# Introduction

**Conductor** es un servidor de **workflow** basado en [Workflow Core](https://github.com/danielgerlag/workflow-core) que le permite coordinar múltiples servicios y scripts dentro de los **workflows** para que pueda crear rápidamente complejas aplicaciones de workflow. Los flujos de trabajo se componen de una serie de pasos (**Step’s**), con un objeto de datos interno compartido entre ellos para pasar información. Conductor ejecuta y rastrea automáticamente cada **step**, y reintenta cuando hay errores.

Los **Workflows** se escriben en **JSON** o **YAML** y luego se agregan al registro interno de **Conductor** a través de la API de definición. A continuación, utilice la API de flujo de trabajo para invocarlos con o sin datos personalizados.

# Installation

Conductor está disponible como una **Docker** **image** - **danielgerlag/conductor**

Conductor utiliza **MongoDB** como **almacén de datos (datastore)**, por lo que también necesitaremos una instancia de **MongoDB** para ejecutar **Conductor**

Utilice este comando para iniciar un contenedor (con la API disponible en el puerto **5001**) que apunte a:

mongodb://my-mongo-server:27017/ ya que es almacén de datos.

$ docker run -p 127.0.0.1:5001:80/tcp --env dbhost=mongodb://my-mongo-server:27017/ danielgerlag/conductor

Si desea ejecutar una **flota** (muchos) de **nodos** **Conductor**, también necesita tener una instancia **de Redis**, que se usarán como **backplane** (placa madre). No es obligatorio si solo está ejecutando una instancia. De lo contrario simplemente haga que todas sus instancias de conductor apunten a la misma instancia de **MongoDB** y **Redis**, y funcionarán como un “**Load** **Balanced**” de la **flota**.

## Environment Variables to configure

You can configure the database and Redis backplane by setting environment variables.

**dbhost**: <<insert connection string to your MongoDB server>>  
**redis**: <<insert connection string to your Redis server>> (optional)

Si deseamos configurar un contenedor de **conductor** (API en el puerto **5001**) y un contenedor MongoDB al mismo tiempo y vincularlos, use este archivo de **docker compose**:

version: '3'  
services:  
 conductor:  
 image: danielgerlag/conductor  
 ports:  
 - "5001:80"  
 links:  
 - mongo  
 environment:  
 dbhost: mongodb://mongo:27017/  
 mongo:  
 image: mongo

# Getting started

## Creating your first workflow

Comenzaremos definiendo un workflow o simple que registrará "Hola mundo" como su primer **step** y luego "Adiós!!!" como su segundo y último **step**. **POST** la definición a **api/definition** en **YAML** o **JSON**.

POST /api/definition  
Content-Type: application/yaml

Id: Hello1  
Steps:  
- Id: Step1  
 StepType: EmitLog  
 NextStepId: Step2  
 Inputs:  
 Message: '"Hello world"'  
 Level: '"Information"'  
- Id: Step2  
 StepType: EmitLog  
 Inputs:  
 Message: '"Goodbye!!!"'  
 Level: '"Information"'

or

POST /api/definition  
Content-Type: application/json

{  
 "Id": "Hello1",  
 "Steps": [  
 {  
 "Id": "Step1",  
 "StepType": "EmitLog",  
 "NextStepId": "Step2",  
 "Inputs": {  
 "Message": "\"Hello world\"",  
 "Level": "\"Information\""  
 }  
 },  
 {  
 "Id": "Step2",  
 "StepType": "EmitLog",  
 "Inputs": {  
 "Message": "\"Goodbye!!!\"",  
 "Level": "\"Information\""  
 }  
 }  
 ]  
}

Comprobamos invocando una nueva instancia de nuestro workflow. Emitimos un **POST** **/api/workflow/Hello1**

POST /api/workflow/Hello1  
Content-Type: application/json

{}

También podemos reescribir nuestro **workflow** para proporcionar datos personalizados a cualquier entrada en cualquiera de sus pasos (steps)

Id: Hello2  
Steps:  
- Id: Step1  
 StepType: EmitLog  
 Inputs:  
 Message: data.CustomMessage  
 Level: '"Information"'

Now, when we start a new instance of the workflow, we also initialize it with some data.

Ahora, cuando iniciamos una nueva instancia del **workflow**, también la inicializamos con algunos datos.

POST /api/workflow/Hello2  
Content-Type: application/json

{  
 "CustomMessage": "foobar"  
}

o

POST /api/workflow/Hello2  
Content-Type: application/x-yaml

CustomMessage: foobar

# Resources

* Download the [Postman Collection](https://raw.githubusercontent.com/danielgerlag/conductor/master/docs/Conductor.postman_collection.json)

# Input / Output Expressions

Al establecer valores de entrada y salida en los pasos, puede especificar valores constantes o una expresión.

