

Buenas Prácticas de Desarrollo

Agenda

1. Componentes

- Transformer (JOLT)
- Session Management
- Object Store
- Choice & Logs
- Validator

2. Loops & Subflujos

- OnProcess & OnException
- Block Execution
- Loops
- Streams

3. Control de Error

- Validación en Llamadas Externas
- Throw Error
- OnException

4. Control de Ejecución

- Resumen de Ejecución

01 Componentes

Transformer

El componente Transformer utiliza una tecnología llamada JOLT como base.

El objetivo es manipular un JSON, siendo las transformaciones de JSON para JSON. A través del componente es configurado una especificación escrita en JSON la cual irá definir la salida del componente.



Transformer (JOLT)

Principales casos de uso:

- Transformaciones JSON -> JSON **simples y complejos**
- Ideal para manipulación de arrays.
- Ideal para manipulación y organización de las llaves de atributos en un JSON.

Como Funciona?

Dado un JSON de entrada y su especificación, obtenemos el siguiente resultado en el cual la llave del atributo fue alterada:

INPUT:

```
{
  "name": "Rodrigo"
}
```



Transformer (JOLT)

Step Name

Transformer (JOLT)

Type Properties

```
1  [
2  {
3  {
4    "operation": "shift",
5    "spec": {
6      "name": "fullName"
7    }
8  }
9  }
10 }
11 ]
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
```

OUTPUT:

```
{
  "fullName": "Rodrigo"
}
```

Tratando itens dentro de objetos

Dado un JSON de entrada y su especificación, obtenemos el siguiente resultado en el cual la llave del atributo fue alterada:

INPUT:

```
{
  "person": {
    "name": "Rodrigo"
  }
}
```

 Transformer (JOLT)

Step Name
Transformer (JOLT)

Type Properties

```
1  [
2    {
3      "operation": "shift",
4      "spec": {
5        "person": {
6          "name": "person.fullName"
7        }
8      }
9    }
10   ]
11
12
13
14
15
16
17
18
19
20
21
22
23
```

OUTPUT:

```
{
  "person": {
    "fullName": "Rodrigo"
  }
}
```


Operaciones

En la especificación jolt existen 2 parámetros, ***operation***, en el cual informamos la operación que iremos ejecutar y ***spec***, donde especificamos lo que queremos alterar/manipular del JSON. Existen 7 tipos de operaciones, siendo ellas:

SHIFT	Modificación de estructura
DEFAULT	Definición de valores patrón/fijos
REMOVE	Remoción de elementos de la estructura
sort	Ordenación alfabética (para debug e mejor lectura)
cardinality	“corrige” la cardinalidad de elementos (lista para elementos únicos)
modify-default-beta	Operaciones que no van a sobrescribir datos
modify-overwrite-beta	Operaciones que van a sobrescribir datos

