paquetes antes de su promoción. El desarrollo se centra, por tanto, en la **Continuous Deployment (CD)**, donde se administran la promoción de artefactos, la configuración por entorno y las validaciones post-despliegue.

La arquitectura del sistema se compone de dos tipos de repositorios:

- **appian-cicd-core**: centraliza la lógica común, los workflows reutilizables y las acciones compartidas para los procesos de despliegue.
- **appian-transversal-\***: repositorios *wrapper* asociados 1:1 a cada aplicación Appian de BiceVida. Cada wrapper gestiona sus propias configuraciones, credenciales y trazabilidad de despliegues, garantizando independencia operativa y consistencia con la matriz CI/CD general.

El manual describe cómo operar el sistema desde GitHub, detalla los parámetros disponibles en la interfaz de ejecución y establece las pautas necesarias para validar resultados y resolver errores frecuentes. Asimismo, incluye lineamientos sobre seguridad, control de accesos y buenas prácticas de promoción, orientadas a mantener la integridad de los entornos y la consistencia del ciclo de despliegue.

Este documento busca servir como guía práctica para el equipo técnico de **BiceVida**, facilitando la adopción del sistema **CI/CD Appian–GitHub** como estándar operativo y sentando las bases para su futura extensión a otras aplicaciones de la organización.

## Alcance y Audiencia

Este manual define el alcance funcional y operativo del sistema de Despliegue Automatizado *CI/CD Appian-GitHub*, utilizado por **BiceVida**, y especifica a quién está dirigido y bajo qué condiciones debe ser aplicado.

### Alcance

El sistema permite ejecutar promociones controladas de aplicaciones y paquetes Appian entre los entornos **DEV**, **QA** y **PROD**, conectándose directamente con la [Appian Deployment REST API](#). Las ejecuciones se realizan desde *GitHub Actions*, donde el usuario selecciona el tipo de despliegue y el flujo correspondiente.

### Flujos de despliegue disponibles

#### Aplicaciones Appian

- DEV → QA
- DEV → QA → PROD
- QA → PROD

*Despliegues planificados y controlados de versiones estables.*

#### Paquetes Appian

- DEV → QA
- DEV → QA → PROD

*Entregas continuas y despliegues frecuentes.*

El sistema estandariza la promoción y trazabilidad de artefactos, gestiona configuraciones por entorno mediante archivos de propiedades, protege credenciales con *GitHub Secrets* y ejecuta validaciones básicas basadas en la respuesta de la API y los logs generados.

Actualmente no contempla:

- Mecanismos de *rollback* automático. La reversión, cuando sea necesaria, debe realizarse manualmente promoviendo una versión anterior (disponible en el repositorio) o restaurando los objetos directamente desde Appian Designer.
- Generación automática de paquetes.
- Promoción de *plug-ins* ni archivos de configuración de la *Admin Console*.
- Monitoreo continuo o métricas post-despliegue.

### Audiencia

Este documento está dirigido a los siguientes perfiles del equipo de BiceVida:

- **Desarrollador Appian:** prepara los paquetes o aplicaciones, verifica su integridad y comunica el nombre del artefacto a promover.
- **Jefe de Proyecto (encargado de promoción):** ejecuta el flujo de despliegue en GitHub, revisa logs y valida el resultado en el entorno destino.

Durante el paso por QA, el responsable de pruebas (que puede coincidir con el promotor o con un miembro del equipo QA) valida la aplicación desplegada y autoriza la promoción a producción.

# 1. Flujo general del pipeline

El siguiente diagrama resume el flujo principal de promoción entre entornos dentro del sistema de Despliegue Automatizado *CI/CD Appian-GitHub*. El proceso se ejecuta en los repositorios **wrapper**, los cuales orquestan la lógica común alojada en **appian-cicd-core** y coordinan las etapas de exportación, procesamiento y promoción.

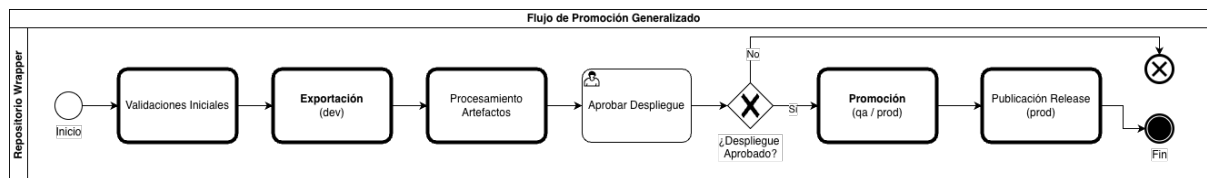


Figura 1: Flujo generalizado de promoción y despliegue entre entornos.

Bajo la notación de *GitHub Actions*, cada ejecución corresponde a un **workflow**, que agrupa un conjunto de **jobs** (representados en el diagrama como los bloques principales). Cada job está compuesto por una secuencia de **steps**, que implementan las tareas específicas del proceso, como la resolución de credenciales, la exportación de artefactos o la carga de archivos al repositorio de destino.

El flujo comienza con una serie de **validaciones iniciales**, seguidas por la **exportación del artefacto** desde el entorno de desarrollo (DEV). Una vez obtenido el paquete o aplicación, se realiza el **procesamiento de artefactos**, que incluye la generación y carga de los archivos de metadatos, versiones y configuración asociada.

Antes de proceder al despliegue en entornos superiores, se activa una **etapa de aprobación** (*gate*) mediante los *GitHub Environments*, asegurando control humano sobre las promociones. Solo tras la aprobación, el sistema ejecuta la **promoción** hacia QA o PROD, proceso que comprende la **inspección** y posterior **importación** del artefacto en el entorno destino.

Cuando el destino de promoción es **PROD** y la importación en Appian concluye con estado exitoso, el pipeline genera un **release en GitHub** asociado a esa ejecución. Este release consolida los artefactos producidos por el workflow (paquete **.zip**, plantillas de ICF, scripts SQL o plug-ins incluidos en el package, logs y metadatos) y sirve como punto formal de trazabilidad para auditorías o eventuales recuperaciones manuales.

Cada job genera **artefactos y registros** que se conservan en GitHub, permitiendo trazabilidad completa de los resultados y soporte para auditorías o reversión manual en caso necesario.

## 2. Entornos, roles y permisos

Esta sección define los *environments* utilizados por el pipeline, su relación con los entornos de Appian y los roles involucrados en el proceso de promoción entre **DEV**, **QA** y **PROD**.

### Entornos y mapeo

- **DEV** ↔ [dev-bicevida.appiancloud.com](https://dev-bicevida.appiancloud.com)
- **QA** ↔ [qa-bicevida.appiancloud.com](https://qa-bicevida.appiancloud.com)
- **PROD** ↔ [bicevida.appiancloud.com](https://bicevida.appiancloud.com)

Los *GitHub Environments* (**dev**, **qa**, **prod**) se definen dentro de cada repositorio **wrapper** para:

- Emular la entidad que ejecuta cada *job* (por ejemplo, exportación en **dev/qa**, promoción en **qa/prod**).
- Resolver de forma precisa las **API keys** correspondientes a cada entorno de Appian.
- Aplicar las **aprobaciones manuales** requeridas antes de promover a QA o PROD.

### Variables y secretos

Cada ejecución del pipeline utiliza un conjunto de variables y secretos definidos a distintos niveles:

**A nivel organización** (autenticación contra Appian):

- **APPIAN\_DEV\_API\_KEY**
- **APPIAN\_QA\_API\_KEY**
- **APPIAN\_PROD\_API\_KEY**

**A nivel wrapper** (configuración por aplicación y entorno):

- **Variables:** APP\_NAME, APP\_UUID
- **Secretos:** ICF\_JSON\_OVERRIDES\_QA, ICF\_JSON\_OVERRIDES\_PROD

### Roles operativos

- **Desarrollador Appian:** prepara el paquete de despliegue desde Appian Designer y comunica el *Package Name* al equipo de promoción.
- **Jefe de Proyecto (promotor):** ejecuta el workflow correspondiente en GitHub Actions, selecciona la ruta de promoción, revisa los logs y aprueba los despliegues entre entornos.
- **QA / Revisor:** valida el comportamiento de la aplicación desplegada en QA y aprueba la promoción hacia producción.

## Permisos requeridos en GitHub

Para ejecutar un despliegue dentro del sistema, el usuario debe contar con los permisos adecuados sobre el repositorio **wrapper** correspondiente a la aplicación Appian.

- **Rol mínimo: Write.** Permite ejecutar manualmente los workflows desde *GitHub Actions* y acceder a los entornos configurados.
- **Rol adicional para aprobadores:** Los usuarios encargados de autorizar promociones hacia QA o PROD deben ser agregados explícitamente en la lista de *Required reviewers* de cada *Environment* (qa, prod) dentro de la configuración del repositorio (**Settings** → **Environments**).

Los permisos específicos requeridos por el sistema son:

- *Run workflow:* ejecutar manualmente los flujos de despliegue.
- *Approve deployment:* autorizar la promoción cuando el flujo alcanza un entorno con *gate* activo.
- *Read repository secrets/variables:* acceder a los secretos y variables necesarios para la autenticación y configuración del pipeline.