Inputs:  
 Message: '"Hello world"'

Todas las expresiones en línea están escritas en **Python**.

Inputs:  
 Message: '"Your value is " + str(data.MyValue)'

Hay varias variables que están disponibles para su expresión

* **data**
* Esto le da acceso al objeto de datos interno específico de su flujo de trabajo. Ejemplo si pasó un objeto que tenía un campo **MyValue** al invocar su flujo de trabajo, podría acceder a él de esta manera
* data.MyValue
* **context**
* This gives you access to some contextual information about your workflow. eg. Get the wotkflow id.
* Esto le da acceso a información contextual sobre su **workflow**. Ejemplo: Obtén el **id** de **wotkflow**.
* context.Workflow.Id
* Ejemplo: Obtener el elemento de contexto actual dentro de un bucle **for**
* context.Item
* **environment**
* Este nos da acceso a las variables de entorno del sistema.
* environment["VARIABLE\_NAME"]

También puede pasar un gráfico de objetos a una entrada determinada.

You can also pass in an object graph to a given input.

Inputs:  
 BaseUrl: '"http://demo7149346.mockable.io/"'  
 Resource: '"pong"'  
 Method: '"POST"'  
 Body:  
 Value1: 1  
 Value2: 2

If you would like to have some of the properties on the object graph evaluated as an expression, simply prepend and **@** and pass an expression string.

Si desea que algunas de las propiedades del gráfico de objetos se evalúen como una expresión, simplemente anteponga y **@** y pase una cadena de expresión.

Inputs:  
 BaseUrl: '"http://demo7149346.mockable.io/"'  
 Resource: '"pong"'  
 Method: '"POST"'  
 Body:  
 Value1: 1  
 '@Value2': data.Value2

## Outputs

When mapping outputs from a **step**, the Key represents a field on your data object that is global to your workflow instance, and the expression allows you to map result values produced by the step after it has run. eg. This sets **ResponseCode** and **ResponseBody** on your workflow wide data object to the corresponding values return by the **step**.

Al asignar resultados de un **paso**, la clave representa un campo en el objeto de datos que es global para la instancia de flujo de trabajo, y la expresión le permite asignar los valores de resultado producidos por el paso después de que se haya ejecutado. Esto establece **ResponseCode** y **ResponseBody** en el objeto de datos de todo el flujo de trabajo en los valores correspondientes devueltos por paso.

Outputs:  
 ResponseCode: step.ResponseCode  
 ResponseBody: step.ResponseBody

## Branching

You can define multiple independent branches within your workflow and select one based on an expression value. Hook up your branches via the SelectNextStep property, instead of a NextStepId. The expressions will be matched to the step Ids listed in SelectNextStep, and the matching next step(s) will be scheduled to execute next. If more then one step is matched, then the workflow will have multiple parallel paths.

Puede definir varias ramas independientes dentro de su flujo de trabajo y seleccionar una en función de un valor de expresión. Conecte sus sucursales a través de la propiedad SelectNextStep , en lugar de un NextStepId. Las expresiones coincidirán con los identificadores de paso enumerados en SelectNextStep, y los siguientes pasos coincidentes se programarán para ejecutarse a continuación. Si coincide con más de un paso, el flujo de trabajo tendrá varias rutas paralelas.

{  
 "Id": "decide-workflow",  
 "Version": 1,  
 "Steps": [  
 {  
 "Id": "Start",  
 "StepType": "Decide",  
 "SelectNextStep": {  
 "A": "data.Value1 == 2",  
 "B": "data.Value1 == 3"  
 }  
 },   
 {  
 "Id": "A",  
 "StepType": "EmitLog",  
 "Inputs": {  
 "Message": "\"Hi from A!\""  
 }  
 },  
 {  
 "Id": "B",  
 "StepType": "EmitLog",  
 "Inputs": {  
 "Message": "\"Hi from B!\""  
 }  
 }  
 ]  
}

Id: decide-workflow  
Version: 1  
Steps:  
- Id: Start  
 StepType: Decide  
 SelectNextStep:  
 A: data.Value1 == 2  
 B: data.Value1 == 3  
- Id: A  
 StepType: EmitLog  
 Inputs:  
 Message: '"Hi from A!"'  
- Id: B  
 StepType: EmitLog  
 Inputs:  
 Message: '"Hi from B!"'

# Error handling

TODO

# If

Use the If step type to branch on a condition expression.

Utilice el tipo de paso If para ramificar en una expresión de condición.

