Camunda 8 - Working with the Zeebe API (CLI Client)

What Will I Learn?

This course will provide you with a brief introduction to working with the Zeebe API using the Zeebe CLI (`zbctl`).
You will use `zbctl` to perform an end-to-end test against a process definition. This will allow you to gain an understanding of the actions required to connect to the Zeebe engine and test your process applications.

What Will I Learn?

At the end of this course you will be able to:

- Access a Camunda Environment
    - Access Console
    - Access Operate
- Deploy a Camunda Application
    - Deploy a Process Definition using the Zeebe CLI
- Test a Process Definition
    - Start a Process Instance using the Zeebe CLI
    - Activate a Job using the Zeebe CLI
    - Complete a Job using the Zeebe CLI
    - Fail a Job using the Zeebe CLI
    - Create a Job Worker using the Zeebe CLI
    - Send a Message Event using the Zeebe CLI
- Monitor a Process Definition
    - View a Process Instance using Operate
    - Retry an Incident using Operate
    - Cancel a Process Instance using the Zeebe CLI
    - Delete a Process Instance using Operate

Business Requirement

We will be using a fictitious organization; Camundanzia, as the scenario for this course.

Camundanzia

Scenario

The Camundanzia insurance company would like to automate the assessment of **Car Insurance Applications** as much as possible. Complex applications will still be manually reviewed by an employee.

A Camundanzia Business Analyst has already created an initial draft of the process model in BPMN. The next step is to ensure that all implementation details have been added correctly.

User Story

One simple User Story to capture Camundanzia's requirements can be found below:

User Story

- As an **Administrator**
- I want to launch a process instance with the Zeebe CLI
- So that I can confirm that the process works as expected

Acceptance Criteria

The initial acceptance criteria that we are working to can be found below:

1. An **Applicant** is able to submit a *Car Insurance Application*
2. The *Credit Score* of the **Applicant** can be retrieved automatically
3. A **Clerk** is able to review complex applications and decide whether the **Applicant** will be accepted or not
4. An **Applicant** is able to cancel their application

Outline Solution

We now have a clear set of requirements and acceptance criteria for testing the Camundanzia *Car Insurance Application Process*.

We will use a draft of the *Car Insurance Application Process* as a mechanism to explore the commands available in the Zeebe CLI Client.

Camundanzia

Scenario

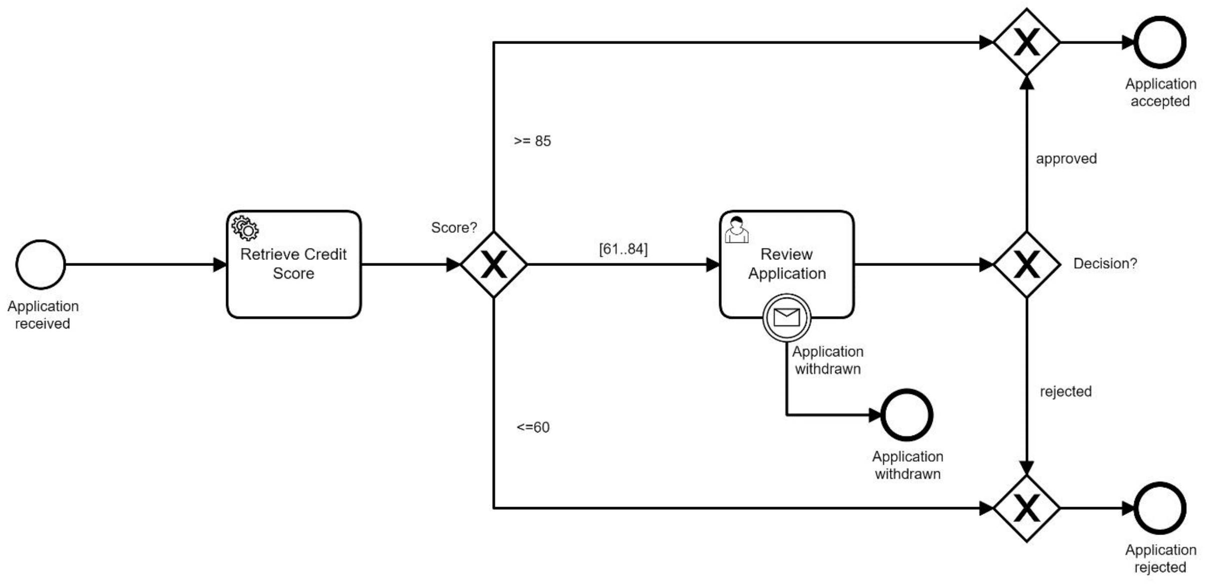
Imagine that you have recently joined Camundanzia as a new Administrator and have been assigned to the Camunda project team. Your first assignment is to understand the *Car Insurance Application Process* and confirm that it is working as expected and that all required implementation details have been provided.

Fortunately for you, a more experienced Camunda Administrator has provided you with a few hints on how to get started. Your task is to complete the required tasks based upon the outline that has been provided.

Car Insurance Application

Process Diagram

The *Car Insurance Application Process* can be obtained from this GitHub Repository and can also be seen in the diagram below.



applicationProcess.bpmn

Process Variables

The process is started with the following variables:

Variable	Type	Description
applicationId	String	Unique ID to track the application
applicationName	String	Name of the Applicant
applicantEmail	String	Email of the Applicant
applicantAge	Integer	Age of the Applicant

Business Logic

The majority of the business logic for the process is captured in the following elements:

Name	Type	Description
Application Received	Start Event	The process starts when an *Application* is received
Retrieve Credit Score	Service Task	Returns a Credit Score (Integer) between 0 and 100
Review Application	User Task	If the Credit Score is between 61 and 84, a **Clerk** must review the application and make a decision

Name	Type	Description
Application Withdrawn	End Event	An *Application* may be withdrawn if it is in the **Review Application** state

Outline Solution

The more experienced Administrator has provided an overview of the work that must be completed for each of the stated acceptance criteria. This covers the configuration that you, the new Administrator must perform and the Camunda components that offer the required functionality.

Connect to Zeebe Engine using the CLI Client

We will be testing the solution by issuing commands to a Zeebe Cluster using the Zeebe CLI Client.

Install Zeebe CLI Client

Install the Zeebe CLI Client (zbctl)

Configure Cluster Credentials

Configure the Zeebe CLI Client

Connect to Cluster

Obtain information from a Zeebe Cluster

Deploy and Start the Process

Once the connection to the Zeebe Cluster has been established, we can start to validate the *Car Insurance Application Process*.

Deploy Process Definition

Deploy the *Car Insurance Application* process definition.

Start Process Instance

Start a *Car Insurance Application* process instance.

Cancel Process Instance

Cancel a *Car Insurance Application* process instance.

Simulate Retrieve Credit Score

We will simulate the behavior of *Retrieve Credit Score* to validate the process logic before developing a Job Worker.

Activate Job

Activate a *Retrieve Credit Score* Job

Complete Job

Complete a *Retrieve Credit Score* Job

Fail Job

Fail a *Retrieve Credit Score* Job

Configure Worker for Service Task

We will configure a Worker to process the **Retrieve Credit Score** Service Task.

Create Worker

Register a Worker that will handle the **Retrieve Credit Score** Jobs.

Review User Task

In addition to the Tasklist, User Tasks can also be completed by using the Zeebe API.

Complete User Task

Complete the **Review Application** User Task.

Send BPMN Message

Finally, BPMN Message Events can also be thrown using the Zeebe API.

Send Message

Throw a message which will be caught by the **Application Withdrawn** Boundary Event.

Development Plan

Now that we have identified the steps to validate our solution, we can create a plan of the work that needs to be performed and the tools that will be required.

- **Install Zeebe CLI Client**
    - This will be performed using the tools appropriate for your machine
- **Configure Cluster Credentials**
    - This will be performed using the Console and zbctl
- **Connect to Cluster**
    - This will be performed using zbctl
- **Deploy Process Definition**
    - This will be performed using zbctl and Operate
- **Start Process Instance**
    - This will be performed using zbctl and Operate
- **Cancel Process Instance**
    - This will be performed using zbctl and Operate
- **Activate Job**
    - This will be performed using zbctl and Operate
- **Complete Job**
    - This will be performed using zbctl and Operate
- **Fail Job**
    - This will be performed using zbctl and Operate
- **Create Worker**
    - This will be performed using zbctl and Operate
- **Complete User Task**
    - This will be performed using zbctl and Operate
- **Send Message**
    - This will be performed using zbctl and Operate

Ready to get started? Then proceed to the next lesson.

Installation

`zbctl` is a Command Line Interface (CLI) which can be used to interact with Camunda using the Zeebe API. `zbctl` connects to a Camunda Cluster using gRPC which is a high-performance, open-source RPC framework.

Note

The CLI client doesn't support multi-tenancy and can only be used when multi-tenancy is disabled. You can find more details on multi-tenancy here.