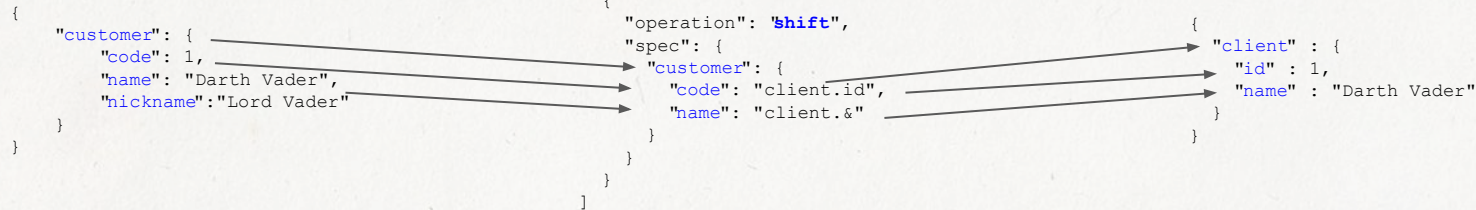
Transformer Ejemplos 1/2

INPUT

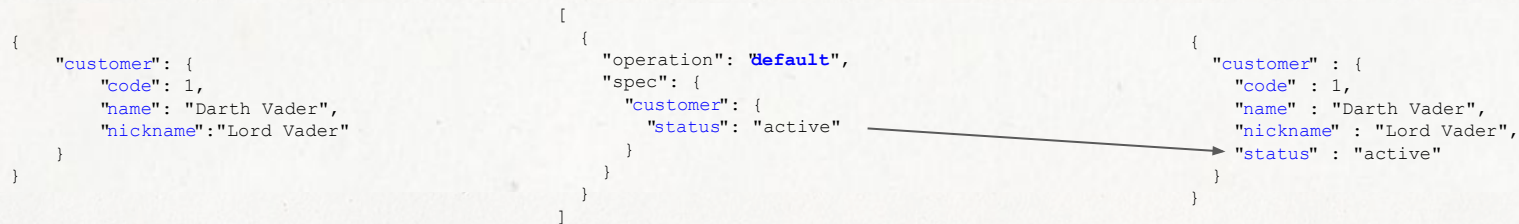
SPEC

OUTPUT

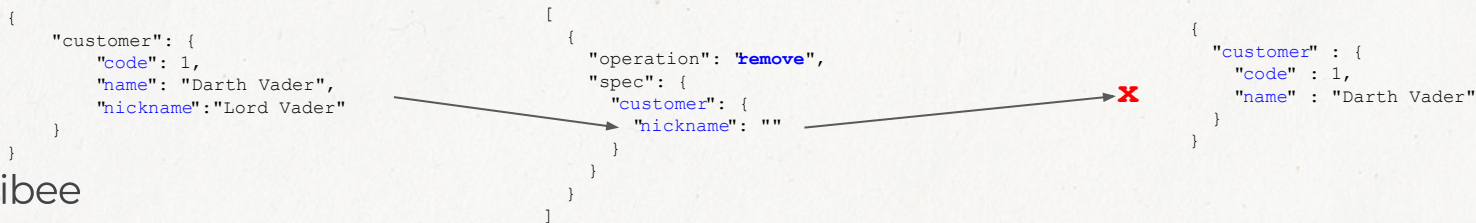
SHIFT



DEFAULT



REMOVE



Transformer Ejemplos 2/2

CARDINALITY

```
{
  "object": {
    "attr": 1
  },
  "list": [
    {
      "attr": "A"
    }
  ]
}
```

```
[
  {
    "operation": "cardinality",
    "spec": {
      "list": "ONE",
      "object": "MANY"
    }
  }
]
```

```
{
  "list": {
    "attr": "A"
  },
  "object": [ {
    "attr": 1
  } ]
}
```

MODIFY-DEFAULT

```
{
  "test": 1,
  "scores": [
    4,
    2,
    8,
    7,
    5
  ]
}
```

```
[
  {
    "operation": "modify-default-beta",
    "spec": {
      "test": 2,
      "numScores": "=size(@1,scores)",
      "firstScore": "=firstElement(@1,scores)",
      "lastScore": "=lastElement(@1,scores)",
      "sortedScores": "=sort(@1,scores)"
    }
  }
]
```

```
{
  "test": 1,
  "scores": [ 4, 2, 8, 7, 5 ],
  "numScores": 5,
  "firstScore": 4,
  "lastScore": 5,
  "sortedScores": [ 2, 4, 5, 7, 8 ]
}
```

MODIFY
OVERWRITE

```
{
  "test": 1,
  "scores": [
    4,
    2,
    8,
    7,
    5
  ]
}
```

```
[
  {
    "operation": "modify-overwrite-beta",
    "spec": {
      "test": 2,
      "numScores": "=size(@1,scores)",
      "firstScore": "=firstElement(@1,scores)",
      "lastScore": "=lastElement(@1,scores)",
      "sortedScores": "=sort(@1,scores)"
    }
  }
]
```

```
{
  "test": 2,
  "scores": [ 4, 2, 8, 7, 5 ],
  "numScores": 5,
  "firstScore": 4,
  "lastScore": 5,
  "sortedScores": [ 2, 4, 5, 7, 8 ]
}
```

Session Management

Session Management guarda atributos de un JSON en la memoria de ejecución de un pipeline. Actúa como equivalente a guardar variables en programación tradicional.

Principales caso de uso:

- Guardar atributos de un JSON que serán utilizados en tiempo de ejecución en la memoria de ejecución del pipeline.
- Es utilizado en autenticación JWT, con variables de sesión de escopo global.

IMPORTANTE:

- Los atributos guardados con session management son salvos en **memoria de ejecución** del pipeline.
 - ◆ Dependiendo el tamaño de aquello a ser guardado, considerar utilizar un banco o un Object Store.
 - ◆ Para casos en que un atributo sea guardado mas ya no más utilizado durante la ejecución del pipeline, es recomendado efectuar la limpieza de dicho valor de la sesión.

Object Store

El Object Store encapsula el acceso a un banco de datos no relacional (NoSQL), que está disponible en la plataforma para uso de los desarrolladores.

Principales casos de uso:

- Paginación de Datos
- Control de status/estado de procesamientos asíncronos
- Almacenamiento de payloads para permitir el reprocesamiento de un pipeline.
- Agrupamiento de registros para resumen de ejecuciones.

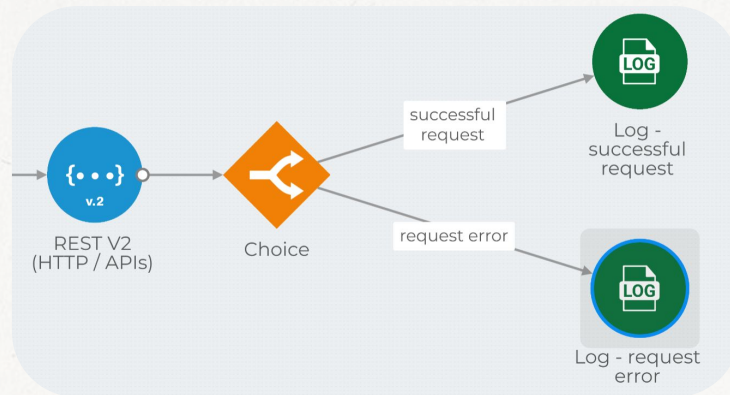
IMPORTANTE:

- Los datos guardados en object store son compartidos entre todos los pipelines.
- Atención al almacenar datos temporales, porque los mismos no serán limpiados.
- Datos almacenados en object store no ocupan memoria de ejecución.
- Es una buena práctica tener una rutina de limpieza de object store.