Id: if-test  
Steps:  
- Id: Step1  
 StepType: If  
 NextStepId: Step2  
 Inputs:  
 Condition: 'data.MyValue > 3'  
 Do:  
 - - Id: Step1.1  
 StepType: EmitLog  
 NextStepId: Step1.2  
 Inputs:  
 Message: '"You value is greater than 3"'  
 - Id: Step1.2  
 StepType: EmitLog  
 Inputs:  
 Message: '"Thank you"'  
- Id: Step2  
 StepType: EmitLog  
 Inputs:  
 Message: '"Your value is " + str(data.MyValue)'

# While

Use the While step type to loop on a condition expression.

Utilice el tipo de paso **While** para realizar un bucle en una expresión de condición.

Id: while-test  
Steps:  
- Id: Step1  
 StepType: While  
 Inputs:  
 Condition: 'data.Counter < 3'  
 Do:  
 - - Id: Step1.1  
 StepType: NoOp  
 Outputs:  
 Counter: 'data.Counter + 1'

# ForEach

Use the ForEach step type to iterate on a collection expression. Note: ForEach iterates in parallel.

Utilice el tipo de paso ForEach para iterar en una expresión de colección. Nota: ForEach itera en paralelo.

Id: foreach-test  
Steps:  
- Id: Step1  
 StepType: Foreach  
 Inputs:  
 Collection: '[1, 2, 3]'  
 Do:  
 - - Id: Step1.1  
 StepType: EmitLog  
 Inputs:  
 Message: 'context.Item'

# Parallel

Use the Parallel step type to branch into multiple parallel sequences.

Utilice el tipo de paso Paralelo para ramificar en varias secuencias paralelas.

Id: parallel-test  
Steps:  
- Id: Step1  
 StepType: Parallel  
 NextStepId: Step2  
 Do:  
 - - Id: Step1.1 #Branch 1  
 StepType: EmitLog  
 NextStepId: Step1.2  
 Inputs:  
 Message: '"Step 1.1"'  
 - Id: Step1.2  
 StepType: EmitLog  
 Inputs:  
 Message: '"Step 1.2"'  
 - - Id: Step2.1 #Branch 2  
 StepType: EmitLog  
 NextStepId: Step2.2  
 Inputs:  
 Message: '"Step 2.1"'  
 - Id: Step2.2  
 StepType: EmitLog  
 Inputs:  
 Message: '"Step 2.2"'  
- Id: Step2  
 StepType: EmitLog  
 Inputs:  
 Message: '"done"'

# Delay

Use the Delay step type to wait a specified time based on an expresion.

Utilice el tipo de paso Retraso para esperar un tiempo especificado en función de una expresión.

Id: delay-test  
Steps:  
- Id: Step1  
 StepType: Delay  
 NextStepId: Step2  
 Inputs:  
 Period: '...'  
- Id: Step2  
 StepType: EmitLog  
 Inputs:  
 Message: '"done"'

# Recur

Use the Recur step type to trigger a recurring background sequence of steps.

Utilice el tipo de paso Repetir para desencadenar una secuencia de pasos de fondo recurrente.

Id: recur-test  
Steps:  
- Id: Step1  
 StepType: Recur  
 Inputs:  
 Interval: '...'  
 StopCondition: '...'  
 Do:  
 - - Id: Step1.1  
 StepType: EmitLog  
 Inputs:  
 Message: '"doing recurring work..."'

# Schedule

Use the Schedule step type to schedule a future set of steps to execute after a specified interval.

Utilice el tipo de paso Schedule para programar un conjunto futuro de pasos que se ejecutarán después de un intervalo especificado.

Id: schedule-test  
Steps:  
- Id: Step1  
 StepType: Schedule  
 Inputs:  
 Interval: '...'  
 Do:  
 - - Id: Step1.1  
 StepType: EmitLog  
 Inputs:  
 Message: '"doing scheduled work..."'

# WaitFor

Use the WaitFor step type to pause your workflow and wait for an external event.

Utilice el tipo de paso WaitFor para pausar el flujo de trabajo y esperar un evento externo.

The following example will wait for an external event of name my-event with a key of 5. You can of course, choose a name and key based on data in the custom data object of your workflow instance, using a Python expression.

En el ejemplo siguiente se esperará un evento externo de name my-event con una clave de 5. Por supuesto, puede elegir un nombre y una clave basados en los datos del objeto de datos personalizado de la instancia de flujo de trabajo, utilizando una expresión de Python.

Id: waitfor-test  
Steps:  
- Id: Step1  
 StepType: WaitFor  
 NextStepId: Step2  
 Inputs:  
 EventName: '"my-event"'  
 EventKey: '5'  
 Outputs:  
 EventData: step.EventData  
- Id: Step2  
 StepType: EmitLog  
 Inputs:  
 Message: 'data.EventData'

We can then use the event API to publish an event of a given name and key so that all workflows that are listening for it will be notified and recieve any data associated with the event.

