

Comience con el cliente Java

En este tutorial, aprenderá a usar el cliente Java en una aplicación Java para interactuar con Zeebe.

Se lo guiará a través de los siguientes pasos:

- [Configurar un proyecto](#)
- [Modelar un flujo de trabajo](#)
- [Implementar un flujo de trabajo](#)
- [Crear una instancia de flujo de trabajo](#)
- [Trabajar en un trabajo](#)
- [Trabajar con datos](#)

Puede encontrar el código fuente completo, incluidos los diagramas BPMN, en [GitHub](#).

Puede ver un video [tutorial](#) de esta guía en el [canal de YouTube Zeebe](#) aquí.

Prerrequisitos

- [Java 8](#)
- [Apache Maven](#)
- [Zeebe Modeler](#)

Uno de los siguientes:

(Usando Docker)

- [Estibador](#)
- [Configuraciones de Zeebe Docker](#)

(Sin usar Docker)

- [Distribución Zeebe](#)
- [Monitor Zeebe](#)

Inicie el corredor

Antes de comenzar a configurar su proyecto, inicie el corredor.

Si está utilizando Docker con `zeebe-docker-compose`, cambie al `simple-monitor` subdirectorio y ejecute `docker-compose up`.

Si no está utilizando Docker, ejecute el script de inicio `bin/broker` o `bin/broker.bat` en la distribución.

De forma predeterminada, el intermediario se une a `localhost:26500`, que se utiliza como punto de contacto en esta guía.

Configurar un proyecto

Primero, necesitamos un proyecto Maven. Cree un nuevo proyecto con su IDE o ejecute el comando Maven:

```
mvn archetype:generate \
  -DgroupId=io.zeebe \
  -DartifactId=zeebe-get-started-java-client \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DinteractiveMode=false
```



Agregue la biblioteca del cliente Zeebe como dependencia de los proyectos `pom.xml`:

```
<dependency>
  <groupId>io.zeebe</groupId>
  <artifactId>zeebe-client-java</artifactId>
  <version>${zeebe.version}</version>
</dependency>
```



Cree una clase principal y agregue las siguientes líneas para arrancar el cliente Zeebe:

```
package io.zeebe;

import io.zeebe.client.ZeebeClient;

public class App
{
    public static void main(String[] args)
    {
        final ZeebeClient client = ZeebeClient.newClientBuilder()
            // change the contact point if needed
            .brokerContactPoint("127.0.0.1:26500")
            .usePlaintext()
            .build();

        System.out.println("Connected.");

        // ...

        client.close();
        System.out.println("Closed.");
    }
}
```

Ejecuta el programa:

- Si usa un IDE, simplemente puede ejecutar la clase principal, usando su IDE.
- De lo contrario, debe compilar un archivo JAR ejecutable con Maven y ejecutarlo.

Interludio: construir un archivo JAR ejecutable

Agregue el complemento Maven Shade a su pom.xml:

```
<!-- Maven Shade Plugin -->
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>2.3</version>
  <executions>
    <!-- Run shade goal on package phase -->
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
      <configuration>
        <transformers>
          <!-- add Main-Class to manifest file -->
          <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransfor
mer">
            <mainClass>io.zeebe.App</mainClass>
          </transformer>
        </transformers>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Ahora ejecute `mvn package`, y generará un archivo JAR en el `target` subdirectorio. Puedes ejecutar esto con `java -jar target/${JAR file}`.

Salida del programa en ejecución

Deberías ver la salida:

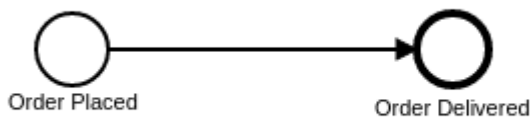
```
Connected.
```

```
Closed.
```

Modelar un flujo de trabajo

Ahora, necesitamos un primer flujo de trabajo que luego se pueda implementar. Más tarde, ampliaremos el flujo de trabajo con más funcionalidad.

Abra el Modelador Zeebe y cree un nuevo diagrama BPMN. Agregue un evento inicial y un evento final al diagrama y conecte los eventos.



Establezca la identificación (la identificación del proceso BPMN) y marque el diagrama como ejecutable.

Guarde el diagrama como `src/main/resources/order-process.bpmn` debajo de la carpeta del proyecto.

Implementar un flujo de trabajo

A continuación, queremos implementar el flujo de trabajo modelado en el intermediario.

El intermediario almacena el flujo de trabajo bajo su ID de proceso BPMN y asigna una versión.

Agregue el siguiente comando de despliegue a la clase principal:

```
package io.zeebe;

import io.zeebe.client.api.response.DeploymentEvent;

public class Application
{
    public static void main(String[] args)
    {
        // after the client is connected

        final DeploymentEvent deployment = client.newDeployCommand()
            .addResourceFromClasspath("order-process.bpmn")
            .send()
            .join();

        final int version = deployment.getWorkflows().get(0).getVersion();
        System.out.println("Workflow deployed. Version: " + version);

        // ...
    }
}
```

Ejecute el programa y verifique que el flujo de trabajo se implemente correctamente. Deberías ver la salida:

```
Workflow deployed. Version: 1
```

Crear una instancia de flujo de trabajo

Finalmente, estamos listos para crear una primera instancia del flujo de trabajo implementado. Se crea una instancia de flujo de trabajo de una versión específica del flujo de trabajo, que se puede configurar en la creación.

Agregue el siguiente comando de creación a la clase principal:

```
package io.zeebe;

import io.zeebe.client.api.response.WorkflowInstanceEvent;

public class Application
{
    public static void main(String[] args)
    {
        // after the workflow is deployed

        final WorkflowInstanceEvent wfInstance =
client.newCreateInstanceCommand()
        .bpmnProcessId("order-process")
        .latestVersion()
        .send()
        .join();

        final long workflowInstanceKey = wfInstance.getWorkflowInstanceKey();

        System.out.println("Workflow instance created. Key: " +
workflowInstanceKey);

        // ...
    }
}
```

Ejecute el programa y verifique que se haya creado la instancia de flujo de trabajo. Deberías ver la salida:

```
Workflow instance created. Key: 2113425532
```

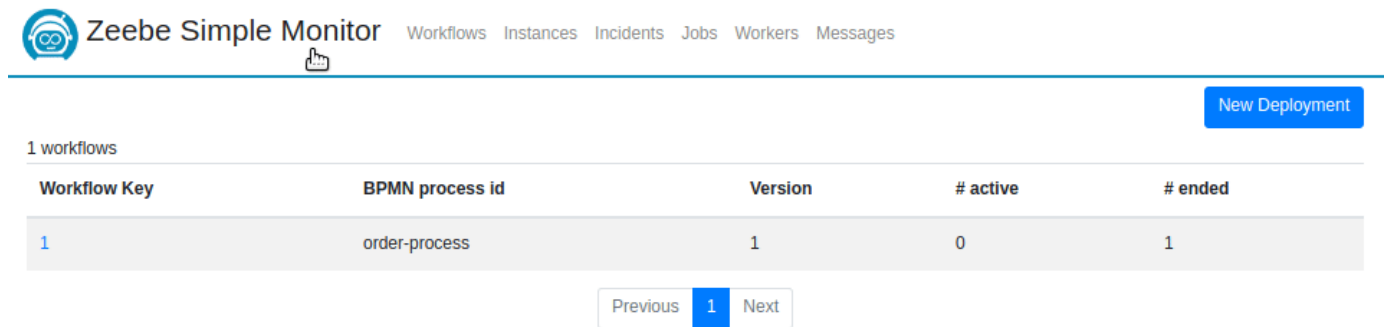
¡Lo hiciste! ¿Desea ver cómo se ejecuta la instancia de flujo de trabajo?

Si está ejecutando con Docker, simplemente abra <http://localhost:8082> en su navegador.

Si está ejecutando sin Docker:

- Inicie el monitor Zeebe con `java -jar zeebe-simple-monitor-app-*.jar`.
- Abra un navegador web y vaya a <http://localhost:8080/>.

En la interfaz de Monitor simple, verá el estado actual de la instancia de flujo de trabajo.



The screenshot shows the Zeebe Simple Monitor interface. At the top, there's a navigation bar with links: Workflows, Instances, Incidents, Jobs, Workers, Messages. A 'New Deployment' button is on the right. Below the navigation bar, there's a table with the following data:

Workflow Key	BPMN process id	Version	# active	# ended
1	order-process	1	0	1

Below the table, there are navigation buttons: Previous, 1, Next.

localhost:8080

Trabajar en un trabajo

Ahora queremos hacer algo de trabajo dentro de su flujo de trabajo. Primero, agregue algunos trabajos de servicio al diagrama BPMN y establezca los atributos requeridos. Luego amplíe su clase principal y cree un trabajador de trabajo para procesar los trabajos que se crean cuando la instancia de flujo de trabajo alcanza una tarea de servicio.

Abra el diagrama BPMN en Zeebe Modeler. Inserte algunas tareas de servicio entre el evento inicial y final.



Debe establecer el tipo de cada tarea, que identifica la naturaleza del trabajo a realizar. Establezca el tipo de la primera tarea en 'servicio de pago'.

Establezca el tipo de la segunda tarea en 'fetcher-service'.

Establezca el tipo de la tercera tarea en 'servicio de envío'.

Guarde el diagrama BPMN y vuelva a la clase principal.

Agregue las siguientes líneas para crear un trabajador de trabajo para el primer tipo de trabajo:

```
package io.zeebe;

import io.zeebe.client.api.worker.JobWorker;

public class App
{
    public static void main(String[] args)
    {
        // after the workflow instance is created

        final JobWorker jobWorker = client.newWorker()
            .jobType("payment-service")
            .handler((jobClient, job) ->
            {
                System.out.println("Collect money");

                // ...

                jobClient.newCompleteCommand(job.getKey())
                    .send()
                    .join();
            })
            .open();

        // waiting for the jobs

        // Don't close, we need to keep polling to get work
        // jobWorker.close();

        // ...
    }
}
```

Ejecute el programa y verifique que se procese el trabajo. Deberías ver la salida:

```
Collect money
```

Cuando eche un vistazo al Monitor Zeebe, podrá ver que la instancia de flujo de trabajo se movió de la primera tarea de servicio a la siguiente:



2 workflows

Workflow Key	BPMN process id	Version	# active	# ended
1	order-process	1	0	1
191	order-process	2	1	0

[Previous](#) [1](#) [Next](#)

Trabajar con datos

Por lo general, un flujo de trabajo es más que solo tareas, también hay un flujo de datos. El trabajador obtiene los datos de la instancia de flujo de trabajo para hacer su trabajo y envía el resultado a la instancia de flujo de trabajo.

En Zeebe, los datos se almacenan como pares clave-valor en forma de variables. Las variables se pueden establecer cuando se crea la instancia de flujo de trabajo. Dentro del flujo de trabajo, los trabajadores pueden leer y modificar las variables.

En nuestro ejemplo, queremos crear una instancia de flujo de trabajo con las siguientes variables:

```
"orderId": 31243  
"orderItems": [435, 182, 376]
```



La primera tarea debe leerse `orderId` como entrada y regresar `totalPrice` como resultado.

Modifique el comando de creación de instancia de flujo de trabajo y pase los datos como variables. Además, modifique el trabajador de trabajo para leer las variables de trabajo y

complete el trabajo con un resultado.

```
package io.zeebe;

public class App
{
    public static void main(String[] args)
    {
        // after the workflow is deployed

        final Map<String, Object> data = new HashMap<>();
        data.put("orderId", 31243);
        data.put("orderItems", Arrays.asList(435, 182, 376));

        final WorkflowInstanceEvent wfInstance =
client.newCreateInstanceCommand()
        .bpmnProcessId("order-process")
        .latestVersion()
        .variables(data)
        .send()
        .join();

        // ...

        final JobWorker jobWorker = client.newWorker()
        .jobType("payment-service")
        .handler((jobClient, job) ->
        {
            final Map<String, Object> variables = job.getVariablesAsMap();

            System.out.println("Process order: " +
variables.get("orderId"));
            double price = 46.50;
            System.out.println("Collect money: $" + price);

            // ...

            final Map<String, Object> result = new HashMap<>();
            result.put("totalPrice", price);


            jobClient.newCompleteCommand(job.getKey())
                .variables(result)
                .send()
                .join();
        })
        .fetchVariables("orderId")
        .open();

        // ...
    }
}
```

Ejecute el programa y verifique que se lea la variable. Deberías ver la salida:

```
Process order: 31243
Collect money: $46.50
```

Cuando echamos un vistazo al monitor Zeebe, podemos ver que la variable `totalPrice` está configurada:

 **Zeebe Simple Monitor**

Workflows Instances Incidents Jobs Workers Messages

New Deployment

3 workflows

Workflow Key	BPMN process id	Version	# active	# ended
1	order-process	1	0	1
191	order-process	2	1	0
256	order-process	3	1	0

Previous 1 Next

localhost:8080

¿Que sigue?

¡Hurra! Terminó este tutorial y aprendió el uso básico del cliente Java.

Próximos pasos:

- Aprenda más sobre los [conceptos detrás de Zeebe](#)
- Obtenga más información sobre los [flujos de trabajo BPMN](#)
- Eche un vistazo más profundo al [cliente Java](#)