Choice & Logs

El componente **choice** es un componente condicional en la creación de pipelines con Digibee. Es un equivalente a *switch* de la programación tradicional.

El componente **Log** es utilizado para presentar mensajes de log con informaciones relevantes de la ejecución del pipeline.



IMPORTANTE:

- Siempre utilizar el componente Log luego de un Choice
- Utilizar misma descripción de la condición en el componente log.
- Mantener logs descriptivos para que sean fáciles de encontrar.

Validator

El componente Validator es utilizado para validar la estructura de un JSON recibido como input. Retorna error caso la estructura no sea la esperada.

Principal caso de uso:

- Validar la estructura JSON presente en el payload del pipeline.

IMPORTANTE:

- Validator utiliza tecnología JSON Schema
- Es comúnmente utilizado dentro de un **block execution** para facilitar el tratamiento en caso de errores de validación a través del `OnException` del propio componente.

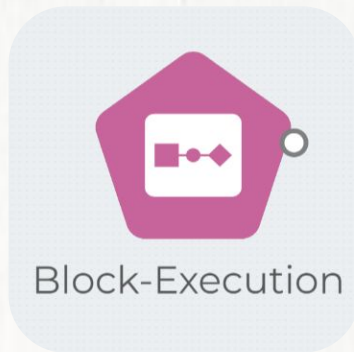
02 Loops & Subflujos

Loops & Subflujos

Algunos componentes de la plataforma Digibee nos permiten crear loops y subflujos de procesamiento conforme sea necesario. Dichos componentes crean “sub-pipelines” para el procesamiento, siendo alguno de ellos, los siguientes:



Lazo de Repetición
(for each, do while...)



Bloque de Ejecución
(block execution,
retry...)



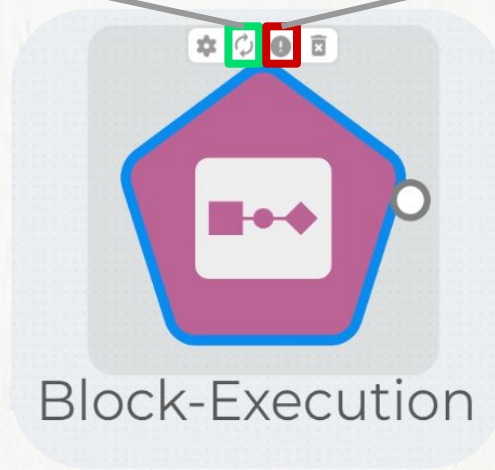
Streaming Loops
(Stream DB, Stream
Excel...)

OnProcess & OnException

Para cada componente de subflujo en Digibee esten 2 opciones adicionales de configuración adicional a las ya mencionadas con anterior, siendo ellas `onProcess` y `onException`.

OnProcess

- Irá abrir un sub-pipeline para construcción.
- En caso de *loop* de procesamiento sobre un array, dicho sub-pipeline irá a ser procesado para cada ítem del array.



OnException

- Irá abrir un sub-pipeline para construcción.
- Dicho sub-pipeline es utilizado para tratamiento de errores y será llamado caso ocurra un error durante el sub-pipeline creado en el *onProcess*.

Block Execution

El componente Block Execution permite al desarrollador agrupar de forma lógica trechos de un pipeline.

Principales casos de uso:

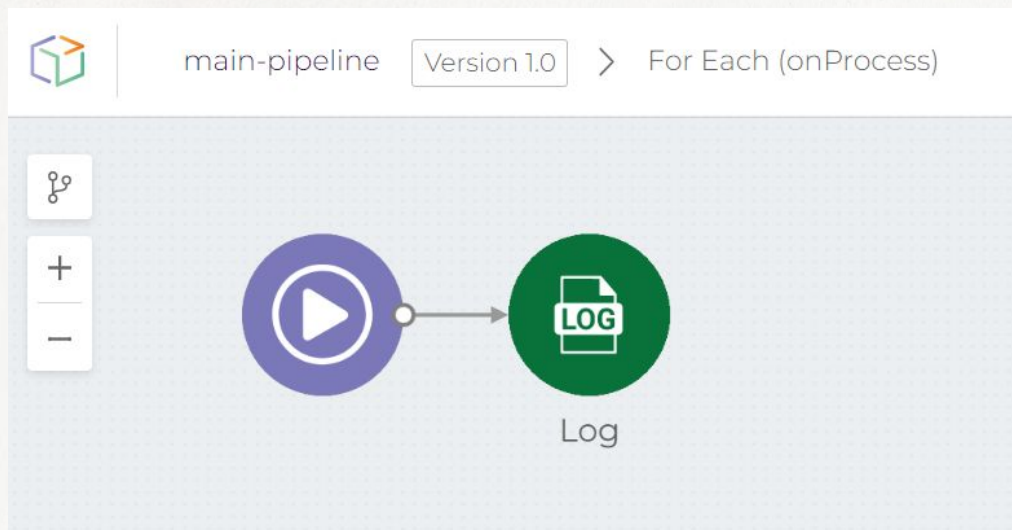
- Separar y agrupar trechos de un pipeline para que el proceso sea más simple de entender.
- Permite que diferentes caminos de procesamiento creados por los pipelines se unan en un único procesamiento evitando redundancias.