Luego podemos usar la API de eventos para publicar un evento de un nombre y clave determinados para que todos los flujos de trabajo que lo estén escuchando sean notificados y reciban cualquier dato asociado con el evento.

First, let’s start a new workflow instance.

Primero, comencemos una nueva instancia de flujo de trabajo.

POST /api/workflow/waitfor-test  
{}

Response:

{  
 "workflowId": "5d274481ec9ce50001bc9c34",  
 "data": {},  
 "definitionId": "waitfor-test",  
 "version": 2,  
 "status": "Runnable",  
 "reference": null,  
 "startTime": "2019-07-11T14:15:29.86Z",  
 "endTime": null  
}

Then, let’s publish an event with a name of my-event and a key of 5. We’ll also pass the string "test" as data on the event.

POST /api/event/my-event/5

"test"

This will notify all workflows that are waiting on this particular name/key combo and pass "test" to them. The outcome of this particular example will store the data from the event in the workflow’s own internal data object.

Esto notificará a todos los flujos de trabajo que están esperando en esta combinación particular de nombre / clave y les pasará la "prueba ". El resultado de este ejemplo en particular almacenará los datos del evento en el propio objeto de datos interno del flujo de trabajo.

So, if we query the status of the workflow, we will see:

Así pues, si consultamos el estado del flujo de trabajo, veremos:

GET /api/workflow/5d274481ec9ce50001bc9c34

Response:

{  
 "workflowId": "5d274481ec9ce50001bc9c34",  
 "data": {  
 "EventData": "test"  
 },  
 "definitionId": "waitfor-test",  
 "version": 2,  
 "status": "Complete",  
 "reference": null,  
 "startTime": "2019-07-11T14:15:29.86Z",  
 "endTime": "2019-07-11T14:15:30.185Z"  
}

TODO: effective date

# Activity

An activity is defined as an item on an external queue of work, that a workflow can wait for. Use the Activity step type to pause your workflow and wait for an external activity worker that you implement.

Una actividad se define como un elemento en una cola de trabajo externa, que un flujo de trabajo puede esperar. Utilice el tipo de paso Actividad para pausar el flujo de trabajo y esperar a un trabajador de actividad externo que implemente.

The following example will wait for an external activity of name my-activity and pass “Hello from the workflow” as the input data to the worker that will process this activity. Once the worker submits a response, the workflow will continue and log the result from the worker.

En el ejemplo siguiente se esperará una actividad externa de nombre my-activity y se pasará "Hello from the workflow" como datos de entrada al trabajador que procesará esta actividad. Una vez que el trabajador envía una respuesta, el flujo de trabajo continuará y registrará el resultado del trabajador.

Id: activity-test  
Steps:  
- Id: Step1  
 StepType: Activity  
 NextStepId: Step2  
 Inputs:  
 ActivityName: '"my-activity"'  
 Parameters: '"Hello from the workflow"'  
 Outputs:  
 ActivityResult: step.Result  
- Id: Step2  
 StepType: EmitLog  
 Inputs:  
 Message: 'data.ActivityResult'

We can then use the activity API to implement a worker to process “my-activity” activities.

First, let’s start a new workflow instance.

Luego podemos usar la API de actividad para implementar un trabajador para procesar actividades de "mi actividad".

Primero, comencemos una nueva instancia de flujo de trabajo.

POST /api/workflow/activity-test  
{}

Response:

{  
 "workflowId": "5d274481ec9ce50001bc9c34",  
 "data": {},  
 "definitionId": "activity-test",  
 "version": 1,  
 "status": "Runnable",  
 "reference": null,  
 "startTime": "2019-07-11T14:15:29.86Z",  
 "endTime": null  
}

In our external worker process, we can fetch a waiting activity of an active workflow with the following request.

En nuestro proceso de trabajador externo, podemos obtener una actividad de espera de un flujo de trabajo activo con la siguiente solicitud.

GET /api/activity/my-activity?timeout=30

This will wait up to 30 seconds for some work be become ready to process, if there is no workflow waiting on the activity requested, then a 404 Not Found will be returned. If there is work waiting for that activity queue, then an exclusive token will be issued and the reponse will look as follows

Esto esperará hasta 30 segundos para que algún trabajo esté listo para procesar, si no hay un flujo de trabajo esperando en la actividad solicitada, se devolverá un 404 No encontrado . Si hay trabajo esperando esa cola de actividad, se emitirá un token exclusivo y la respuesta se verá de la siguiente manera:

{  
 "token": "eyJTdWJzY3JpcHRpb25JZCI6IjVlMD",  
 "activityName": "my-activity",  
 "parameters": "Hello from the workflow",  
 "tokenExpiry": "9999-12-31T23:59:59.9999999"  
}

* token An exclusive token is issued to the worker to use in future requests for this activity.
* parameters The input data that the workflow attached to this actvity. Can be any object.
* tokenExpiry When the token expires and the activity will be made available to other workers.