## Matriz operativa (RACI)

Actividad	Des. Appian	Jefe de Proyecto	QA / Revisor
Preparar paquete en Appian Designer	R	C	I
Ejecutar workflow de despliegue	I	R	C
Aprobar promociones (QA / PROD)	I	A	R
Validar aplicación desplegada en QA	I	C	R
Revisión de logs y trazabilidad post-despliegue	C	R	C

*Leyenda: R = Responsable, A = Aprueba, C = Colabora, I = Informado.*

### Resumen de responsabilidades

El **Desarrollador Appian** prepara el paquete o aplicación desde Appian Designer, verificando su integridad antes del traspaso. Comunica los detalles del paquete a promover. El **Jefe de Proyecto (promotor)** opera el flujo en *GitHub Actions*, selecciona el tipo de despliegue, revisa logs y gestiona las aprobaciones requeridas. El **QA o revisor** valida la aplicación desplegada en el entorno de QA y aprueba la continuación hacia producción, asegurando la calidad funcional.

### 3. Incorporación de aplicaciones Appian al sistema CI/CD

Cada aplicación Appian integrada al sistema CI/CD requiere su propio **repositorio wrapper**, el cual actúa como capa operativa frente al repositorio central `appian-cicd-core`. Este wrapper permite ejecutar los flujos de despliegue (*Deploy App* y *Deploy Package*) con sus respectivas aprobaciones, variables y secretos, asegurando trazabilidad y control de versiones por aplicación.

#### 1. Crear el repositorio wrapper

- Crear un nuevo repositorio en GitHub bajo la organización de **Bice Vida**.
- Utilizar la siguiente nomenclatura estandarizada: `transversal-1154-appian_[nombre_app]`.
- Seleccionar la opción “**Create from template**” y elegir el repositorio base `appian-cicd-wrapper`.
- Definir la visibilidad del repositorio como **Privada**.

#### 2. Crear los entornos (environments)

En el nuevo repositorio, ingresar a *Settings* → *Environments* y crear los siguientes entornos:

- `dev`
- `qa`
- `prod`

Para los entornos `qa` y `prod`, activar la opción **Require reviewers** e incluir a los usuarios responsables de aprobar los despliegues (habitualmente Jefes de Proyecto o QA designados). Esto garantiza que las promociones entre entornos requieran validación humana antes de ejecutarse.

#### 3. Definir secretos y variables

Agregar en *Settings* → *Secrets and variables* → *Actions* los parámetros específicos de la aplicación:

Nombre	Tipo	Descripción
ICF_JSON_OVERRIDES_QA	Secret	Archivo JSON con configuraciones específicas para el entorno QA (constantes, connected systems, etc.).
ICF_JSON_OVERRIDES_PROD	Secret	Archivo JSON con configuraciones específicas para el entorno PROD.
APP_NAME	Variable	Nombre exacto de la aplicación Appian.
APP_UUID	Variable	Identificador único de la aplicación en Appian Designer.

### Nota técnica

Los secretos ICF\_JSON\_OVERRIDES\_QA y ICF\_JSON\_OVERRIDES\_PROD deben crearse inicialmente con el valor {} (JSON vacío). Esto se debe a que GitHub no permite definir secretos sin contenido. El completado de estos secretos, explicado en el **Flujo B**, corresponde a cada ejecución del sistema, por lo que su valor es dinámico. No es necesario, ni tiene sentido práctico, asignarles un contenido previo a la ejecución.

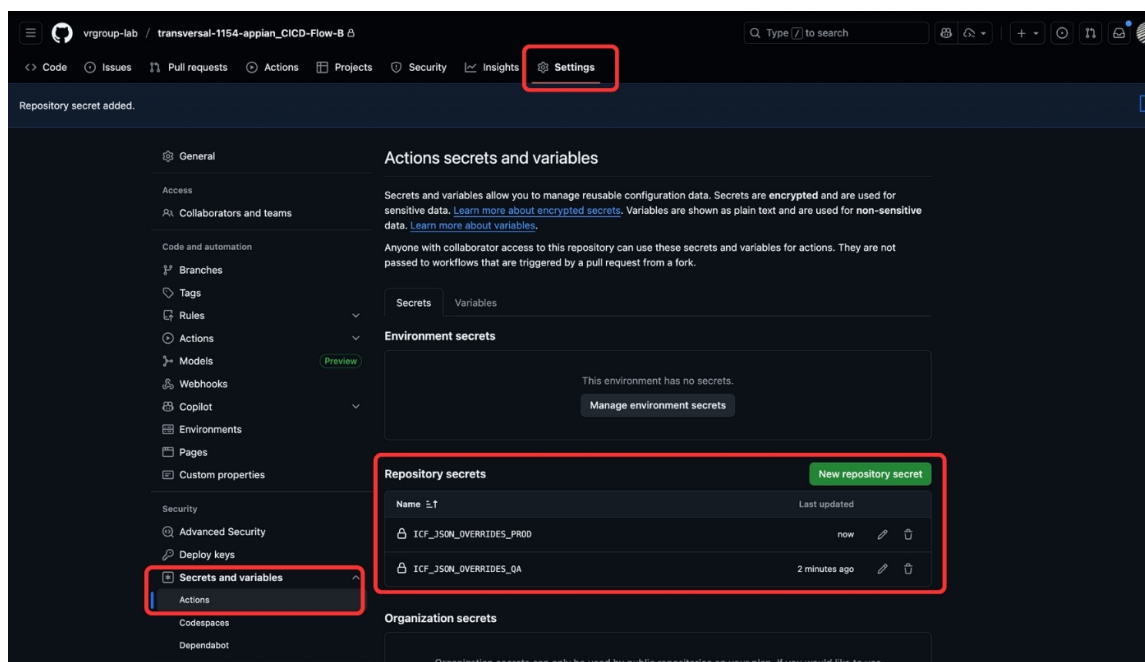
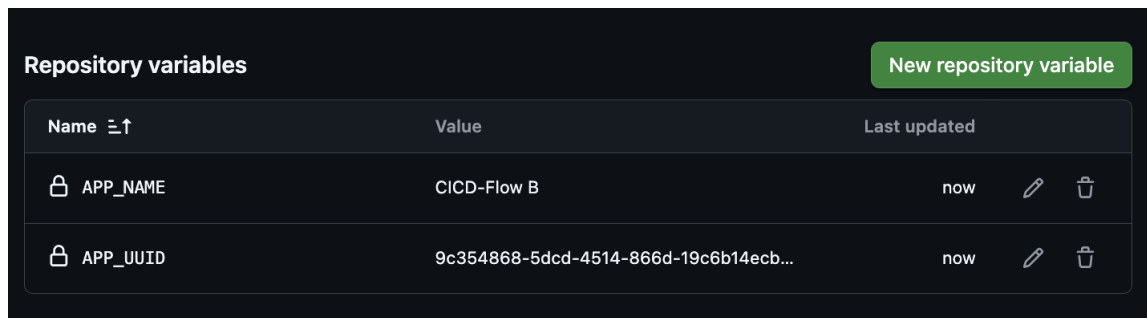


Figura 2: Configuración de secretos en el repositorio wrapper.



Name ↕	Value	Last updated
APP_NAME	CICD-Flow B	now
APP_UUID	9c354868-5dcd-4514-866d-19c6b14ecb...	now

Figura 3: Configuración de variables (APP\_NAME y APP\_UUID).

*Nota:* Las API keys necesarias para autenticar con los entornos Appian (DEV, QA, PROD) ya están configuradas a nivel organizacional y no requieren acción manual.

## 4. Validar conexión con Appian

Una vez completada la configuración, ejecutar manualmente el workflow **Deploy App** o **Deploy Package** seleccionando la ruta **dev-to-qa**. Si la integración fue correcta, la ejecución finalizará exitosamente generando los artefactos de exportación y los logs correspondientes.

En caso de error, revisar la consola de ejecución en **Actions** → **Run workflow** → **Logs**, verificando:

- Que las API keys estén activas y asociadas correctamente.
- Que el APP\_UUID y la URL del entorno sean válidos.
- Que el usuario tenga permisos de **write** en el repositorio.

## 5. Gestionar acceso del equipo

Si más de una persona participa en los despliegues de una aplicación, se deben agregar los colaboradores desde **Settings** → **Collaborators and teams**:

- Invitar a cada integrante con permiso **Write**, lo que permite ejecutar workflows, revisar artefactos y consultar logs.
- Si un integrante debe aprobar promociones (*gates*), debe agregarse también como **Required reviewer** dentro del entorno correspondiente (**qa** o **prod**).
- Los aprobadores pueden ser los mismos que ejecutan los despliegues o personas distintas, según el marco de control definido por cada proyecto.

### Nota

La correcta configuración del wrapper es un requisito previo para que una aplicación Appian pueda operar dentro del sistema CI/CD. Cada wrapper es autónomo y mantiene su propio ciclo de aprobaciones, secretos y usuarios, garantizando trazabilidad y control independiente por aplicación.

Una vez incorporada la aplicación al sistema CI/CD mediante su wrapper, los flujos de despliegue (**A**, **B** y **C**) pueden ejecutarse de forma controlada según el tipo de contenido Appian promovido.