IMPORTANTE:

- Verificaciones con el componente choice pueden dividir un pipeline.
 - ◆ Luego de los procesamientos necesarios a partir del choice, los 2 caminos creados **pueden** comenzar a realizar un **mismo** procesamiento generando una **redundancia** en el pipeline. En esos casos el componente Block Execution puede ser utilizado para unir el flujo y evitar que ese problema suceda.

For Each

El componente For Each recibe como parámetro un array. Cada item de dicho array ira pasar por un subflujo creado dentro del onProcess.

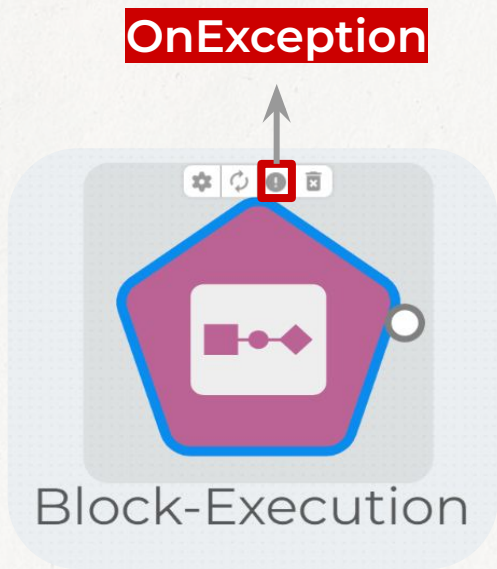


OnProcess

En la barra de navegación superior es presentado un mapa, informando la posición del loop. En el ejemplo, tenemos un loop dentro del pipeline principal.

OnException

Componentes que permiten la utilización de subflujos cuentan con la posibilidad de configurar un onException para el subflujo.




IMPORTANTE:

- Es una buena práctica utilizar al menos un componente log en el flujo de onException, mismo que éste no sea utilizado para tratar errores.
 - Lo anterior irá tornar la rastreabilidad de error más fácil en los casos que ocurra algún error inesperado durante el proceso.

Stream DB

El componente *Stream DB* recibe como parámetro una consulta a un banco de datos, en el que cada registro retornado de la consulta por el subflujo es creado dentro del *onProcess*.

Componentes de stream funciona como “buffers” de datos, generando procesamiento conforme los datos son recibidos.

 Stream DB V3

Step Name
Stream DB V3

Account
Sets the account to be used by the connector
mysql-2

Database URL
jdbc:mysql://35.223.175.97/db-training

SQL Statement
Accepts any SQL statement that the underlying database supports. Use `?field_name` to replace with value coming from property `parameters.field_name`.

1	select * from clientes LIMIT 2

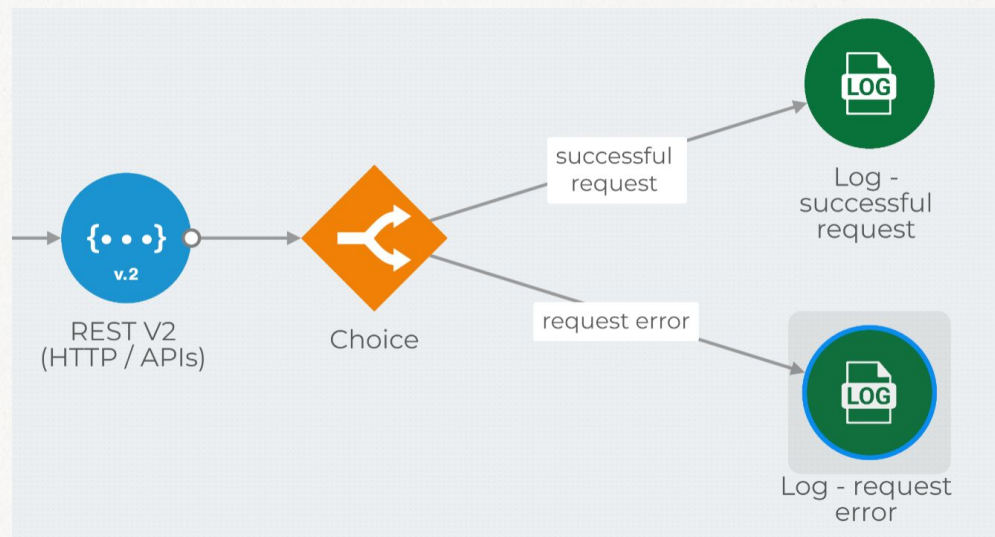
03 Control de Errores

Validación en Llamadas Externas

Es una buena práctica verificar la ocurrencia de errores en cualquier componente que realice algún tipo de llamada **externa a la plataforma**, por ejemplo, REST, SOAP, DB, FTP e etc...