To submit a successful response to an activity and pass some response data back to the workflow in the body of the request

* token Se emite un token exclusivo al trabajador para que lo use en futuras solicitudes de esta actividad.
* parámetros Los datos de entrada que el flujo de trabajo adjuntó a esta actvity. Puede ser cualquier objeto.
* tokenExpiry Cuando el token expira y la actividad se pondrá a disposición de otros trabajadores.

Para enviar una respuesta correcta a una actividad y devolver algunos datos de respuesta al flujo de trabajo en el cuerpo de la solicitud

POST /api/activity/success/eyJTdWJzY3JpcHRpb25JZCI6IjVlMD

"Hello from the worker"

The workflow will now continue and the “Hello from the worker” string will be mapped to the ActivityResult field on the workflow data, and then logged in the final step. The response data can be any object, not just scalar values.

El flujo de trabajo ahora continuará y la cadena "Hola del trabajador" se asignará al campo ActivityResult en los datos del flujo de trabajo y, a continuación, se registrará en el paso final. Los datos de respuesta pueden ser cualquier objeto, no solo valores escalares.

# Using custom steps in your workflow

Conductor also allows you to define your own steps that can be used within your workflows. Currently, the only supported language is Python. More languages will be implemented in the future.

The following call creates a step called add, which is a Python script that sets c to a + b

Conductor también le permite definir sus propios pasos que se pueden utilizar dentro de sus flujos de trabajo. Actualmente, el único lenguaje soportado es Python. En el futuro se implementarán más idiomas.

La siguiente llamada crea un paso llamado add, que es un script de Python que establece c en a + b

POST /api/step/add  
Content-Type: text/x-python

c = a + b

Next, we create a workflow definition that invokes our add step with values from the internal data object of the workflow and outputs the result to a log.

A continuación, creamos una definición de flujo de trabajo que invoca nuestro step de adición con valores del objeto de datos interno del flujo de trabajo y envía el resultado a un registro.

POST /api/definition  
Content-Type: application/yaml

Id: MyStepWorkflow  
Steps:  
 - Id: Step1  
 StepType: add  
 Inputs:  
 a: data.Value1  
 b: data.Value2  
 NextStepId: Step2  
 Outputs:  
 Result: step["c"]   
 - Id: Step2  
 StepType: EmitLog  
 Inputs:  
 Message: '"Answer is " + str(data.Result)'

Now, lets test it by invoking a new instance of our workflow

Ahora, probémoslo invocando una nueva instancia de nuestro flujo de trabajo

POST /api/workflow/MyStepWorkflow  
Content-Type: application/json

{  
 "Value1": 7,  
 "Value2": 3  
}

Response:

{  
 "workflowId": "5d0ab0ff23576b61e4afbcfb",  
 "data": {  
 "Value1": 7,  
 "Value2": 3  
 },  
 "definitionId": "MyStepWorkflow",  
 "version": 1,  
 "status": "Runnable",  
}

Esto debería generar lo siguiente en los logs

Answer is 10

Además, puede inspeccionar los datos internos del **workflow**

GET /api/workflow/5d0ab0ff23576b61e4afbcfb

Response:

{  
 "workflowId": "5d0ab0ff23576b61e4afbcfb",  
 "data": {  
 "Value1": 7,  
 "Value2": 3,  
 "Result": 10  
 },  
 "definitionId": "MyStepWorkflow",  
 "version": 1,  
 "status": "Complete",  
}

# Making Http requests in your workflow

Definamos un **workflow** utilizando el **step** **HttpRequest** integrado.