## 4. Inputs del workflow (UI de Actions)

Al ejecutar manualmente un workflow desde *GitHub Actions*, el usuario debe ingresar a la pestaña **Actions** dentro del repositorio wrapper de la aplicación Appian correspondiente. Desde esta vista se listan los workflows disponibles, permitiendo seleccionar el tipo de despliegue según el objetivo y la rama desde la cual se ejecutará (por defecto, **main**):

- **Deploy App:** utilizado para promover una aplicación completa de Appian, incluyendo todos sus objetos asociados (records, interfaces, process models, expresiones, etc.) entre los distintos entornos definidos.
- **Deploy Package:** utilizado para promover un paquete específico exportado desde Appian Designer, que contiene un subconjunto de objetos de una aplicación preparado para entregas incrementales o controladas.

Una vez seleccionado el flujo deseado, se despliega un formulario donde el usuario define los parámetros del despliegue, incluyendo la **rama de origen** y la **ruta de promoción**. Desde esta misma vista también es posible revisar las ejecuciones previas, junto con su estado (**éxito**, **en curso**, **error**), el usuario ejecutor y la duración de cada job.

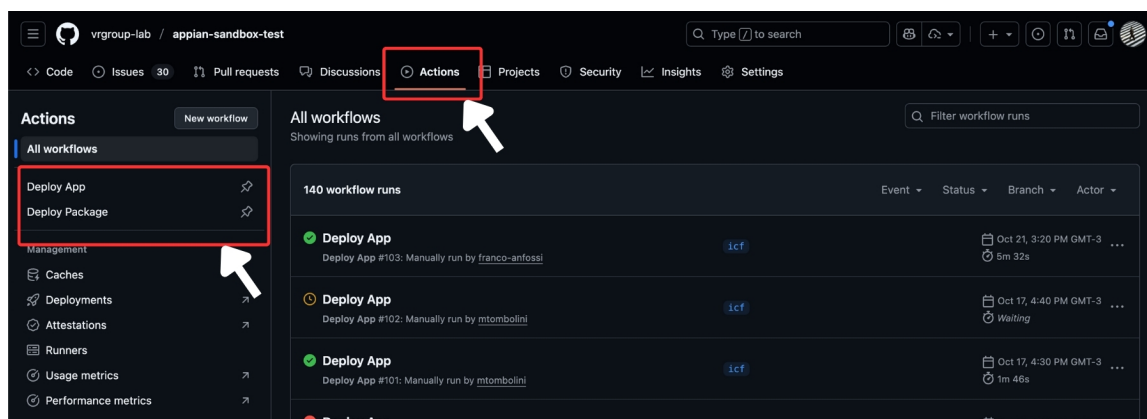


Figura 4: Acceso a los workflows de despliegue en GitHub Actions.

### Inputs visibles en la UI

**Workflow *Deploy App*** El usuario selecciona la rama de ejecución (por defecto, **main**) y la ruta de promoción. Las variables de la aplicación (**APP\_NAME**, **APP\_UUID**) ya están definidas en el repositorio wrapper y no se solicitan en la interfaz.

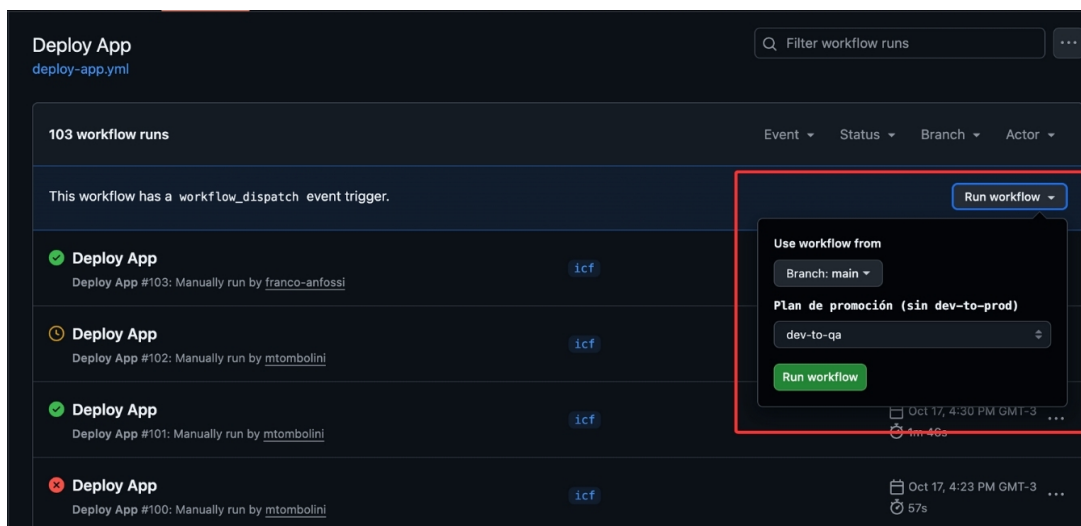


Figura 5: Inputs visibles del workflow *Deploy App*.

Input	Tipo	Descripción	Requerido
branch	choice	Rama de ejecución del workflow. Por defecto, <code>main</code> .	Sí
route	choice	Plan de promoción: <code>dev-to-qa</code> , <code>dev-qa-prod</code> , <code>qa-to-prod</code> .	Sí

**Workflow *Deploy Package*** El flujo de promoción de paquetes solicita el nombre del artefacto exportado desde Appian Designer, la ruta de despliegue y la rama de ejecución (generalmente `main`).

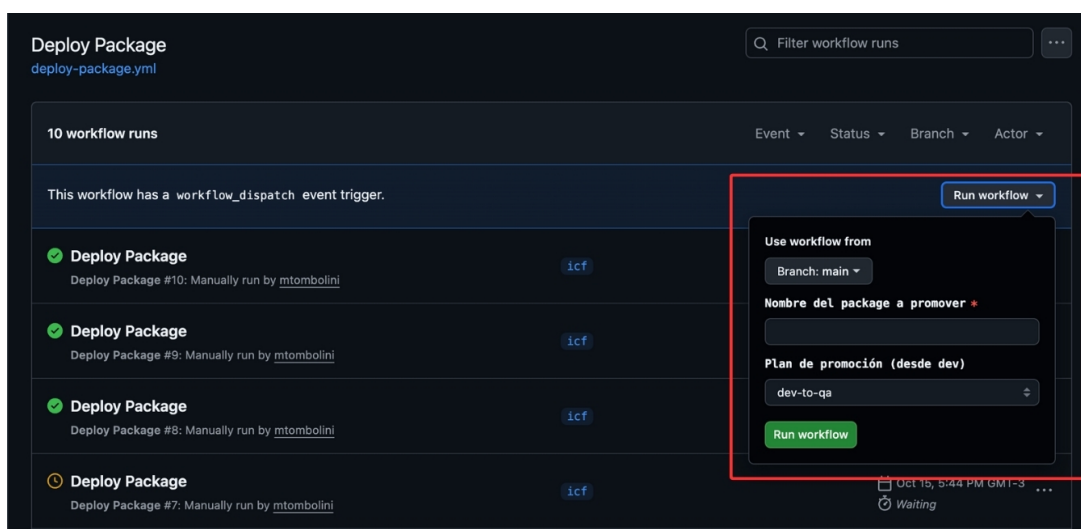


Figura 6: Inputs visibles del workflow *Deploy Package*.

Input	Tipo	Descripción	Requerido
branch	choice	Rama de ejecución del workflow. Por defecto, <code>main</code> .	Sí
package_name	string	Nombre exacto del package a promover (tal como figura en Appian Designer).	Sí
route	choice	Plan de promoción: <code>dev-to-qa</code> , <code>dev-qa-prod</code> .	Sí

## Salidas (artefactos publicados)

Cada ejecución genera artefactos y registros accesibles desde la pestaña de *Actions*. En caso de un despliegue exitoso, el sistema publica los siguientes resultados:

- **Paquete exportado (.zip)** con el contenido del despliegue.
- **Metadatos de exportación**, incluyendo UUID, timestamp y entorno origen/destino.
- **Archivos complementarios**: según el flujo, pueden incluir el ICF template (`.properties`), scripts SQL, archivos de configuración o paquetes de plug-ins.
- **Logs detallados** de cada job y un resumen consolidado de promoción.

### Funcionalidad en desarrollo

En futuras versiones del sistema, los artefactos exitosos serán empaquetados automáticamente en un **Release de GitHub**, asociado a la ejecución del workflow. Esto permitirá mantener trazabilidad de versiones y facilitar auditorías o rollback manuales.

## 5. Flujo A — Aplicación base

El **Flujo A** representa el escenario base del sistema de despliegue, utilizado para promover una aplicación Appian completa entre entornos (DEV, QA, PROD) sin requerir configuraciones adicionales. Incluye únicamente objetos estándar de Appian y no demanda intervención manual más allá de las aprobaciones correspondientes en los *gates* de promoción.