IMPORTANTE:

- Cualquier error de componente que ocurra en la plataforma, una llave "error" será retornada en el payload.
- Verificar la existencia de esa llave luego de llamadas externas es una buena práctica de verificación caso sucedan errores.




Throw Error

El componente Throw Error es utilizado para emitir errores personalizados dentro de un pipeline o sub-pipeline.

IMPORTANTE:


- Es posible configurar el mensaje de error.
- Es posible configurar el status code caso sea necesario.
- Es posible crear un error personalizado en el cual se puede incorporar un JSON personalizado para el mismo.

 Throw Error

Step Name

Error Message
An error message that will be displayed with the error

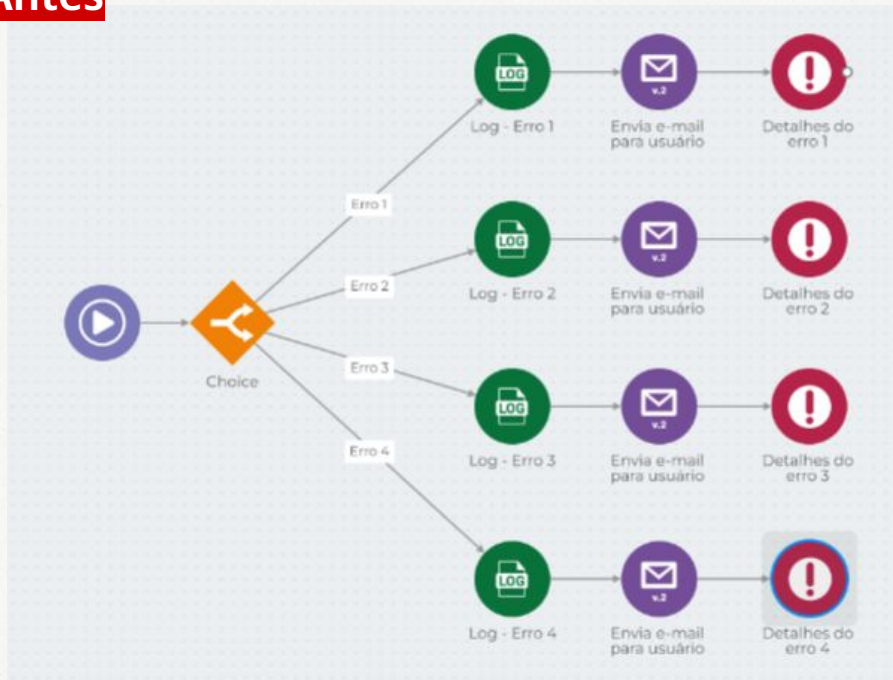
HTTP Status Code

☐  Enable Custom Error

OnProcess vs OnException

También es posible utilizar el OnException de componentes para centralizar el tratamiento de errores evitando redundancias.