Installation

`zbctl` can be installed locally on your machine using any one of the following methods.

npm

The installation can be done quickly via the package manager **npm**.

```
npm i -g zbctl
```

Binary Installation

You can also download a binary for your operating system from the Zeebe GitHub Releases Page.

Docker

You can also use `zbctl` with Docker if the direct installation method is unsuitable. You will need to pass in your client credentials using the syntax below.

```
docker run --env ZEEBE_ADDRESS=$ZEEBE_ADDRESS \
           --env ZEEBE_CLIENT_ID=$ZEEBE_CLIENT_ID \
           --env ZEEBE_CLIENT_SECRET=$ZEEBE_CLIENT_SECRET \
           --env
ZEEBE_AUTHORIZATION_SERVER_URL=$ZEEBE_AUTHORIZATION_SERVER_URL \
           sitapati/zbctl status
```

Configure Environment Variables

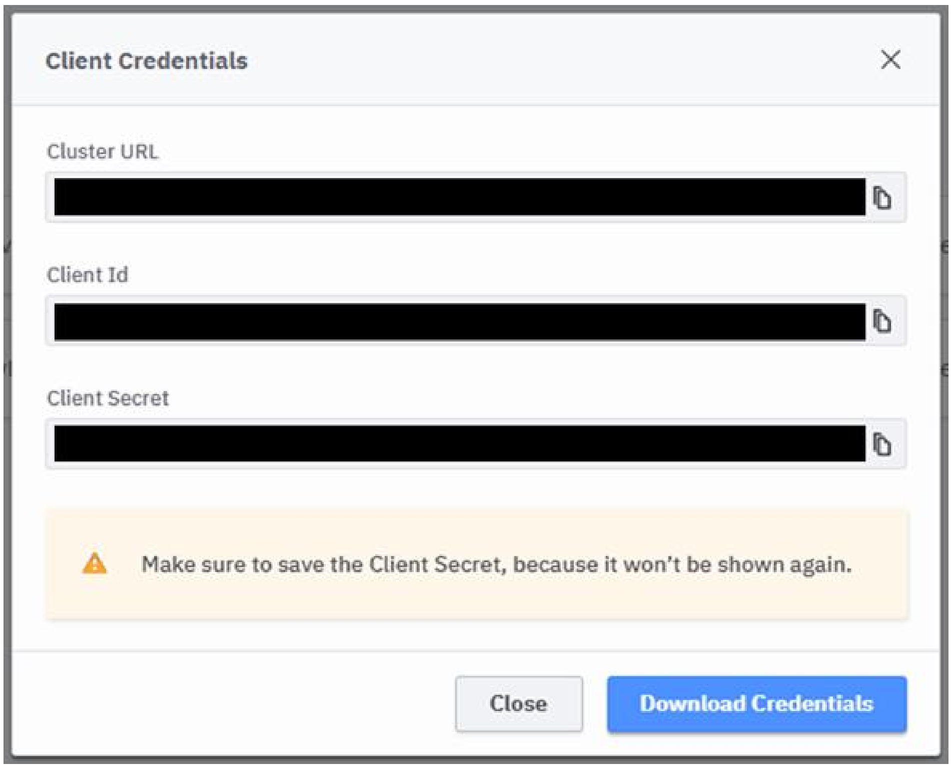
Note

This course is optimized for **Camunda SaaS**.

When using `zbctl`, it is recommended that you define environment variables so that you do not need to pass your client credentials with each request.

Download Client Credentials

Whenever you create a new set of client credentials for Camunda SaaS, you have the option to download a file which also contains the required configuration properties.


Client Credentials

Cluster URL

[redacted]

Client Id

[redacted]

Client Secret

[redacted]

Make sure to save the Client Secret, because it won't be shown again.

Configure Environment Variables

You can configure environment variables on your machine using the instructions below:

Setting Environment Variables on Linux

After downloading your client credentials, you will need to `source` them so that they are available to your environment.
`source ~/Downloads/CamundaCloudMgmtAPI-Client-test-client.txt`
Alternatively, you can execute the following commands using the Terminal so that they are available to your environment.

```
export ZEEBE_ADDRESS='[Zeebe API]'
export ZEEBE_CLIENT_ID='[Client ID]'
export ZEEBE_CLIENT_SECRET='[Client Secret]'
export ZEEBE_AUTHORIZATION_SERVER_URL='[OAuth API]'
```

Setting Environment Variables on Windows (Powershell)

Execute the following commands using Powershell so that they are available to your environment.

```
$env:ZEEBE_ADDRESS="[Zeebe API]"
$env:ZEEBE_CLIENT_ID="[Client ID]"
$env:ZEEBE_CLIENT_SECRET="[Client Secret]"
$env:ZEEBE_AUTHORIZATION_SERVER_URL="[OAuth API]"
```

Setting Environment Variables on Windows (Command Line)

Execute the following commands using the Command Line (cmd.exe) so that they are available to your environment.

```
set ZEEBE_ADDRESS=[Zeebe API]
set ZEEBE_CLIENT_ID=[Client ID]
set ZEEBE_CLIENT_SECRET=[Client Secret]
set ZEEBE_AUTHORIZATION_SERVER_URL=[OAuth API]
```

Obtaining Help

You can obtain help at any time by passing the `--help` flag along with the command that you wish to execute.
For example, issuing `zbctl status --help` will show you the help text and available flags for the `status` command.

```
Checks the current status of the cluster

Usage:
  zbctl status [flags]

Flags:
  -h, --help            help for status
  -o, --output string   Specify output format. Default is human readable.
Possible Values: human, json (default "human")
```

Validating the Installation

You can now validate your installation and client credentials by issuing a `status` command to `zbctl`; you can see the available commands in the below:

```
Usage:
  zbctl [command]

Available Commands:
  activate    Activate a resource
  broadcast   Broadcast a signal
  cancel      Cancel resource
  complete    Complete a resource
  completion  Generate the autocompletion script for the specified shell
  create      Create resources
  deploy      Deploys new resources for each file provided
  evaluate    Evaluate resources
  fail        Fail a resource
  generate    Generate documentation
  help        Help about any command
  publish     Publish a message
  resolve     Resolve a resource
  set         Set a resource
  status      Checks the current status of the cluster
  throwError  Throw an error
  update      Update a resource
  version     Print the version of zbctl
```

Passing Client Credentials using Flags

You can also pass the `--address`, `--clientId` and `--clientSecret` as optional flags along with any command, further information can be found here.

Name	Data Type	Description
address	String	Specify a contact point address. If omitted, will read from the environment variable 'ZEEBE_ADDRESS' (default '127.0.0.1:26500')
clientId	String	Specify a client identifier to request an access token. If omitted, will read from the environment variable 'ZEEBE_CLIENT_ID'
clientSecret	String	Specify a client secret to request an access token. If omitted, will read from the environment variable 'ZEEBE_CLIENT_SECRET'

Test the Cluster Connection

To confirm that everything is working as expected, use the following command to connect to your cluster using `zbctl`

```
zbctl status
```

Or, the following if you prefer to pass your client credentials using flags rather than environment variables.

```
zbctl status --address "[Zeebe API]" --clientId "[Client ID]" --clientSecret
"[Client Secret]"
```

If everything is working as expected, you should see output similar to the below:

```
Cluster size: 3
Partitions count: 3
Replication factor: 3
Gateway version: 8.3.3
Brokers:
  Broker 0 - zeebe-0.zeebe-broker-service.xxx.svc.cluster.local:26501
    Version: 8.3.3
    Partition 1 : Follower, Healthy
    Partition 2 : Follower, Healthy
    Partition 3 : Follower, Healthy
  Broker 1 - zeebe-1.zeebe-broker-service.xxx.svc.cluster.local:26501
    Version: 8.3.3
    Partition 1 : Leader, Healthy
    Partition 2 : Follower, Healthy
    Partition 3 : Follower, Healthy
  Broker 2 - zeebe-2.zeebe-broker-service.xxx.svc.cluster.local:26501
    Version: 8.3.3
    Partition 1 : Follower, Healthy
    Partition 2 : Leader, Healthy
    Partition 3 : Leader, Healthy
```

Connected to a Cluster

Congratulations! You have installed `zbctl` on your local machine, configured the client credentials and obtained the status of your cluster.

Deploy Process

To deploy a process, we will use the **DeployResource RPC** which deploys one or more processes to Zeebe.

Atomic Transaction

A request to the **DeployResource RPC** is an atomic call i.e. either all processes are deployed or none are deployed.

Deploy a Process Definition

The following command can be used to deploy a process definition to Zeebe. Ensure that you have downloaded the *Car Insurance Application Process* and are in the correct directory before executing the command.