### Objetos incluidos

El **Flujo A** contempla todos los objetos de aplicación que pueden promoverse entre entornos sin requerir configuraciones adicionales ni dependencias de entorno. Estos son:

- **Data objects:** CDT.
- **Process objects:** Process Model, Process Report, Robotic Task, Robot Pool.
- **User objects:** Interface, Report, Site, Portal, Control Panel.
- **Rule objects:** Constant (sin dependencia de entorno), Decision, Expression Rule, AI Skill, Translation Set.
- **Integration objects:** Integration, Web API.
- **Group objects:** Group, Group Type.
- **Content-management objects:** Document, Folder.
- **Notification objects:** Feed.

Todos estos objetos mantienen su comportamiento original al ser importados y no requieren parametrización mediante archivos de configuración ni valores contextuales.

### Objetos excluidos

Los siguientes objetos no se despliegan mediante este flujo, ya que requieren configuración adicional o parámetros específicos por entorno:

- **Record Types**
- **Data Stores**
- **Connected Systems**
- **Constants** con valores dependientes de entorno

Estos objetos son gestionados a través de los **Flujos B y C**, que extienden la lógica del presente flujo base.

### Ejecución del flujo

El usuario debe acceder al repositorio *wrapper* correspondiente, seleccionar el workflow de despliegue (**Deploy App** o **Deploy Package**) y ejecutarlo desde la rama **main**. La única

entrada requerida en la interfaz es la variable `route`, que define la ruta de promoción (`dev-to-qa`, `dev-qa-prod` o `qa-to-prod`).

Durante la ejecución, el sistema realiza automáticamente las siguientes acciones:

1. Exporta la aplicación o paquete desde el entorno origen.
2. Genera un artefacto comprimido (`.zip`) junto con los metadatos y logs de exportación.
3. Solicita aprobación manual (*gate*) en el entorno destino según la ruta seleccionada.
4. Una vez aprobada, inspecciona e importa el artefacto en el entorno final, completando el proceso de promoción.

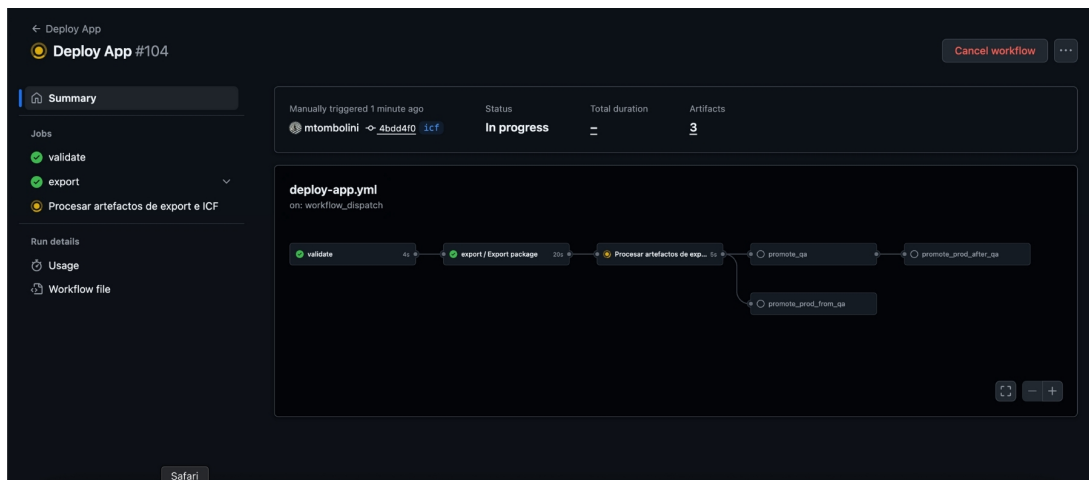


Figura 7: Ejecución del workflow *Deploy App* desde GitHub Actions.

## Aprobación del despliegue

Los entornos QA y PROD utilizan el mecanismo de aprobación nativo de GitHub Actions (*Environment Gate*). Cuando un flujo requiere aprobación, el job queda en espera hasta que un usuario autorizado (definido en la lista de **Required reviewers**) aprueba manualmente la promoción.

- Cada aprobación se realiza directamente desde la vista de ejecución del workflow.
- Solo los usuarios con permiso `write` sobre el repositorio wrapper y registrados como revisores pueden aprobar.
- Una vez aprobada, la ejecución continúa automáticamente hacia el siguiente entorno.

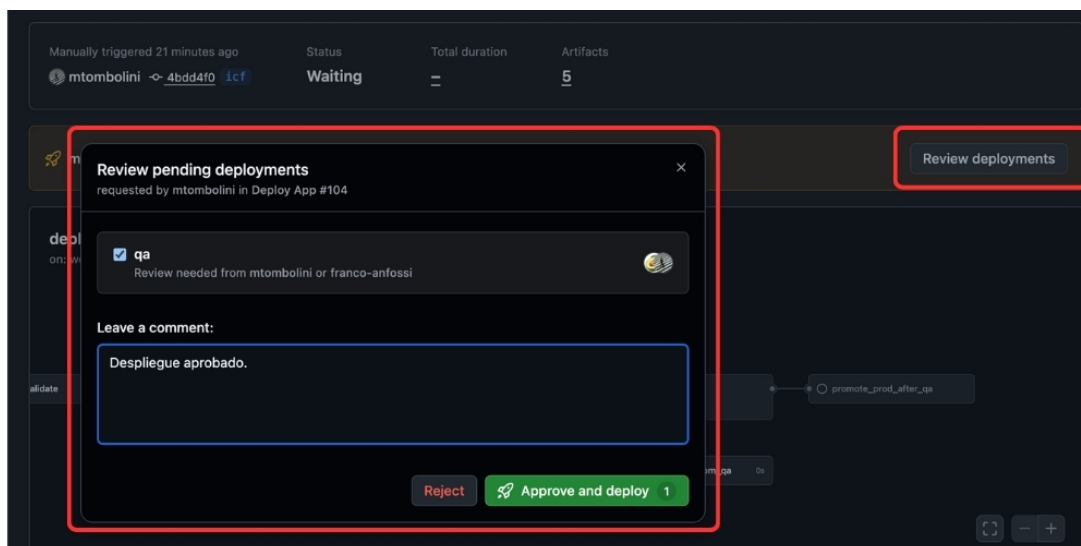


Figura 8: Aprobación manual del despliegue (*GitHub Environment Gate*).

## Resultado esperado

Una vez completado el flujo:

- La aplicación se despliega correctamente en el entorno de destino, conservando las dependencias internas.
- Los artefactos y logs quedan registrados en la pestaña de *Actions*, accesibles para auditoría o verificación posterior.
- No se requiere intervención manual adicional, salvo la revisión funcional en QA o PROD.

### Nota

El **Flujo A** constituye la base sobre la cual operan los flujos extendidos (**B** y **C**). Estos últimos añaden la gestión de objetos configurables por entorno, como *Connected Systems*, *Constants* o *Record Types*, pero comparten la misma arquitectura y secuencia operativa del presente flujo.

## 6. Flujo B — Package + ICF

El **Flujo B** extiende el comportamiento del flujo base, incorporando la gestión de configuración dependiente de entorno mediante **Import Customization Files (ICF)**. Este flujo se utiliza cuando la aplicación Appian o el paquete a desplegar contiene objetos que requieren parametrización por entorno de destino, como **Connected Systems**, **Data Stores** o **Constants** con valores variables. El flujo puede ejecutarse tanto a través de *Deploy App* como de *Deploy Package*.

### Objetos incluidos

El **Flujo B** se activa automáticamente cuando el sistema detecta la presencia de objetos configurables por entorno. Estos incluyen:

- **Connected Systems**
- **Data Stores**
- **Constants** con valores dependientes de entorno

El resto de los objetos Appian (Interfaces, Process Models, Reports, Sites, Rules, etc.) continúan promovidos bajo la misma lógica del **Flujo A**, sin requerir intervención adicional.

### Preparación previa al workflow (Desarrollador → Promotor)

1. **Desde Appian Designer (Desarrollador):** Durante la inspección o exportación de la aplicación, si Appian indica que se requiere un archivo de personalización por entorno, descargar la **plantilla .properties** (*customization file template*).
2. **Comunicación (Desarrollador → Promotor):** Informar al promotor que la promoción corresponde al **Flujo B** y **adjuntar la plantilla .properties**.
3. **Preparación del secreto (Promotor):**
  - a) Ingresar a **Settings → Secrets and variables → Actions → Repository secrets**.
  - b) Crear o actualizar el secreto correspondiente al entorno de destino (por ejemplo: `ICF_OVERRIDES_QA` o `ICF_OVERRIDES_PROD`, según la configuración del repositorio).
  - c) **Copiar las líneas relevantes de la plantilla, eliminar el prefijo # y completar los valores**, dejando una asignación por línea en formato `clave=valor`.
  - d) Guardar el secreto **antes de iniciar** la ejecución del workflow.