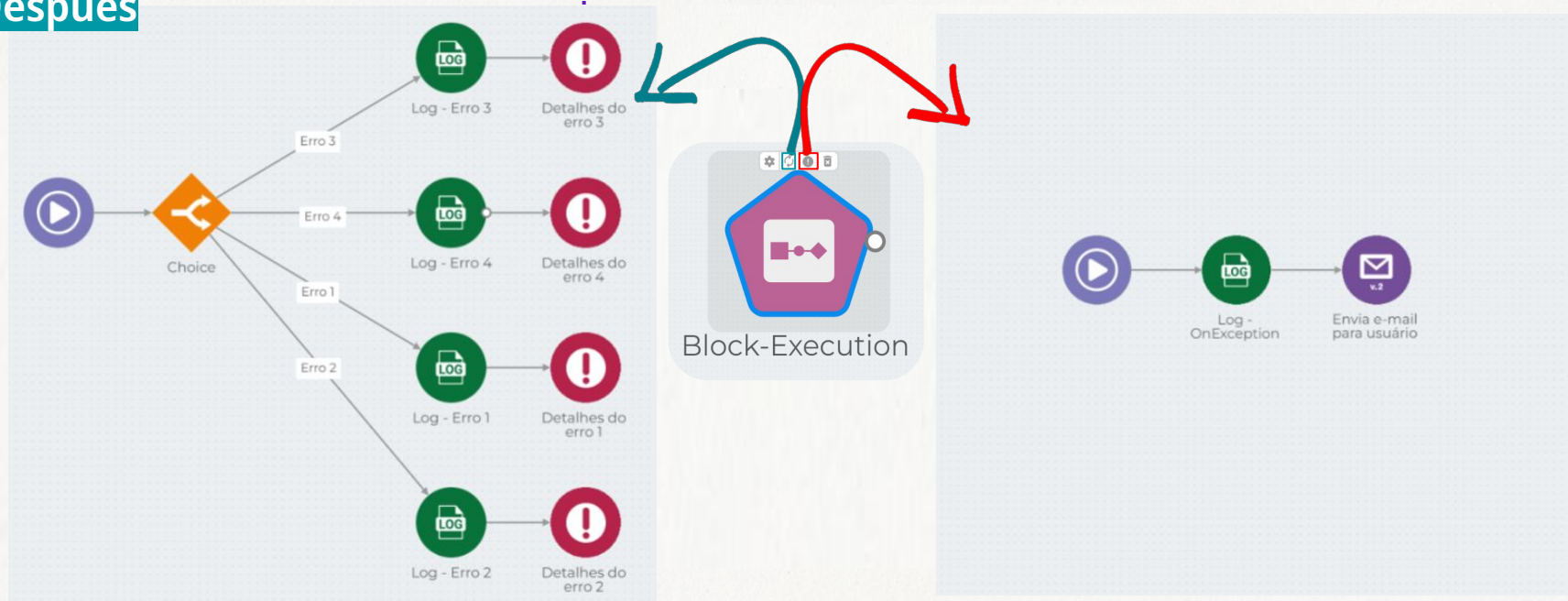
Antes



OnProcess

OnException

Después



04 Resumen de Ejecución

Resumen de Ejecución

Los pipelines de Digibee pueden ser ejecutados de forma asíncrona, procesando una enorme cantidad de datos dependiendo de la situación.

Para esos casos en específico es de **suma importancia** generar un resumen de ejecución al final del pipeline.

Dichos resúmenes van a facilitar la depuración del pipeline en caso de errores cuando el mismo se encuentre en ambiente de producción.

Ejemplo de Resumen:

```
{
  "processed": 897,
  "success": 895,
  "failed": 2,
  "failed_details": [
    {
      "message": "error while updating",
      "error": "exception.123.xyz"
    },
    {
      "message": "REST Request Error",
      "error": "exception.123.xyz"
    }
  ]
}
```


Por qué?

Al publicar un pipeline en producción, tenemos acceso a los logs de todas las ejecuciones del mismo y para cada ejecución podemos verificar los logs, el **input** y el **output** del pipeline.

Execution Details

Pipeline Key: a82ec0d2-5923-459c-ab2d-a63a8c148d5c

Request message
Request timestamp: 27/01/2022 12:15:00:0090 Request size: 2

```
{}
```

Response message
Response timestamp: 27/01/2022 12:15:00:1300 Response size: 107

```
{
  "publishEventsResult": {
    "total": 0,
    "success": 0,
    "failed": 0
  },
  "emailResult": {
    "total": 0,
    "success": 0,
    "failed": 0
  }
}
```

Time for the logs history to expire: **3 days**

Cómo hacer?

Para generar un resumen de ejecución podemos utilizar componentes tales como **Object Store** y **Session Management**.

A través de ellos conseguimos almacenar un JSON con informaciones sobre éxito o error de un debido procesamiento mientras que el mismo está sucediendo.



Object Store



Session Management

Demo Case

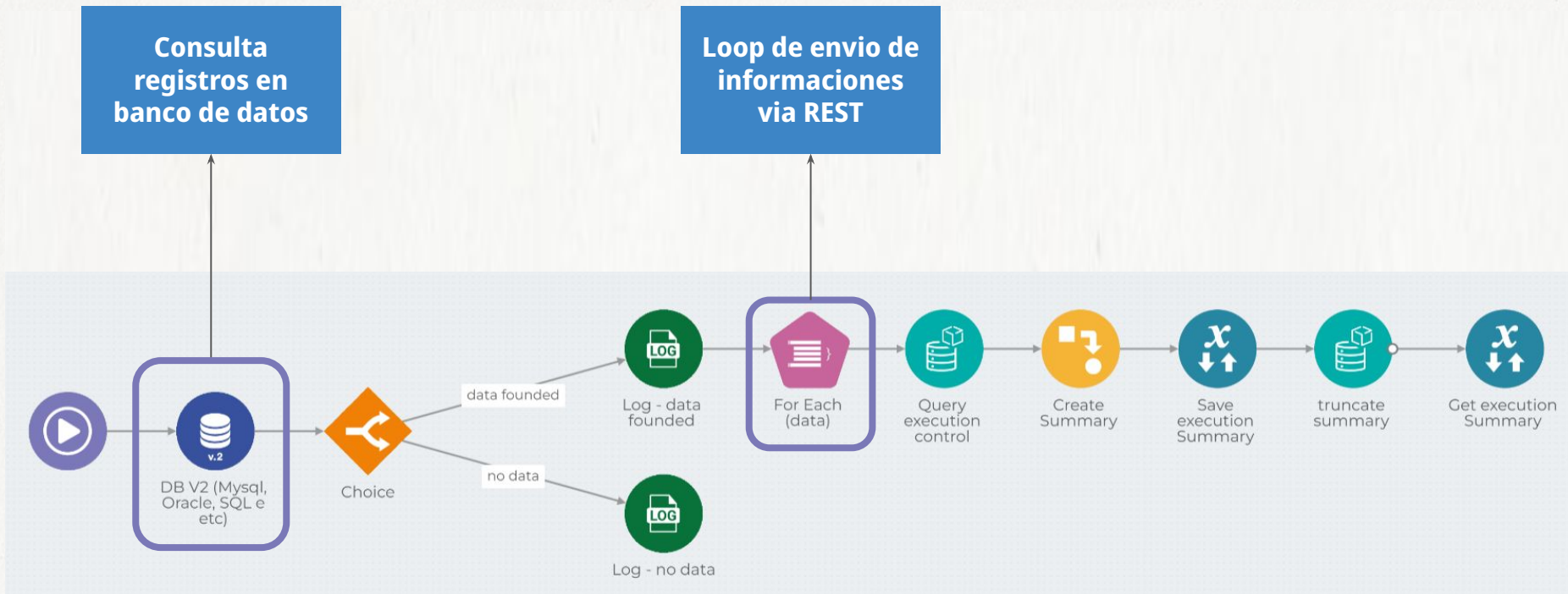
Imagina que precisamos crear un pipeline que hará una migración diaria entre un **banco de datos** y una **API REST**