Car Insurance Application Process

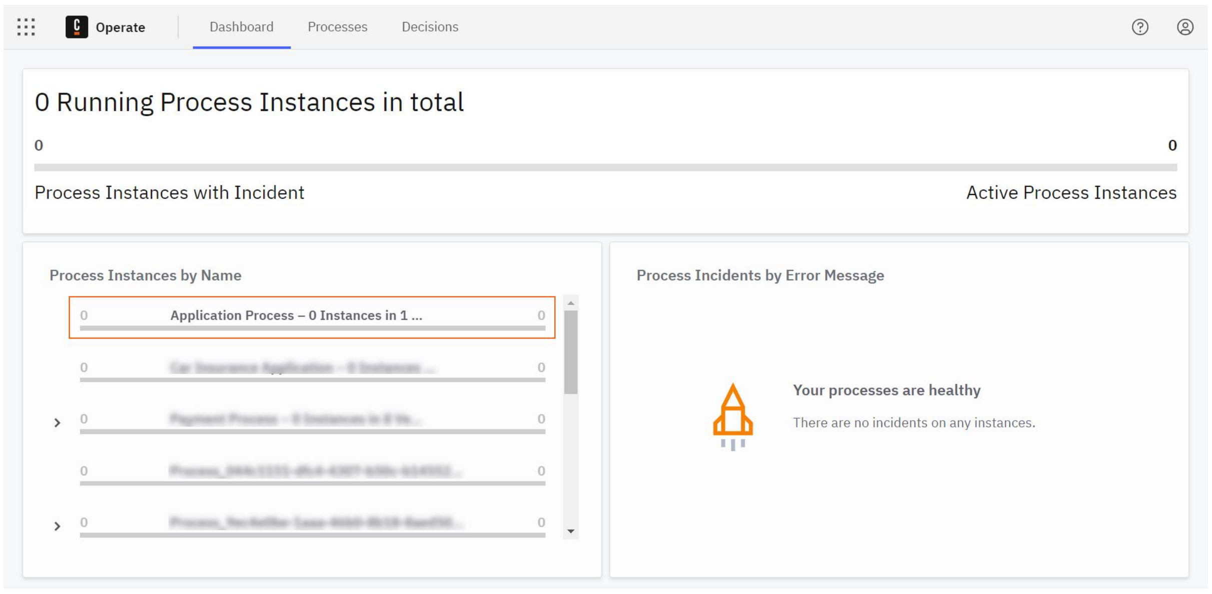
The *Car Insurance Application Process* can be obtained from this GitHub Repository

```
zbctl deploy ApplicationProcess.bpmn
```

If the deployment was successful, you should see output similar to the below:

```
{
  "key": "2251799813685250",
  "processes": [
    {
      "bpmnProcessId": "ApplicationProcess",
      "version": 1,
      "processDefinitionKey": "2251799813685249",
      "resourceName": "ApplicationProcess.bpmn"
    }
  ]
}
```

If you now launch **Operate**, you will see that the **Process Definition** has been successfully deployed and has been marked as Version 1. Version numbers are automatically set by Zeebe upon deployment.



0 Running Process Instances in total

0 Process Instances with Incident — 0 Active Process Instances

Process Instances by Name

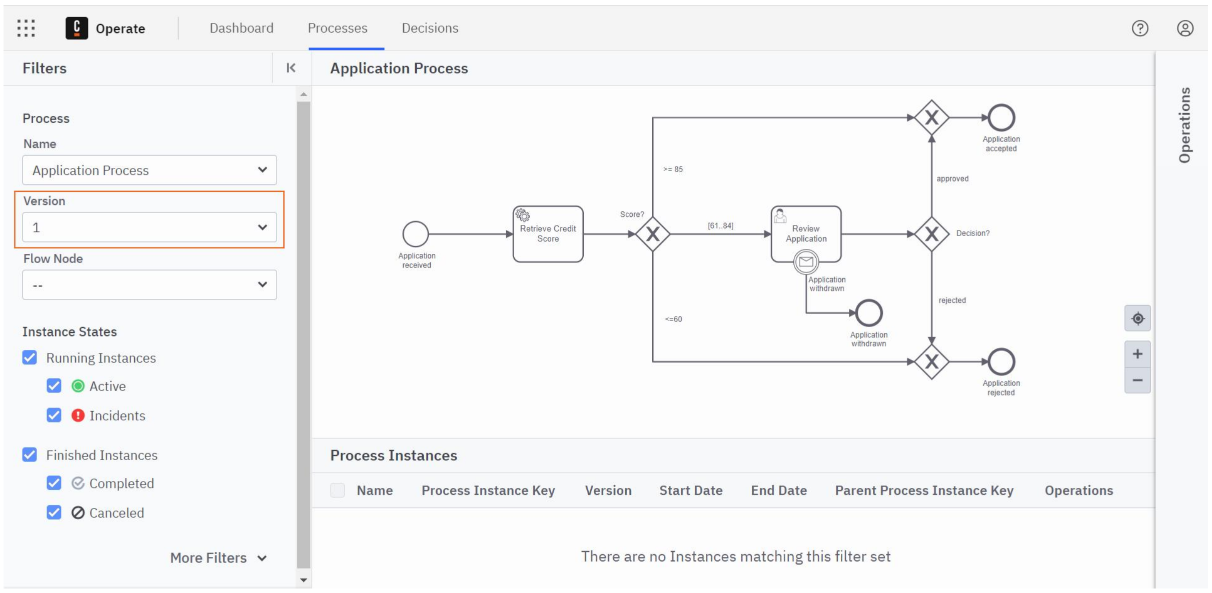
- 0 Application Process – 0 Instances in 1 ... 0
- 0 [redacted] 0
- 0 [redacted] 0
- 0 [redacted] 0
- 0 [redacted] 0

Process Incidents by Error Message

Your processes are healthy

There are no incidents on any instances.

However, you can also see that we don't have any **Process Instances** yet.



Filters

Process

Name: Application Process

Version: 1

Flow Node: --

Instance States

- ☑ Running Instances
  - ☑ Active
  - ☑ Incidents
- ☑ Finished Instances
  - ☑ Completed
  - ☑ Canceled

Application Process

Process Instances

Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance Key	Operations

There are no Instances matching this filter set

Process Definition Versioning

Let's run the same command again to see how Zeebe manages versioning.

```
zbctl deploy ApplicationProcess.bpmn
```

As you can see from the output below, the version of the **Process Definition** is still Version 1 after deploying for a second time.

```
{
  "key": "2251799813685250",
  "processes": [
    {
      "bpmnProcessId": "ApplicationProcess",
      "version": 1,
      "processDefinitionKey": "2251799813685249",
      "resourceName": "ApplicationProcess.bpmn"
    }
  ]
}
```

How are Process Definitions versioned by Zeebe?

You can determine the versioning behavior of Zeebe from the rules below:

- **Duplicate Checking** - If a Process Definition with the same name and the same resources has already been deployed to Zeebe, a new version will not be created.
- **Model Changes** - The slightest change to a Process Definition, even if the execution logic and technical properties remain the same e.g. moving a Task to the left or right slightly will result in a new version of the Process Definition.

Deploy Multiple Process Definitions

The same command can also be used to deploy multiple processes at the same time. You simply need to specify the filenames separated by a space. For example:

```
zbctl deploy ApplicationProcess.bpmn RejectionProcess.bpmn
```

If the deployment was successful, you should see output similar to the below:

```
{
  "key":  "2251799821876913",
  "processes":  [
    {
      "bpmnProcessId":  "ApplicationProcess",
      "version":  1,
      "processDefinitionKey":  "2251799821876911",
      "resourceName":  "ApplicationProcess.bpmn"
    },
    {
      "bpmnProcessId":  "RejectionProcess",
      "version":  1,
      "processDefinitionKey":  "2251799821876912",
      "resourceName":  "RejectionProcess.bpmn"
    }
  ]
}
```

Deployed a Process Definition

Congratulations! You have successfully deployed a Process Definition using `zbctl`

Starting and Cancelling

To start a Process Instance, we will use the **CreateProcessInstance RPC**. This command creates and starts an instance from a specific Process Definition.

Process Definition ID & Process Key

The Process Definition used to create an instance can either be specified by the Process Definition ID or the Process Key.

- **Process Definition ID** - For example, `2251799813685714` and is defined by Zeebe.
- **Process Key** - For example, `ApplicationProcess` and is defined by the Process Designer. The latest version of the Process Definition will be used by default.