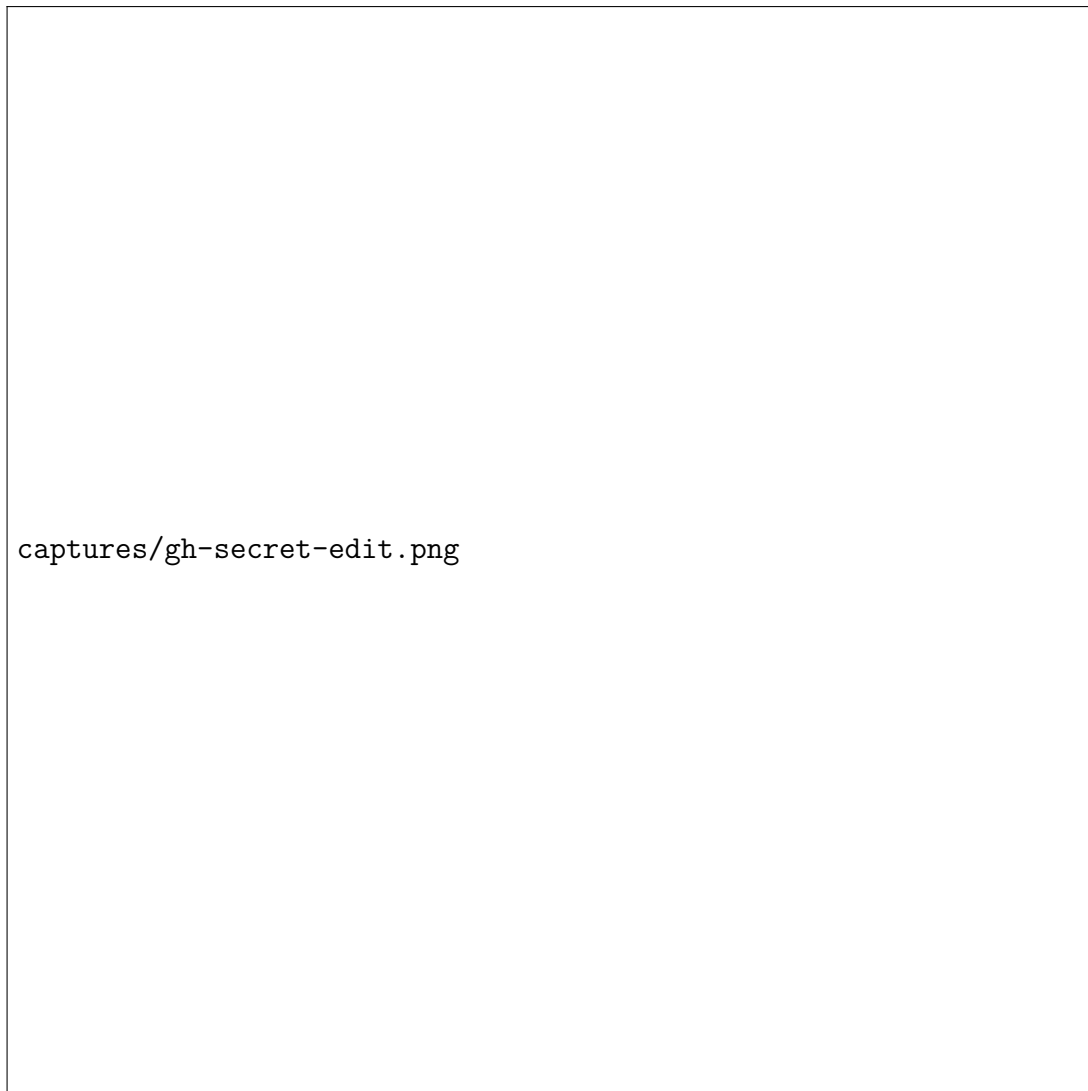


Figura 9: Edición del secreto del repositorio para el entorno objetivo. Pegar el contenido en formato `clave=valor`, una asignación por línea.

## Ejecución y validaciones en GitHub Actions

Al iniciar la acción de *Deploy App* o *Deploy Package*, el flujo realiza las siguientes operaciones:

- Lee el **secreto del repositorio** correspondiente al entorno objetivo.
- **Normaliza y valida** el contenido (ignora líneas vacías y las que comienzan con `#`; divide cada línea por el primer signo `=` preservando el resto del valor).
- **Genera** el archivo `artifacts/customization.properties` sin imprimir valores en los logs (sólo muestra el conteo de líneas válidas y una huella hash).
- Entrega la ruta del `customization.properties` al paso de importación (**ICF**) del pipeline, que reside en el repositorio core.
- Si el secreto **no existe**, está **vacío** o contiene **líneas inválidas**, el job **falla inmediatamente** con un mensaje orientativo (sin exponer datos sensibles).

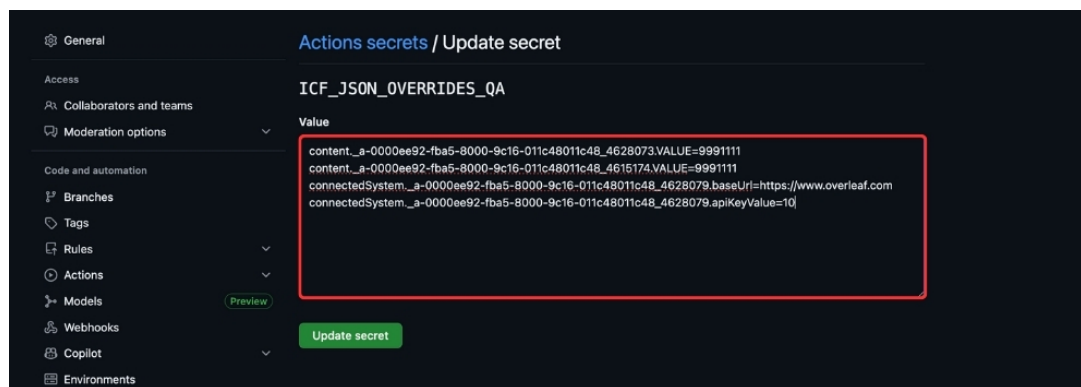


Figura 10: Construcción del archivo `customization.properties` desde el secreto. Se valida el formato y se reporta sólo el conteo de líneas válidas y una huella hash.

## Formato aceptado del secreto (texto plano)

### Formato del contenido del secreto

```
# Comentarios opcionales (se ignoran) y líneas en blanco permitidas
connectedSystem.<UUID>.baseUrl=https://example
connectedSystem.<UUID>.apiKeyValue=AAA
content.<UUID>.VALUE=10
importSetting.FORCE_UPDATE=true
```

#### Reglas:

- Una asignación por línea: `clave=valor`.
- Líneas vacías o que comiencen con `#` se ignoran.
- Se divide por el **primer** `=`; valores con `=` son válidos.
- Prefijos típicos: `connectedSystem.`, `content.`, `recordType.`, `importSetting.`.
- Se soportan fin de línea LF y CRLF.



Figura 11: Aprobación manual del gate previo a la promoción en GitHub Actions.

## Resultado esperado

- La promoción aplica correctamente las configuraciones personalizadas del ICF según el entorno destino.
- Se genera el archivo `artifacts/customization.properties`, manteniendo trazabilidad y auditoría completa en *Actions*.
- Las credenciales y configuraciones sensibles se gestionan de forma segura y efímera, sin exposición en texto plano.
- No se requieren issues intermedias ni re-ejecuciones manuales.

## 7. Flujo C — Package + SQL / Plugins

El **Flujo C** extiende la arquitectura de despliegue para cubrir los casos en que un paquete Appian contiene **scripts SQL** o **plug-ins (.zip)**. Se utiliza principalmente cuando la promoción involucra cambios en la estructura de la base de datos o la instalación de complementos de servidor.

### Propósito y alcance

Este flujo está diseñado exclusivamente para la promoción de **packages**. Su uso garantiza que los artefactos que afectan la base de datos o el entorno de ejecución se gestionen de forma controlada, evitando mezclar cambios estructurales con lógica de aplicación.

Aunque un package SQL puede incluir otros objetos Appian, la buena práctica es mantenerlo **exclusivo para scripts SQL o plug-ins**. Esto facilita la trazabilidad, reduce riesgos de error y permite una sincronización automática de los Record Types en los despliegues posteriores.

### Preparación del paquete en Appian

La preparación recae en el **desarrollador**, antes de iniciar la promoción. Para casos con cambios en la base de datos, se deben generar **dos packages independientes**:

- [código](sql) → contiene los archivos **SQL y/o plug-ins**.
- [código](pkg) → contiene los **objetos Appian** asociados (interfaces, rules, record types, etc.).

Ejemplo: AA-15(sql) y AA-15(pkg). Esta separación permite que, al ejecutar primero el package SQL, los cambios en la base de datos queden aplicados antes de importar los objetos Appian, habilitando la sincronización automática de los Record Types.

El orden correcto de despliegue es siempre:

[Package SQL] → [Package Appian]

En Appian Designer, el desarrollador debe cargar los archivos en:

Appian Designer → Packages → [Seleccionar package] → Data Scripts /  
Plug-ins

Allí define el orden de ejecución y valida que todos los scripts se hayan probado previamente en **dev**. Una vez listo, el desarrollador comunica al equipo de despliegue los nombres exactos de los packages a promover.