- Para cada registro del banco de datos precisamos efectuar una requisición POST pasando las informaciones del registro.
- Durante la ejecución, varias llamadas REST serán realizadas y caso alguna de ellas diera error precisamos saber cuál fue la que retorno la ocurrencia y porqué. El pipeline debe generar un retorno especificado como el siguiente:

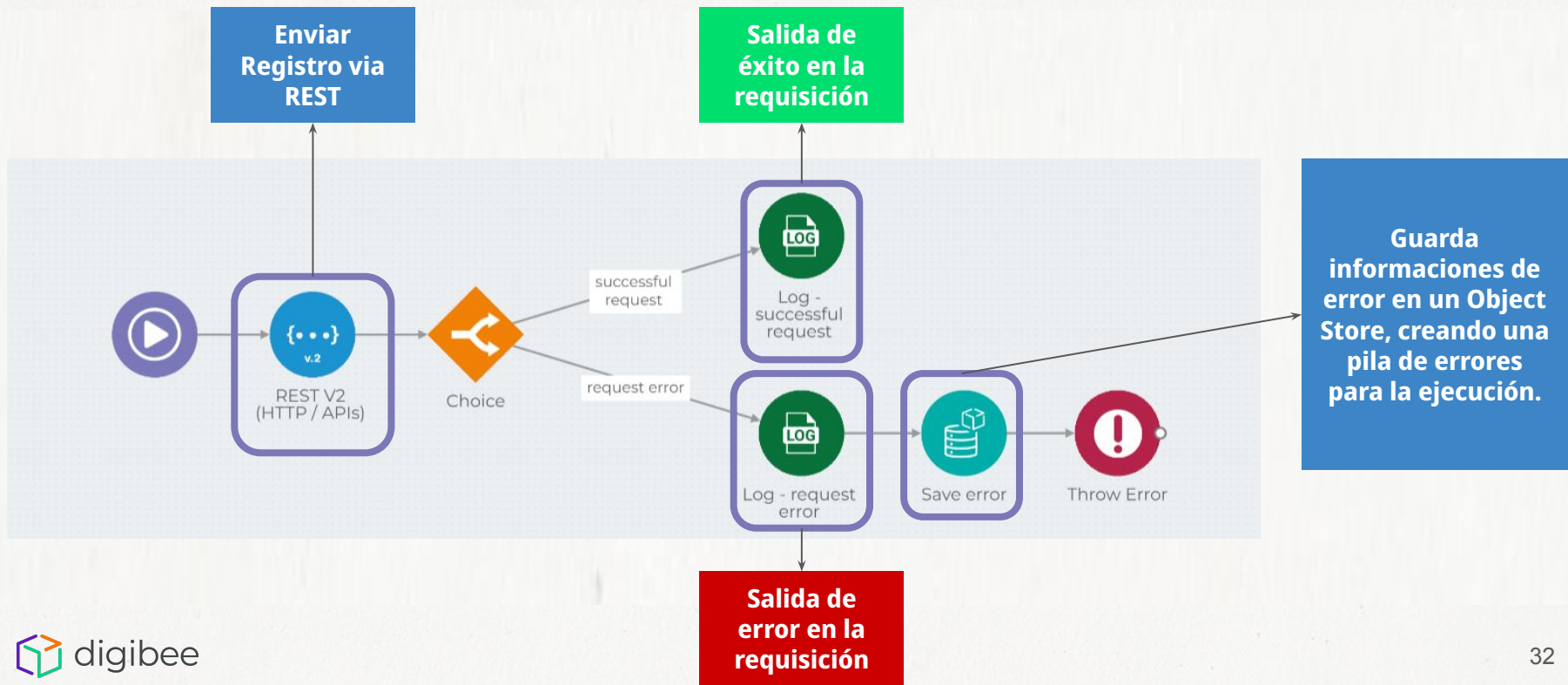
Mensaje de retorno de pipeline:

```
{
  "failed": 2,
  "failed_details": [
    {
      "message": "error while updating",
      "error": "exception.123.xyz"
    },
    {
      "message": "REST Request Error",
      "error": "exception.123.xyz"
    }
  ]
}
```