Start a Process Instance

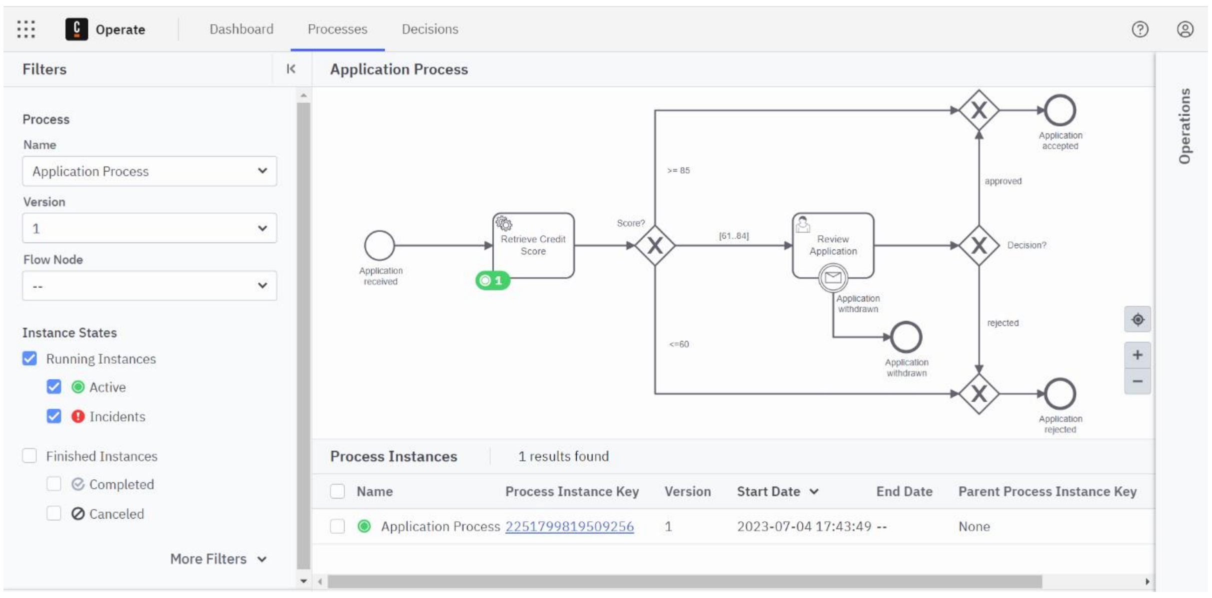
The following command can be used to create and start a process instance in Zeebe.

```
zbctl create instance ApplicationProcess
```

If the instance creation was successful, you should see output similar to the below:

```
{
  "processDefinitionKey": "2251799813685249",
  "bpmnProcessId": "ApplicationProcess",
  "version": 1,
  "processInstanceKey": "2251799813685254"
}
```

Now, when we look in **Operate**, we can also see that a process instance has been created and started successfully.



Process
Name: Application Process
Version: 1
Flow Node: --

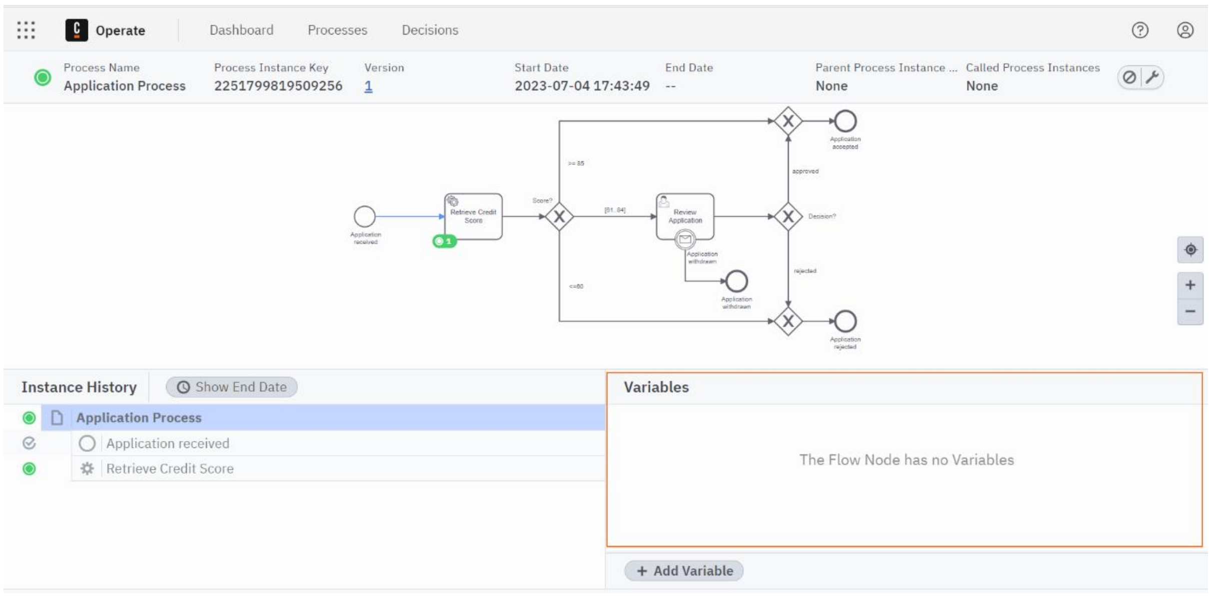
Instance States
- [x] Running Instances
  - [x] Active
  - [x] Incidents
- [ ] Finished Instances
  - [ ] Completed
  - [ ] Canceled

More Filters

Process Instances 1 results found

Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance Key
Application Process	2251799819509256	1	2023-07-04 17:43:49	--	None

However, we did not specify any variables when starting the process. It is important that the required information about the application is available when the process is started.



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance ...	Called Process Instances
Application Process	2251799819509256	1	2023-07-04 17:43:49	--	None	None

Instance History
- Application Process
  - Application received
  - Retrieve Credit Score

Variables

The Flow Node has no Variables

We could now modify the process instance to supply the missing variables. However, in order to keep to a realistic development process we will cancel the current process instance and then start a new one with the required variables.

Cancel a Process Instance

To cancel a Process Instance, we will use **CancelProcessInstance RPC**. This command cancels an existing process instance.

To cancel the process instance that we just started, you can use the following command:

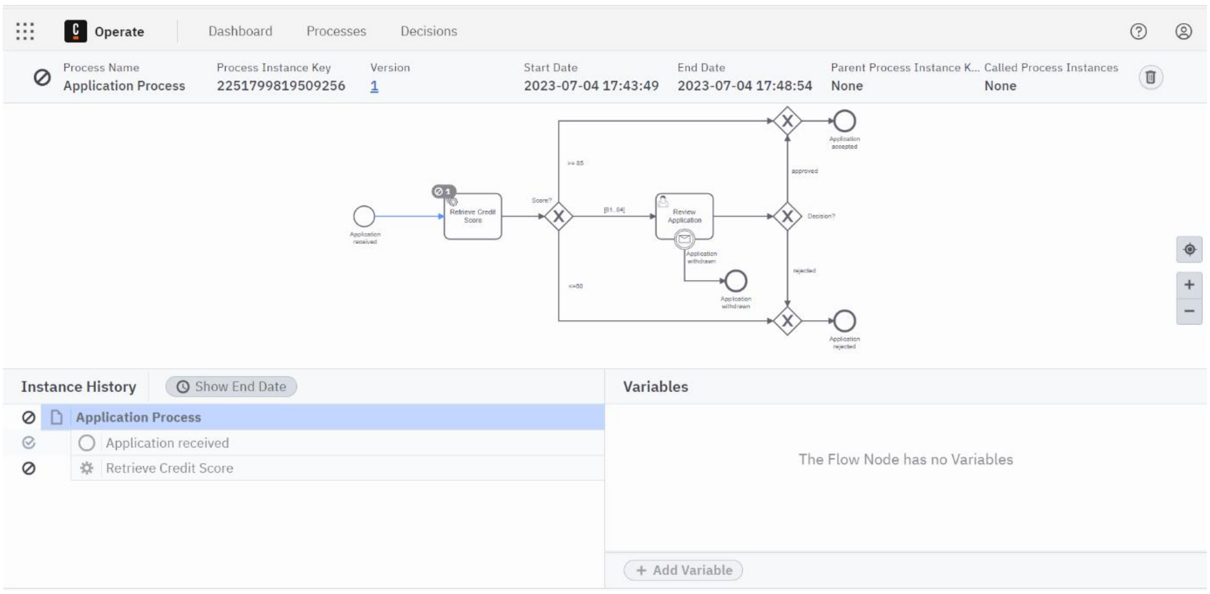
```
zbctl cancel instance [processInstanceKey]
```

Process Instance Key

The process instance key is returned by `zbctl` when a process is started. Alternatively, you can obtain the process instance key from Operate.
If the instance cancellation was successful, you should see output similar to the below:

```
Canceled process instance with key '2251799813685254'
```

Finally, you can see that the process instance has now been cancelled if you refresh Operate.