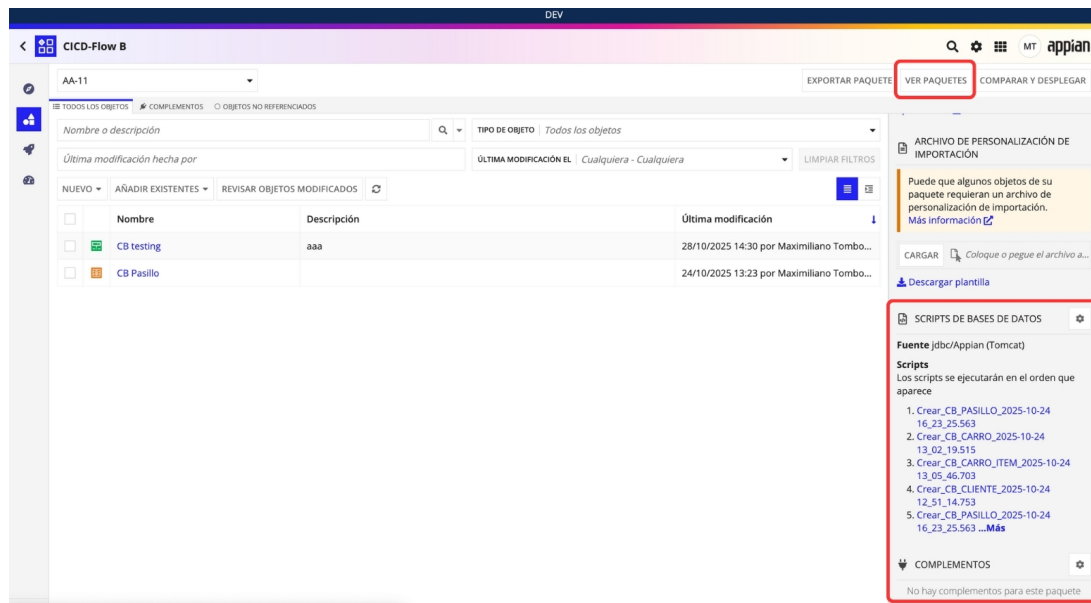


Figura 12: Configuración de los scripts de base de datos dentro de un package Appian.

## Ejecución del flujo

El **Flujo C** se ejecuta desde el repositorio *wrapper* mediante el workflow:

- **Deploy Package:** promueve el package que contiene los scripts SQL y/o plug-ins cargados.

Si el package SQL incluye objetos dependientes de entorno (constants, data stores, etc.), el sistema aplicará además las reglas del **Flujo B** para procesar los valores definidos en los archivos ICF.

## Comportamiento de ejecución en Appian

Appian ejecuta los scripts SQL según el orden definido por el desarrollador. El sistema no valida dependencias ni comprueba si las tablas ya existen; por lo tanto:

- Reimportar un mismo package volverá a ejecutar todos los scripts.
- Si un objeto ya existe, se producirá un error SQL (`Table already exists`).
- Los cambios aplicados antes del error permanecen, sin rollback automático.

## Sincronización de Record Types

Cuando el flujo C se ejecuta en primer lugar (package SQL → package Appian), los Record Types asociados se sincronizan automáticamente tras la importación. Esto garantiza que los modelos de datos reflejen la nueva estructura de base de datos sin intervención manual.

Si el orden de despliegue se invierte (primero el package Appian y luego el SQL), la sincronización debe realizarse manualmente desde Appian Designer.

## Idempotencia

Los archivos `.sql` generados automáticamente por Appian no cumplen completamente con las propiedades de **idempotencia**, ya que su ejecución repetida puede provocar errores o duplicación de objetos (por ejemplo, tablas ya existentes). Por esta razón, se recomienda —a criterio del **jefe de equipo** o **líder técnico**— evaluar la conveniencia de transformar estos scripts a versiones idempotentes antes de su promoción, especialmente en entornos productivos.

Incorporar validaciones como `IF NOT EXISTS` o condiciones previas a las operaciones de creación y eliminación puede prevenir fallos en ejecuciones posteriores o reimportaciones accidentales.

## Resultado esperado

Tras completar el flujo:

- El package SQL se ejecuta correctamente y aplica los cambios definidos.
- Los Record Types quedan sincronizados (automáticamente si el orden fue correcto).
- Los logs registran la secuencia y resultado de cada archivo ejecutado.
- Si la promoción corresponde a un despliegue exitoso hacia **prod**, el pipeline genera un **release automático** que incluye los artefactos del despliegue (`.zip`, `.sql`, logs y metadatos).

## 8. Criterios de aplicación de los flujos A, B y C

Esta sección sintetiza **cuándo usar** cada flujo de promoción y cómo se relacionan entre sí. El **Flujo A** constituye la base (sin configuración adicional); el **Flujo B** extiende con *ICF/overrides* para objetos dependientes de entorno; y el **Flujo C** amplía el alcance a paquetes que incluyen **scripts SQL** o **plug-ins (.zip)**, actuando como complemento en la promoción de aplicaciones.

### Diagrama de decisión (cuándo se gatilla cada flujo)

El siguiente diagrama resume la lógica de selección del flujo adecuado según el tipo de artefacto y los requisitos de configuración detectados durante la promoción:

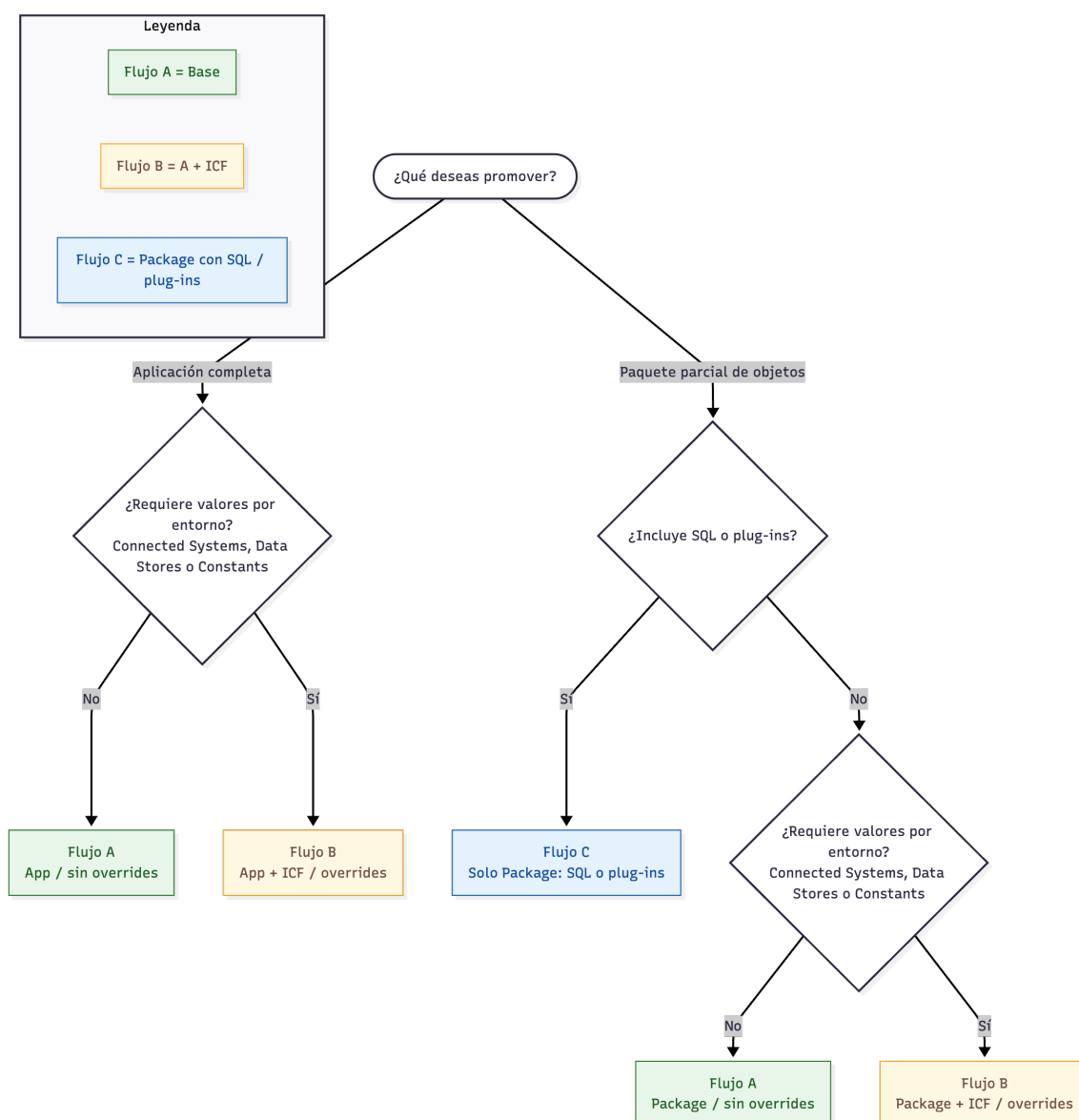


Figura 13: Árbol de decisión para la aplicación de los flujos A, B y C.