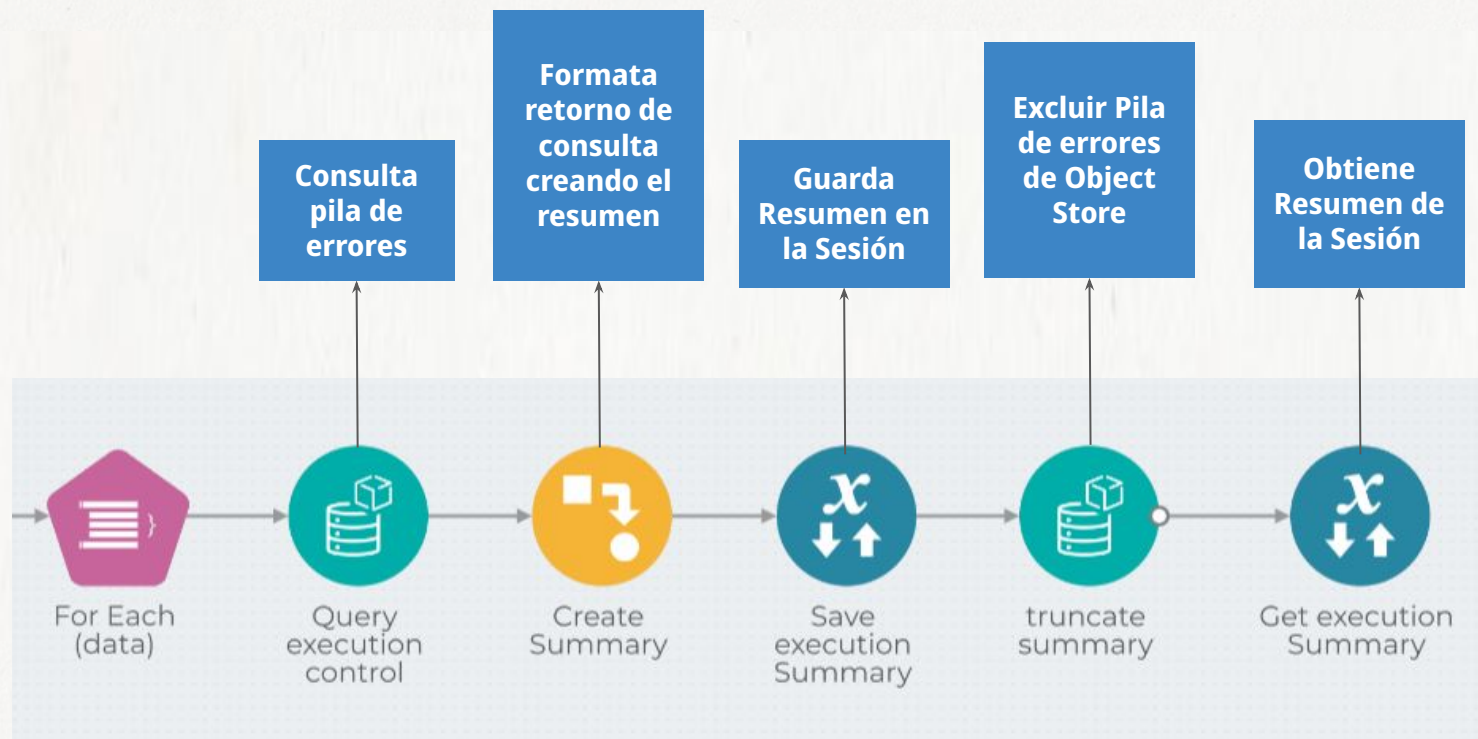
Demo Case - Proceso Principal



Demo Case - Subflujo (For Each)



Demo Case - Creando el Resumen



Demo Case - Logs - Ejecuciones Concluidas

Execution Details

Pipeline Key: 359f0ac0-37ba-4b0a-83b5-f393dfbdbfec

Request message

Request timestamp: 27/01/2022 14:51:45:0060 Request size: 2

```
{}
```



Response message

Response timestamp: 27/01/2022 14:51:45:0210 Response size: 107

```
{
  "failed": 2,
  "failed_details": [
    {
      "message": "REST Request Error on Register 2",
      "error": "exception.123.xyz"
    },
    {
      "message": "REST Request Error on Register 5",
      "error": "exception.123.xyz"
    }
  ]
}
```



Time for the logs history to expire: **3 days**

Transformer Material de Apoyo - JOLT

1. [Conociendo JOLT](#)
2. [Transformaciones con JOLT](#)
3. [Transformer \(JOLT\)](#)
4. [IF-ELSE con JOLT](#)

[Documentación Digibee](#)

Estudie los artículos siguiendo el orden propuesto.

Dudas?

Thanks



Enterprise Integration. Redesigned.