Process Name: Application Process
Process Instance Key: 2251799819509256
Version: 1
Start Date: 2023-07-04 17:43:49
End Date: 2023-07-04 17:48:54
Parent Process Instance K...: None
Called Process Instances: None

Instance History
- Application Process
  - Application received
  - Retrieve Credit Score

Variables

The Flow Node has no Variables

Start a Process Instance with Process Variables

The **CreateProcessInstance RPC** can be used in conjunction with the `--variables` flag to specify one or more process variables as a JSON string.

```
zbctl create instance ApplicationProcess --variables
"{\"applicationId\":\"Application-123\",\"applicantName\":\"Camundo
Camunda\",\"applicantEmail\":\"camundo@camunda.com\",\"applicantAge\":21}"
```

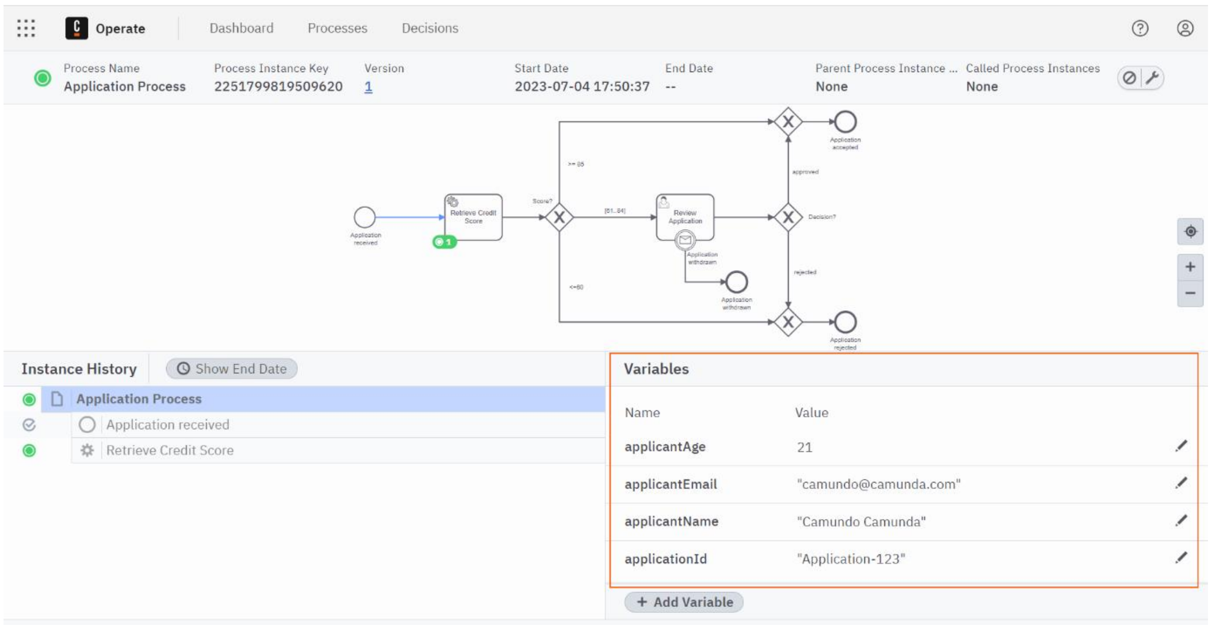
Escaping Quotes in Windows Powershell

If the instance creation was successful, you should see output similar to the below:

```
{
  "processDefinitionKey": "2251799813685249",
  "bpmnProcessId": "ApplicationProcess",
  "version": 1,
  "processInstanceKey": "2251799813685260"
}
```

Process Definition Version

You can also specify the version of the process which should be executed with the `--version` flag. By default, the latest version of the process will be used if this flag is not passed. Finally, in Operate, you can now see that the process variables have been correctly passed to the process instance.



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance ...	Called Process Instances
Application Process	2251799819509620	1	2023-07-04 17:50:37	--	None	None

Instance History

- Application Process
  - Application received
  - Retrieve Credit Score

Variables

Name	Value
applicantAge	21
applicantEmail	"camundo@camunda.com"
applicantName	"Camundo Camunda"
applicationId	"Application-123"

Started and Cancelled a Process Instance

Congratulations! You have successfully started and cancelled a Process Instance using `zbctl`

Managing Jobs

Overview

Before we start to use `zbctl` to manage Jobs, let's refresh our knowledge on Incidents, Jobs and Job Workers.

What is an Incident?

An **Incident** represents a problem in process execution. This means that a process instance is stuck at a particular point, and requires user interaction to resolve the problem.

Incidents are created in different situations, including the following:

- A **Job** is failed and it has no retries left.
- An input or output variable mapping can't be applied.
- A condition can't be evaluated.
- A decision can't be evaluated.

Errors & Incidents

Not all errors will necessarily lead to **Incidents**. For example, unexpected errors in Zeebe do not always result in **Incidents**.

What is a Job?

A **Job** represents a task that needs to be performed by a process. For example, a Job is created by Zeebe whenever a process instance reaches a Service Task. A **Job** representing the Service Task must be completed before the process is able to continue.

Jobs have the following properties:

Name	**Description**
Type	Describes the work item and is defined in each task in the process. The type is referenced by workers to request the jobs they are able to perform.
Custom Headers	Additional static metadata that is defined in the process. Custom headers are used to configure reusable job workers (e.g. a notify Slack worker might read out the Slack channel from its header.)
Key	Unique key to identify a job. The key is used to hand in the results of a job execution, or to report failures during job execution.
Variables	The contextual/business data of the process instance required by the worker to do its work.

What is a Job Worker?

A **Job Worker** is a service capable of performing a particular task in a process. **Job Workers** are used to manage the interactions between the Zeebe Engine and the outside world and perform the following actions:

- **Request Jobs -** Request jobs of a certain type on a regular interval (i.e. polling). This interval and the number of jobs requested are configurable.
- **Complete or Fail Jobs -** After working on an **Activated Job**, a **Job Worker** informs Zeebe that the Job has either **Completed** or **failed**.

Complete or Fail

A **Job Worker** may either **Complete** or **Fail** a **Job**.

- **Complete -** When the **Job Worker** completes its work, it sends a complete job command along with any variables, which in turn is merged into the process instance. This is how the Job Worker exposes the results of its work.
- **Fail -** If the **Job Worker** cannot successfully complete its work, it sends a fail job command along with an error message.

Activate Job

To activate a Job, we will use the **ActivateJobs RPC**.

What is Job Activation?

A **Job** represents a task which needs to be completed in order for the process to continue. Activating a **Job** makes it available to a Zeebe Client for processing so that it can be either **Completed** or **Failed**. An **Activated Job** is a **Job** that has been assigned to a specific Zeebe Client.

Activate a Job

We can activate Jobs for the **creditscore** type in our process by issuing the following command:

```
zbctl activate jobs creditscore
```

If the activation was successful, you should see output similar to the below. This includes information about the process instance and the variables associated with that process instance.

```
{
  "jobs": [
    {
      "key": "2251799819509629",
      "type": "creditscore",
      "processInstanceKey": "2251799819509620",
      "bpmnProcessId": "ApplicationProcess",
      "processDefinitionVersion": 1,
      "processDefinitionKey": "2251799813685249",
      "elementId": "Activity_0jpftwp",
      "elementInstanceKey": "2251799813685268",
      "customHeaders": "{}",
      "worker": "zbctl",
      "retries": 3,
      "deadline": "1640870839966",
      "variables": "{\"applicantAge\":21,\"applicantName\":\"Camunda
Camunda\",\"applicationId\":\"Application-
123\",\"applicantEmail\":\"camundo@camunda.com\"}"
    }
  ]
}
```

No Jobs Returned

You may receive output similar to the below if there are no Jobs available for activation.

```
{
  "jobs":  []
}
```

If this is the case, you can simply start a new process instance using the `zbctl create instance ApplicationProcess` command from a previous lesson.

This Job will now be assigned to a `client` until the time specified in the `deadline` has passed; the next step is to either **Complete** or **Fail** the Job. If no action is taken before the deadline, the Job will become eligible for activation by another client.

Configuring Job Activation

Further information on the parameters that are most commonly used in conjunction with the **ActivateJobs RPC** can be found below.

Name	**Description**
maxJobsToActivate	Specify the maximum amount of jobs to activate (default 1)
requestTimeout	Specify the request timeout for a long polling interval. With long polling, a request will be kept open while no jobs are available. The request is completed when at least one job becomes available
timeout	Specify the maximum time that a service takes to complete the job. Once the timeout is reached (default 5m0s), the job can be done by a different worker again. Zeebe assumes that the worker died during execution.
variables	Specify the list of variable names which should be fetch on job activation (comma-separated), which can decrease load and improve performance
worker	Specify the name of the worker (default "zbctl"), mostly used for logging purposes.

Activated a Job

Congratulations! You have successfully activated a Job using `zbctl`

Complete Job

To complete a Job, we will use the **CompleteJob RPC**. This command completes a Job with the provided payload and allows for completion of the associated Service Task.

What is Job Completion?

An **Activated Job** represents a task that has been assigned to a Zeebe Client which needs to be completed in order for the process to continue. Completing an **Activated Job** informs Zeebe that the task has now been completed and execution can continue.