## Resumen comparativo

Flujo	Aplicación típica	Contenido habitual	Acciones del usuario / notas clave
<b>A</b>	App o Package sin dependencias	Objetos estándar: Interfaces, Rules, Process Models, Reports, Sites.	Solo aprobaciones ( <i>gates</i> ). Flujo base del sistema.
<b>B</b>	App o Package con dependencias de entorno	Connected Systems, Data Stores, Constants con valores distintos.	Completar los <code>ICF_JSON_OVERRIDES_{QA,PROD}</code> según issue generada y re-ejecutar el job.
<b>C</b>	Package con componentes técnicos adicionales	Scripts SQL, plug-ins (.zip) y otros complementos definidos en Appian Designer.	El desarrollador prepara el package (carga y orden). Luego se promueve normalmente.

## Caso combinado (B + C)

Puede ocurrir que un mismo *package* incluya tanto **scripts SQL o plug-ins** (Flujo C) como **objetos con valores dependientes de entorno** (Flujo B). En estos casos:

- El desarrollador realiza la preparación del paquete en Appian Designer (carga y orden de scripts o plug-ins).
- Durante la promoción, el sistema detecta los objetos configurables y genera la *issue* correspondiente al Flujo B.
- Se deben completar los `ICF_OVERRIDES` y continuar con la aprobación normal.

El flujo combinado mantiene la misma secuencia de validación y despliegue que los flujos individuales, garantizando consistencia en la trazabilidad y control del proceso.

### Nota técnica

Por buenas prácticas, se recomienda mantener el **Flujo C aislado** del resto de promociones, ya que los cambios en base de datos no poseen *rollback automático*. Separar el despliegue SQL permite mantener la base de datos libre de errores, asegurar su integridad y facilitar el control de versiones entre entornos.

## 9. Checklist post-despliegue

Esta sección reúne las acciones recomendadas una vez completado un despliegue exitoso en **qa** o **prod**. Si la ejecución del workflow finaliza con estado **Success**, el proceso se considera completado correctamente. Las siguientes verificaciones son complementarias y permiten confirmar resultados, depurar errores o validar detalles del despliegue.

### Validaciones generales

- Confirmar que el workflow finalizó con estado **Success** en GitHub Actions.
- Revisar, en caso necesario, los logs del job **Promote**, que contiene las etapas de inspección e importación hacia el entorno destino.
- Si se detectan errores o comportamientos inesperados, documentarlos en el tablero o registro interno de despliegues del equipo.

### Validaciones en Appian

- Ingresar al entorno de destino en Appian Designer y, desde el **panel lateral izquierdo**, acceder a la sección **Despliegues**.
- Revisar la pestaña **Entrantes (Incoming)** para confirmar la correcta importación del artefacto y su historial de ejecución.
- (Opcional) Revisar la pestaña **Salientes (Outgoing)** en el entorno origen para consultar los detalles del proceso de exportación.
- Verificar que los objetos importados (interfaces, procesos, reglas, etc.) estén disponibles y sin referencias rotas.
- Si el despliegue corresponde a un **Flujo C (SQL)**, validar en **Appian Cloud Database** que las tablas afectadas reflejen los cambios esperados.

### Validaciones en GitHub

- Verificar que la ejecución del workflow fue exitosa y que los artefactos del despliegue (.zip, .sql, .icf\_template, logs y metadatos) aparecen como artefactos asociados a la ejecución en la pestaña *Actions*.
- (Opcional) Acceder al detalle de la ejecución y expandir los **jobs individuales** para revisar los **logs por paso**. Esta vista permite inspeccionar los comandos ejecutados, tiempos de respuesta y resultados de los endpoints de Appian.
- Cerrar las issues automáticas generadas (por ejemplo, las de ICF\_JSON\_OVERRIDES) una vez confirmada la correcta importación.

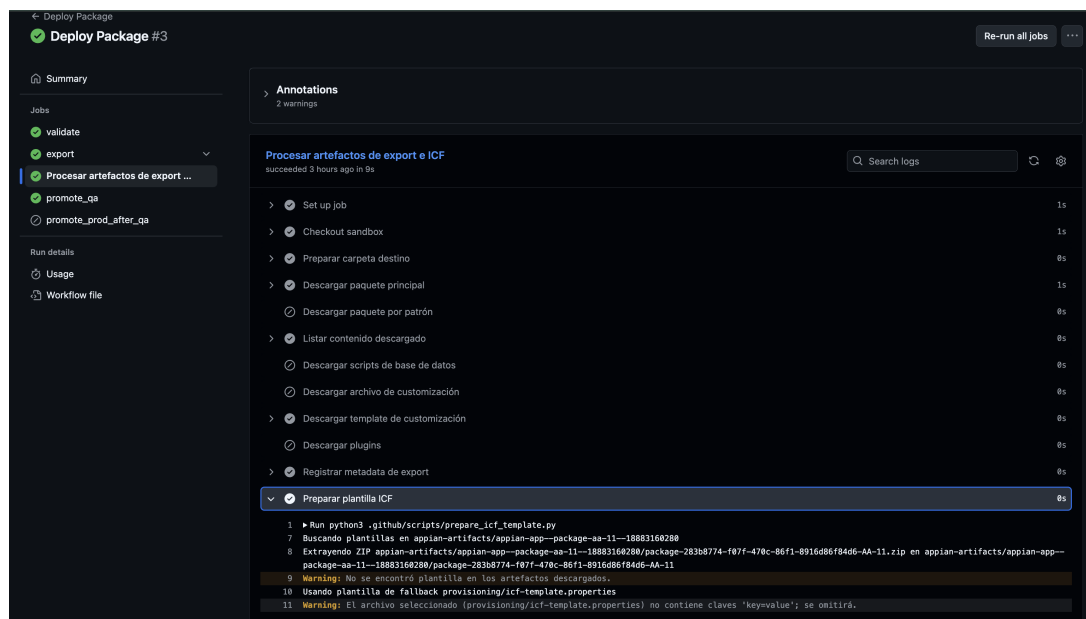


Figura 14: Visualización detallada de la ejecución del workflow y logs de cada job en GitHub Actions.

## Gestión del release en producción

Cuando una promoción hacia **prod** finaliza exitosamente, el sistema genera automáticamente un **release en GitHub**. Este release actúa como registro formal del despliegue y almacena los artefactos generados. No requiere publicación ni verificación manual, ya que se crea y asocia de forma automática al workflow ejecutado.

El único paso requerido por el **promotor** consiste en:

- Acceder a la pestaña **Releases** en el repositorio del proyecto.
- Abrir el release correspondiente y completar el campo **resumen funcional**, indicando los cambios promovidos, las historias de usuario liberadas o las incidencias resueltas en la promoción.

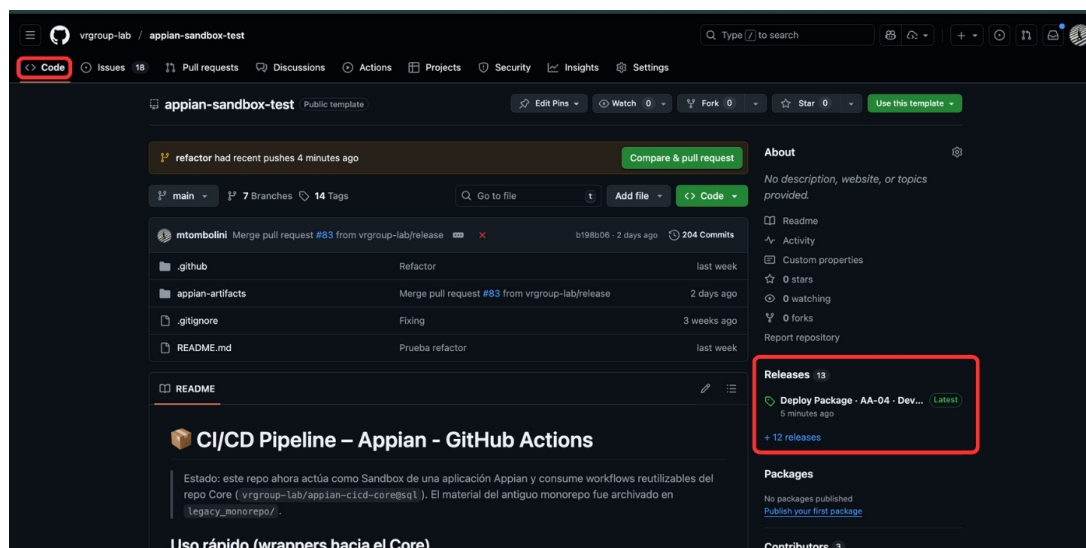


Figura 15: Ubicación de los releases dentro del repositorio en GitHub.