POST /api/definition  
Content-Type: application/json

{  
 "id": "MyHttpWorkflow",  
 "steps": [  
 {   
 "id": "Step1",  
 "stepType": "HttpRequest",  
 "inputs": {  
 "BaseUrl": "\"http://demo7149346.mockable.io/\"",  
 "Resource": "\"ping\"",  
 "Headers": {  
 "@Authorization": ""  
 }  
 },  
 "outputs": {  
 "ResponseCode": "step.ResponseCode",  
 "ResponseBody": "step.ResponseBody"  
 }  
 }  
 ]  
}

or in Yaml format:

POST /api/definition  
Content-Type: application/yaml

id: MyHttpWorkflow  
steps:  
- id: Step1  
 stepType: HttpRequest  
 inputs:  
 BaseUrl: '"http://demo7149346.mockable.io/"'  
 Resource: '"ping"'  
 Headers:  
 Content-Type: application/json  
 outputs:  
 ResponseCode: step.ResponseCode  
 ResponseBody: step.ResponseBody

Ahora, probémoslo invocando una nueva instancia de nuestro workflow con un objeto de datos vacío

POST /api/workflow/MyHttpWorkflow  
Content-Type: application/json

{}

Response:

{  
 "workflowId": "5d0ab0ff23576b61e4afbcfb",  
 "data": {},  
 "definitionId": "MyHttpWorkflow",  
 "version": 1,  
 "status": "Runnable",  
}

Now the workflow should call the **Http** endpoint and record the response code and body in it’s internal data object. Let’s inspect it with via the **API**.

Ahora el flujo de trabajo debe llamar al extremo **Http** y registrar el código de respuesta y el cuerpo en su objeto de datos interno. Vamos a inspeccionarlo a través de la **API**.

GET /api/workflow/5d0ab0ff23576b61e4afbcfb

Response:

{  
 "workflowId": "5d0ab0ff23576b61e4afbcfb",  
 "data": {  
 "ResponseCode": 200,  
 "ResponseBody": {  
 "msg": "Hello world"  
 }  
 },  
 "definitionId": "MyHttpWorkflow",  
 "version": 1,  
 "status": "Complete",  
}

An example of a POST with two values, one static and one sourced from the internal data object of the workflow.

Un ejemplo de un POST con dos valores, uno estático y otro procedente del objeto de datos interno del flujo de trabajo.

id: MyHttpWorkflow  
steps:  
- id: Step1  
 stepType: HttpRequest  
 inputs:  
 BaseUrl: '"http://demo7149346.mockable.io/"'  
 Resource: '"pong"'  
 Method: '"POST"'  
 Body:  
 Value1: 1  
 '@Value2': data.Value2  
 outputs:  
 ResponseCode: step.ResponseCode  
 ResponseBody: step.ResponseBody

* [Definitions](#definition-api)
* [Workflows](#workflow-api)
* [Events](#event-api)
* [Activity Workers](#activity-api)
* [Custom Steps](#steps-api)
* [Diagnostics](#diagnostic-api)
* [Swagger documentation](#swagger-documentation)

Download the [Postman Collection](https://raw.githubusercontent.com/danielgerlag/conductor/master/docs/Conductor.postman_collection.json)

# Definition API

## Create or update a definition

We POST the definition to api/definition in either YAML or JSON.

Emitimos un POST con la definición al api/ definición en YAML o JSON.

POST /api/definition  
Content-Type: application/yaml

Id: Hello1  
Steps:  
- Id: Step1  
 StepType: EmitLog  
 NextStepId: Step2  
 Inputs:  
 Message: '"Hello world"'  
 Level: '"Information"'  
- Id: Step2  
 StepType: EmitLog  
 Inputs:  
 Message: '"Goodbye!!!"'  
 Level: '"Information"'

Posting to a definition ID that already exists, will create a second version of that workflow definition and all existing workflows that were started on the old version, will continue on the old version but all workflows that are started after this will run on the new version.

La publicación en un **ID** de definición que ya existe, creará una segunda versión de esa definición de flujo de trabajo y todos los flujos de trabajo existentes que se iniciaron en la versión anterior, continuarán en la versión anterior, pero todos los flujos de trabajo que se inicien después de esto se ejecutarán en la nueva versión.

# Workflow API

## Start a workflow

To start a workflow, submit a POST to /api/workflow/<<DefinitionId>>, where the body of the request will be the initial data object passed to the new workflow instance.

Para iniciar un flujo de trabajo, envíe un POST a /api/workflow/<<DefinitionId>>, donde el cuerpo de la solicitud será el objeto de datos inicial pasado a la nueva instancia de flujo de trabajo.

POST /api/workflow/<<DefinitionId>>

Example: Start the HelloWorld workflow, with some custom data.

POST /api/workflow/HelloWorld  
Content-Type: application/x-yaml

CustomMessage: foobar

#### Response

{  
 "workflowId": "5d26ae05ec9ce50001bc9c2a",  
 "data": {  
 "CustomMessage": "foobar"  
 },  
 "definitionId": "HelloWorld",  
 "version": 1,  
 "status": "Runnable",  
 "reference": null,  
 "startTime": "2019-07-11T03:33:25.203Z",  
 "endTime": null  
}

## Querying a workflow

If you have the **workflowId** that you get back when you start a workflow, you can query it’s status via the API.