Complete a Job

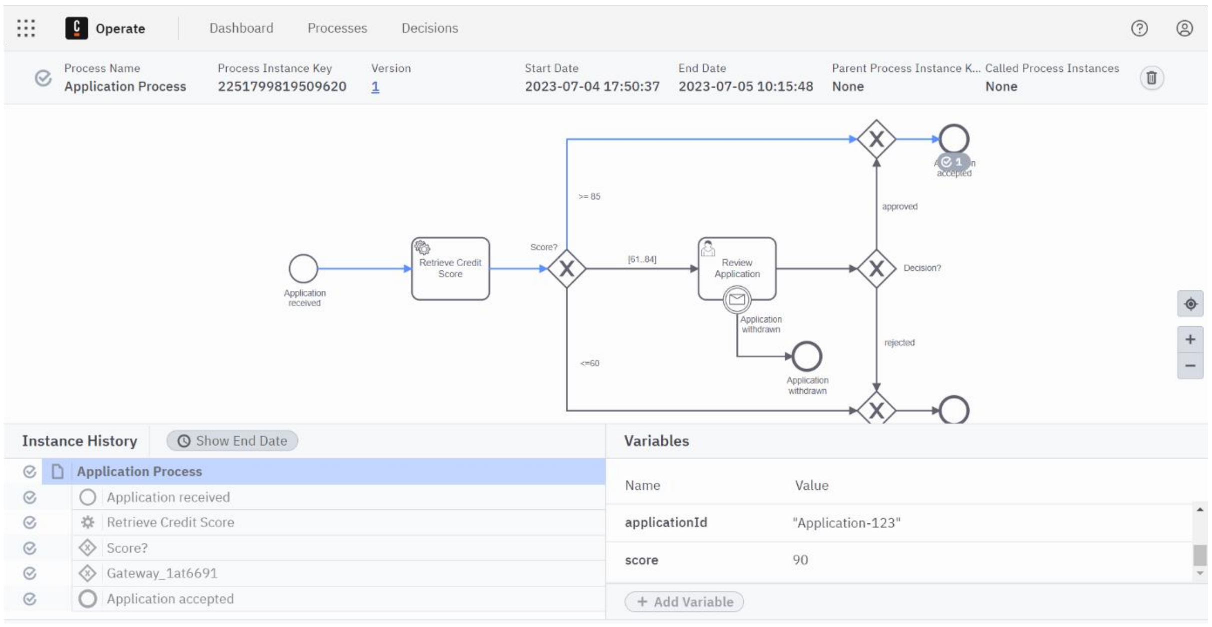
We can complete the previously activated Job by issuing the command below.

```
zbctl complete job [jobKey] --variables "{\"score\":90}"
```

- `[jobKey]` should be replaced with the `key` that was returned by the previously activated Job
- `--variables "{\"score\":90}"` creates a new Process Variable named `score` and sets the value to 90

If the completion was successful, you should see output similar to the below. Note that this also includes confirmation that a process variable was updated.

Now, when we look in Operate, we can see that the Job and the Process Instance have been completed:



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance K...	Called Process Instances
Application Process	2251799819509620	1	2023-07-04 17:50:37	2023-07-05 10:15:48	None	None

Instance History

- Application Process
  - Application received
  - Retrieve Credit Score
  - Score?
  - Gateway_1at6691
  - Application accepted

Variables

Name	Value
applicationId	"Application-123"
score	90

Why did the Process Instance complete?

By setting the `score` variable to 90, the process token followed the *Application Accepted* sequence flow as the flow condition has been set to: `score >- 85`.

Completed a Job

Congratulations! You have successfully completed a Job using `zbctl`

Fail Job

To fail a Job, we will use the **FailJob RPC**, this command will also create an Incident if the retries argument is zero or negative.

What is Job Failure?

An **Activated Job** represents a task that has been assigned to a Zeebe Client which needs to be completed in order for the process to continue. Failing an **Activated Job** informs Zeebe that the task could not be completed and that further processing is required through either a retry or resolution of an Incident.

Fail a Job

We have already seen how to use the `activate` and `complete` commands in `zbctl`. In this lesson, we will see how to use the `fail` command to simulate a real-life use case.
The steps that we will follow in the sections below are:

1. **Start** a new Process Instance
2. **Activate** the Job associated with the Process Instance
3. **Fail** the Job and provide a suitable error message
4. **Review** the Incident in Operate
5. **Retry** the Job

6. **Activate** the previously failed Job associated with the Process Instance
7. **Complete** the Job

Start Process Instance

We can start by creating a new process instance with variables using the following command:

```
zbctl create instance ApplicationProcess --variables
"{\"applicationId\":\"Application-123\",\"applicantName\":\"Camundo
Camunda\",\"applicantEmail\":\"camunda@camunda.com\",\"applicantAge\":21}"
```

If the instance creation was successful, you should see output similar to the below:

```
{
  "processDefinitionKey": "2251799813685249",
  "bpmnProcessId": "ApplicationProcess",
  "version": 1,
  "processInstanceKey": "2251799813685280"
}
```

Activate the Job

We can activate the Job associated with the process instance that we just created using the following command:

```
zbctl activate jobs creditscore
```

If the job activation was successful, you should see output similar to the below:

```
{
  "jobs": [
    {
      "key": "2251799813685285",
      "type": "creditscore",
      "processInstanceKey": "2251799813685280",
      "bpmnProcessId": "ApplicationProcess",
      "processDefinitionVersion": 1,
      "processDefinitionKey": "2251799813685249",
      "elementId": "Activity_0jpftwp",
      "elementInstanceKey": "2251799813685284",
      "customHeaders": "{}",
      "worker": "zbctl",
      "retries": 3,
      "deadline": "1640872681780",
      "variables": "{\"applicantAge\":21,\"applicantName\":\"Camundo
Camunda\",\"applicationId\":\"Application-
123\",\"applicantEmail\":\"camunda@camunda.com\"}"
    }
  ]
}
```

Fail the Job

We can now fail the Job that we just activated using the following command:

```
zbctl fail job [jobKey] --retries 0 --errorMessage "Could not receive score
from external service"
```

- `[jobKey]` should be replaced with the `key` that was returned by the previously activated Job
- `retries` is used to inform Zeebe that the Job should not be retried again
- `--errorMessage"` is used to create add a human readable error message to the Incident that is created

If failing the Job was successful, you should see output similar to the below:

```
Failed job with key '2251799813685285' and set remaining retries to '0' and
variables '{}' with retry backoff '0s'
```

Further information on the parameters that are most commonly used in conjunction with the **FailJobs RPC** can be found below.

Name	**Description**
retries	The amount of retries the job should have left
retryBackoff	The backoff timeout (in ms) for the next retry
errorMessage	An optional message describing why the job failed. This is particularly useful if a job runs out of retries and an incident is raised, as it this message can help explain why an incident was raised

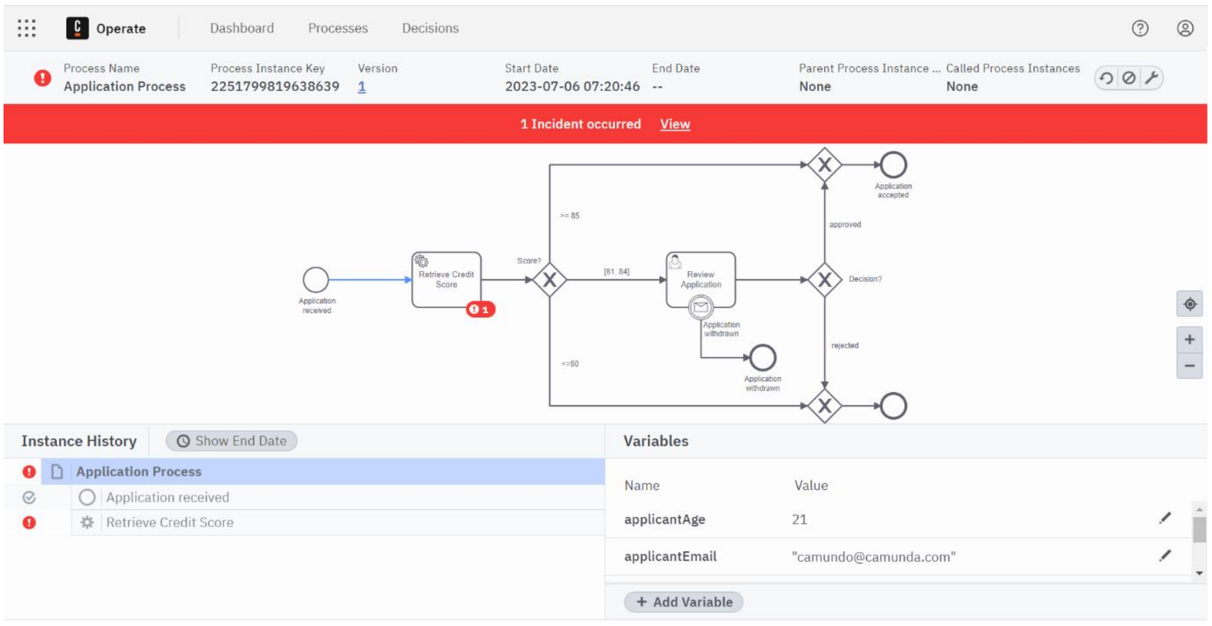
Job Retry Behaviour

A different action will be taken by Zeebe depending on the value that was specified for `retries` in the above

- If `retries` is greater than zero, the job is retried and reassigned.
- If `retries` is zero or negative, an incident is raised and the job is not retried until the incident is resolved.