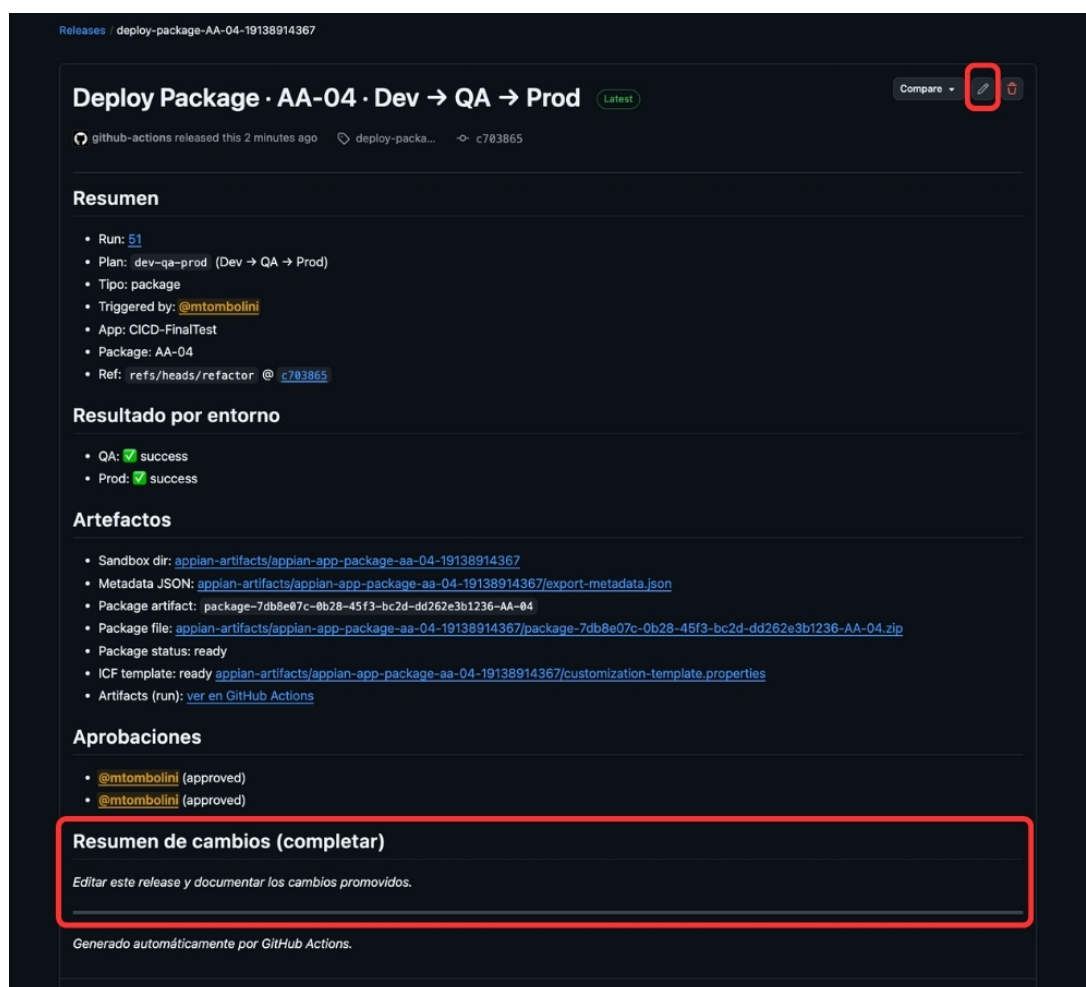


Figura 16: Detalle del release generado automáticamente en GitHub con artefactos adjuntos y sección editable de resumen funcional.

**Nota Operativa**

El cierre de issues, la verificación funcional y la actualización del release son responsabilidad del promotor. Estas acciones garantizan trazabilidad completa y documentación actualizada del ciclo de despliegue.

## 10. Arquitectura de alto nivel del sistema CI/CD

### 10.1. Visión general

El ecosistema de **CI/CD Appian-GitHub** se construye bajo una arquitectura de tipo *hub-and-spoke*. Los repositorios **wrapper** actúan como nodos específicos por aplicación Appian, mientras que el repositorio **core** centraliza la lógica reutilizable de exportación, inspección e importación de artefactos mediante la **Appian Deployment REST API**. De este modo, cada aplicación mantiene su configuración y contexto de despliegue, delegando la ejecución técnica en un conjunto común de *composite actions* del core.

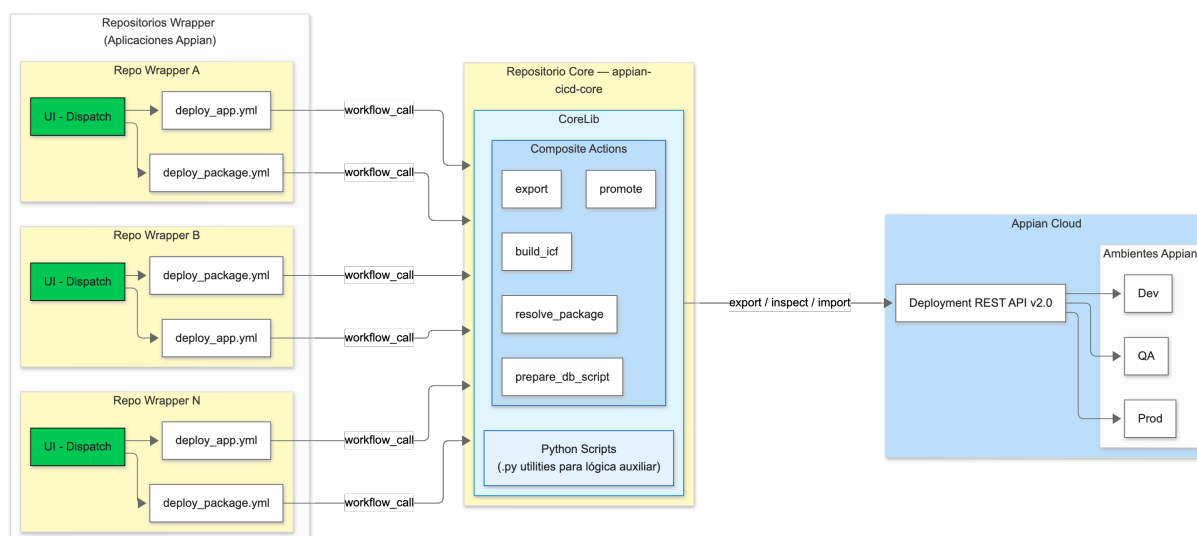


Figura 17: Arquitectura de alto nivel del sistema CI/CD Appian-GitHub.

Esta estructura permite:

- Estandarizar los despliegues entre entornos (dev, qa, prod).
- Mantener independencia entre aplicaciones sin duplicar la lógica técnica.
- Controlar accesos y aprobaciones mediante **GitHub Environments** y **Secrets**.
- Garantizar trazabilidad completa de cada promoción a través de artefactos, logs y releases.

### 10.2. Componentes principales

#### 10.2.1. Repositorios *wrapper*

Los *wrappers* mantienen una relación uno a uno con cada aplicación Appian y son los puntos de entrada del sistema. Cada repositorio contiene:

- Workflows principales (deploy\_app.yml y deploy\_package.yml), que invocan las acciones del core.

- Archivos de configuración local (por ejemplo, plantillas ICF, metadatos o scripts utilitarios).
- Directorios dedicados a artefactos descargados y logs de ejecución.

El wrapper no implementa lógica compleja: simplemente provee contexto (como `APP_UUID`, `APP_NAME` o el entorno destino) y orquesta las acciones del core.

### 10.2.2. Repositorio `appian-cicd-core`

El `appian-cicd-core` centraliza las *composite actions* y módulos Python que ejecutan las operaciones de despliegue. Sus acciones principales son:

- **`appian-export`** — exporta aplicaciones o paquetes desde el entorno origen y genera artefactos con metadatos.
- **`appian-promote`** — inspecciona e importa los artefactos al entorno destino, aplicando los overrides (ICF) o SQL cuando corresponda.
- **`appian-build-icf`** — construye archivos ICF efímeros a partir de plantillas y secretos del entorno.
- **`appian-prepare-db-scripts`** — gestiona los scripts SQL, asegurando trazabilidad de los cambios en base de datos.
- **`appian-resolve-package`** — valida identificadores de paquetes antes de iniciar exportaciones o importaciones.

Toda la lógica se encuentra bajo `.github/actions/`, mientras que la documentación técnica reside en `/docs/`.

### 10.2.3. Integración con Appian Cloud

La comunicación con Appian se realiza mediante la **Deployment REST API v2**. Cada acción del core autentica las peticiones usando claves de API configuradas como `APPIAN_env_API_KEY`, junto a URLs base definidas por entorno. De esta forma, las llamadas REST permiten realizar:

- Exportaciones desde el entorno origen.
- Inspecciones de artefactos previas a la importación.
- Importaciones o promociones controladas al entorno destino.

## 10.3. Flujos soportados y relación entre componentes

Sobre esta arquitectura se soportan tres flujos de despliegue:

- **Flujo A** — promoción estándar de aplicación sin configuraciones adicionales.
- **Flujo B** — promoción con archivos ICF y parámetros dependientes de entorno.
- **Flujo C** — promoción de paquetes con scripts SQL o *plug-ins*, gestionados a través de acciones específicas del core.

Cada wrapper selecciona el flujo apropiado según el tipo de contenido a promover, mientras que el core garantiza ejecución uniforme, parametrizable y auditable mediante las mismas acciones compartidas.

## **11. Estructura de repos y artefactos**

## **12. Seguridad y secretos**

## **13. Rollback y recuperación**

## **14. Troubleshooting / FAQ**

## **15. Control de cambios del documento**