Si tiene el **workflowId** que recupera al iniciar un flujo de trabajo, puede consultar su estado a través de la API.

GET /api/workflow/<<WorkflowId>>

#### Response

{  
 "workflowId": "5d26ae05ec9ce50001bc9c2a",  
 "data": {  
 "CustomMessage": "foobar"  
 },  
 "definitionId": "HelloWorld",  
 "version": 1,  
 "status": "Runnable",  
 "reference": null,  
 "startTime": "2019-07-11T03:33:25.203Z",  
 "endTime": null  
}

## Suspending a workflow

Puede suspender un workflow con un PUT

PUT /api/workflow/<<WorkflowId>>/suspend

## Resuming a workflow

Puede reanudar un **workflow** suspendido con un PUT

PUT /api/workflow/<<WorkflowId>>/resume

## Terminting a workflow

Puede anular un flujo de trabajo con un DELETE

DELETE /api/workflow/<<WorkflowId>>

# Event API

You can publish an event with a particular name and key and attach some data to all workflows that may be listening to it. Use the event API.

Puede publicar un evento con un nombre y una clave determinados y adjuntar algunos datos a todos los flujos de trabajo que puedan estar escuchándolo. Utilice la API de eventos.

POST /api/event/<<name>>/<<key>>

<<data>>

# Activity API

An activity is defined as an item on an external queue of work, that a workflow can wait for.

Una actividad se define como un elemento en una cola de trabajo externa, que un flujo de trabajo puede esperar.

## Getting a pending activity

To fetch a waiting activity of an active workflow.

Para obtener una actividad en espera de un flujo de trabajo activo.

GET /api/activity/<<name>>?workerId=<<workerId>>&timeout=30

* name (**required**) is the activity name to fetch waiting work for.
* workerId (**optional**) is an identifier of the worker pulling the work.
* timout (**optional**) number of seconds to block while waiting for an activity.

#### Response

If there is no workflow waiting on the activity requested, then a 404 Not Found will be returned. If there is work waiting for that activity name, then an exclusive token will be issued and the response will look as follows

Si no hay ningún flujo de trabajo esperando en la actividad solicitada, se devolverá un 404 No encontrado. Si hay trabajo esperando ese nombre de actividad, se emitirá un token exclusivo y la respuesta se verá de la siguiente manera:

{  
 "token": "...",  
 "activityName": "...",  
 "parameters": {},  
 "tokenExpiry": "9999-12-31T23:59:59.9999999"  
}

* **token** An exclusive token is issued to the worker to use in future requests for this activity.
* **parameters** The input data that the workflow attached to this activity.
* **tokenExpiry** When the token expires, and the activity will be made available to other workers.

## Submitting a result for an activity

To submit a successful response to an activity and pass some response data back to the workflow in the body of the request.

Para enviar una respuesta correcta a una actividad y devolver algunos datos de respuesta al flujo de trabajo en el cuerpo de la solicitud.

POST /api/activity/success/<<token>>

<<data>>

To submit a failure response to an activity and pass some response data back to the workflow in the body of the request.

Para enviar una respuesta de error a una actividad y devolver algunos datos de respuesta al flujo de trabajo en el cuerpo de la solicitud.

POST /api/activity/fail/<<token>>

<<data>>

## Release a token

To release a token held by a worker, so that another worker could pick it up.

Liberar un token en poder de un trabajador, para que otro trabajador pueda recogerlo.

DELETE /api/activity/<<token>>

# Steps API

Conductor also allows you to define your own steps that can be used within your workflows. Currently, the only supported language is Python. More languages will be implemented in the future.

Conductor también le permite definir sus propios pasos que se pueden utilizar dentro de sus flujos de trabajo. Actualmente, el único lenguaje soportado es Python. En el futuro se implementarán más idiomas.

## Creating a step

La siguiente llamada crea un **step** llamado **add**, que es un **script** **de** **Python** que establece **c en a + b**

POST /api/step/add  
Content-Type: text/x-python

c = a + b

## Viewing a step

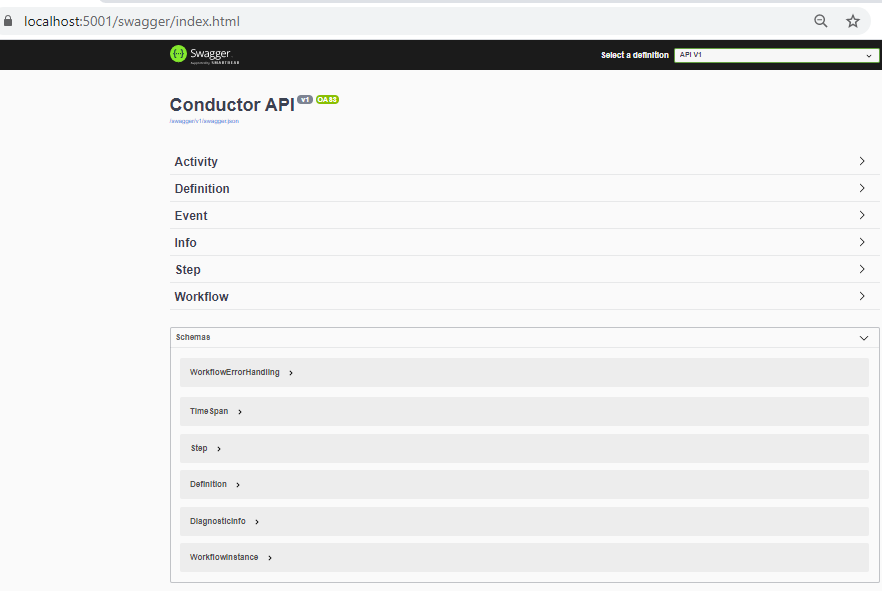
GET /api/step/<<id>>

# Diagnostic API

GET /api/info

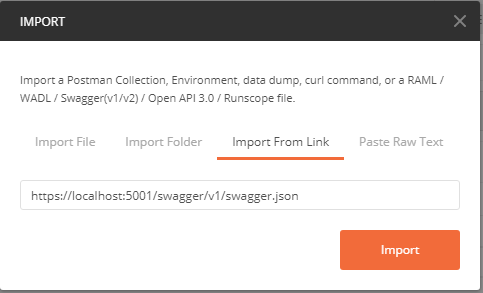
# Swagger Documentation

It is possible to discover and try out API with [Swagger](https://swagger.io/) in a browser https://localhost:5001/swagger.



**Swagger in browser**

It’s also convenient to import a Postman collection as swagger **json** https://localhost:5001/swagger/v1/swagger.json.



**Import Posrmat collection**

# Authentication

**Conductor** supports integrated authentication using the [OpenID Connect](https://openid.net/connect/) protocol.

**Conductor** admite la autenticación integrada mediante el protocolo [OpenID Connect](https://openid.net/connect/).

By default, authentication is disabled. To enable it, \* Set the auth environment variable to 'true' \* Set the alg environment variable to the signing algorithm (RS256 or ES256) \* Set the publickey variable to a Base64 encoded public key.

De forma predeterminada, la autenticación está deshabilitada. Para habilitarlo, \* Establezca la variable de entorno de autenticación en 'true' \* Establezca la variable de entorno alg en el algoritmo de firma (RS256 o ES256) \* Establezca la variable de clave pública en una clave pública codificada en Base64.

If authentication is enabled then you need to include a signed [JWT bearer token](https://jwt.io/) along with every request. The is done by adding the Authorization: Bearer <<token>> header to each request. The token should be a valid JWT token that was signed with the corresponding private key to the public one in the environment variable.

Si la autenticación está habilitada, debe incluir un [token de portador JWT](https://jwt.io/) firmado junto con cada solicitud. Esto se hace agregando el encabezado Authorization: Bearer <<token>> a cada solicitud. El token debe ser un token JWT válido que se firmó con la clave privada correspondiente a la pública en la variable de entorno.

The token must also include a scope claim that indicate the level of access. The following scopes are used within Conductor.

El token también debe incluir una notificación de ámbito que indique el nivel de acceso. Los siguientes ámbitos se utilizan dentro de Conductor.

* conductor:admin - Adminstrative tasks.
* conductor:author - Authoring of workflow definitions and steps.
* conductor:controller - Starting, stopping, suspending and resuming workflows.
* conductor:viewer - Querying the status of a workflow.
* conductor:worker - Activity workers.

A minimal **JWT** payload the include all the scopes would look as follows

Un payload **JWT** mínimo que incluya todos los alcances se vería de la siguiente manera:

{  
 "scope": "conductor:admin conductor:author conductor:controller conductor:viewer conductor:worker"  
}

Some authentication servers that support [OpenID Connect](https://openid.net/connect/) include

Algunos servidores de autenticación que [admiten OpenID Connect](https://openid.net/connect/) incluyen

* [Auth0](https://auth0.com/) - A cloud service
* [Okta](https://www.okta.com/) - A cloud service
* [Keycloak](https://github.com/keycloak/keycloak/) - Open source auth server
* [Identity Server](https://identityserver.io/) - Open source auth server
* [Dex](https://github.com/dexidp/dex) - Open source auth server