View the Incident in Operate

If you now launch Operate, you will see that the Process Instance that you launched is now showing an Incident.



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance ...	Called Process Instances
Application Process	2251799819638639	1	2023-07-06 07:20:46	--	None	None

1 Incident occurred View

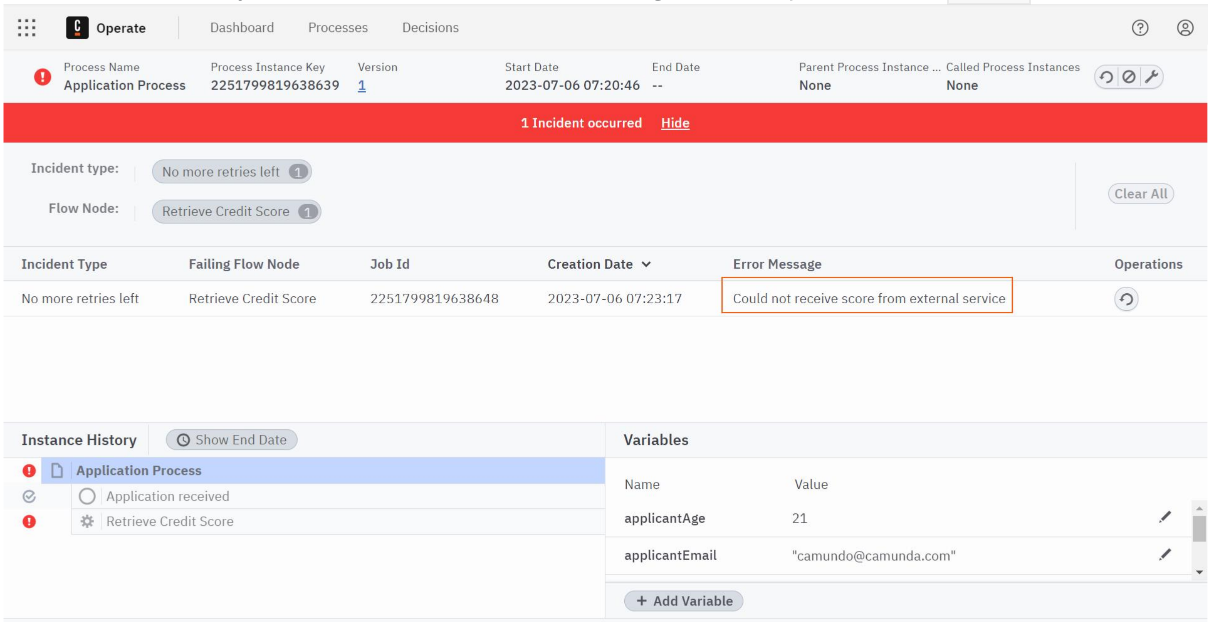
Instance History

- Application Process
  - Application received
  - Retrieve Credit Score

Variables

Name	Value
applicantAge	21
applicantEmail	"camundo@camunda.com"

In the Detail View, you can also see the error message that we provided via `zbctl`



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance ...	Called Process Instances
Application Process	2251799819638639	1	2023-07-06 07:20:46	--	None	None

1 Incident occurred Hide

Incident type: No more retries left 1

Flow Node: Retrieve Credit Score 1

Incident Type	Failing Flow Node	Job Id	Creation Date	Error Message	Operations
No more retries left	Retrieve Credit Score	2251799819638648	2023-07-06 07:23:17	Could not receive score from external service	

Instance History

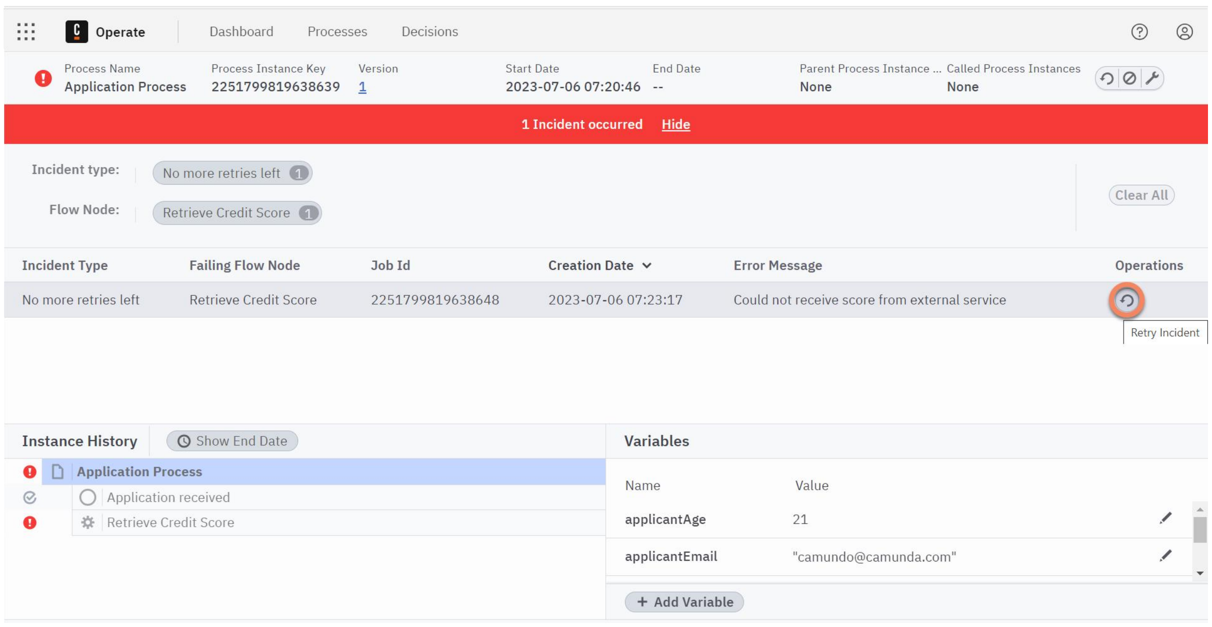
- Application Process
  - Application received
  - Retrieve Credit Score

Variables

Name	Value
applicantAge	21
applicantEmail	"camundo@camunda.com"

Retry the Incident

We can now retry the Incident in Operate so that we can continue with our example and successfully complete the Job.



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance ...	Called Process Instances
Application Process	2251799819638639	1	2023-07-06 07:20:46	--	None	None

1 Incident occurred Hide

Incident type: No more retries left 1
Flow Node: Retrieve Credit Score 1
Clear All

Incident Type	Failing Flow Node	Job Id	Creation Date	Error Message	Operations
No more retries left	Retrieve Credit Score	2251799819638648	2023-07-06 07:23:17	Could not receive score from external service	Retry Incident

Instance History
Show End Date
Application Process
Application received
Retrieve Credit Score

Variables

Name	Value
applicantAge	21
applicantEmail	"camundo@camunda.com"

+ Add Variable

Resolve Incident

It is also possible to resolve an Incident using `zbctl` and the ResolveIncident RPC
This can be achieved by passing the ID of the Incident to `zbctl` as can be seen below:

```
zbctl resolve incident 2251799821886326
```

Activate the Job (Again)

We can now re-activate the previously job using the following command:

```
zbctl activate jobs creditscore
```

If the job activation was successful, you should see output similar to the below:

```
{
  "jobs": [
    {
      "key": "2251799813685285",
      "type": "creditscore",
      "processInstanceKey": "2251799813685280",
      "bpmnProcessId": "ApplicationProcess",
      "processDefinitionVersion": 1,
      "processDefinitionKey": "2251799813685249",
      "elementId": "Activity_0jpftwp",
      "elementInstanceKey": "2251799813685284",
      "customHeaders": "{}",
      "worker": "zbctl",
      "retries": 3,
      "deadline": "1640872681780",
      "variables": "{\"applicantAge\":21,\"applicantName\":\"Camundo
Camunda\",\"applicationId\":\"Application-
123\",\"applicantEmail\":\"camunda@camunda.com\"}"
    }
  ]
}
```

Complete the Job

This time, instead of failing the Job we will use the `complete` command which will allow the process instance to complete:

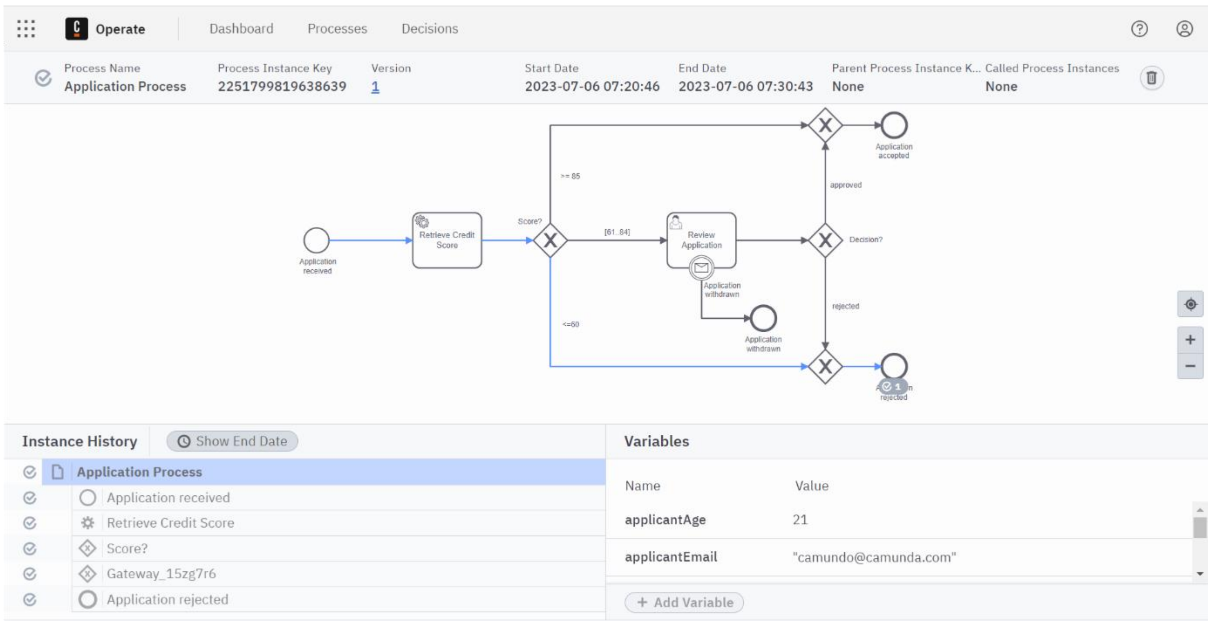
```
zbctl complete job [jobKey] --variables "{\"score\":60}"
```

If the job completion was successful, you should see output similar to the below:

```
Completed job with key '2251799813685529' and variables '{"score":60}'
```

View the Process Instance in Operate

If you now launch Operate, you will see that the Process Instance that you launched is now showing as complete and that the application was rejected..



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance K...	Called Process Instances
Application Process	2251799819638639	1	2023-07-06 07:20:46	2023-07-06 07:30:43	None	None

Instance History

- Application Process
  - Application received
  - Retrieve Credit Score
  - Score?
  - Gateway_15zg7r6
  - Application rejected

Variables

Name	Value
applicantAge	21
applicantEmail	"camundo@camunda.com"

Failed a Job

Congratulations! You have successfully failed a Job using `zbctl`

Create Worker

In the last section, you saw how `zbctl` can be used to activate, complete and fail jobs that are created by Zeebe.
However, in practice it would be unusual for a Developer or an Administrator to handle the low level lifecycle of a **Job** as Camunda provides several [Official Zeebe Clients] which provide a high level abstractions over the [Zeebe API] and other utility / convenience functions that you would expect to find in a client library.

We will now use `zbctl` to create a **Job Worker** which will process **Jobs** from the *Retrieve Credit Score* Service Task. This **Job Worker** will combine the `activate` and `complete` commands from the previous lessons into a single worker.

Create a Worker

In the earlier **Outline Solution** lesson we saw that the *Retrieve Credit Score* Service Task needs to return a Credit Score between 0 and 100.

This time, instead of issuing low level commands (`activate`, `complete` and `fail`) using `zbctl`, we will create a **Job Worker** which will complete the Service Task instead.

Create a Process Instance

We can start by creating a new process instance using the following command:

```
zbctl create instance ApplicationProcess --variables
"{\"applicationId\":\"Application-123\",\"applicantName\":\"Camundo
Camunda\",\"applicantEmail\":\"camundo@camunda.com\",\"applicantAge\":21}"
```

If the instance creation was successful, you should see output similar to the below:

```
{
  "processDefinitionKey": "2251799813685249",
  "bpmnProcessId": "ApplicationProcess",
  "version": 1,
  "processInstanceKey": "2251799813685540"
}
```

Create a Job Worker for Retrieve Credit Score

We will now create a **Job Worker** which will always return a score of 80 to the process instance by issuing the following command:

```
zbctl create worker creditscore --handler "echo {\"score\":80}"
```

If the worker creation was successful, you should see output similar to the below:

```
2023/09/13 16:54:13 Activated job 2251799814701216 with variables
{"applicantAge":21,"applicantName":"Camundo
Camunda","applicationId":"Application-
123","applicantEmail":"camundo@camunda.com"}
2023/09/13 16:54:13 Handler completed job 2251799814701216 with variables
{"score":80}
```

You can see that the **Job** was acquired and completed by the **Job Worker** without any need to issue the low level `activate`, `complete` and `fail` commands.

What just happened?

The `zbctl create worker` command created a polling **Job Worker** which calls the provided handler for every **Job**. The handler will receive the variables of the **Activated Job** as a JSON object. If the handler completes successfully, the **Job** will be completed with the variables provided. Otherwise, if the handler exists with a non-zero exit code the **Job** will be failed.

Windows Command Line

If you are using the Windows Command Line you may need to take a different approach to implement the handler. An example has been provided below:

Create a `creditScore.bat` file which contains the following JSON:

```
@echo off
echo {"score":80}
```

Create the **Job Worker** using:

```
zbctl create worker creditscore --handler .\creditScore.bat
```

Configuring the Job Worker

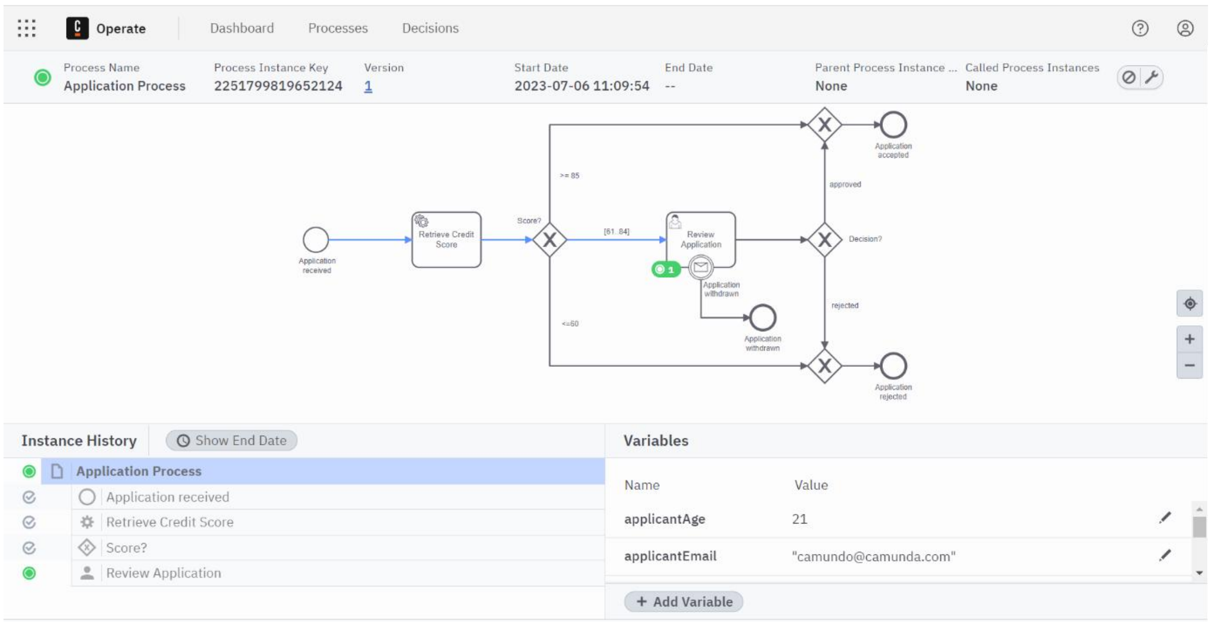
Further information on the parameters that are most commonly used in conjunction with creating a **Job Worker** can be found below.

Name	**Description**
concurrency	Specify the maximum number of concurrent spawned go routines to complete jobs
handler	Specify handler to invoke for each job. The handler will then complete the job with its result or fail the job if it encounters a problem while processing the job

Name	Description
maxJobsActive	Specify the maximum number of jobs which will be activated for this worker at the same time
name	Specify the worker name
pollInterval	Specify the maximal interval between polling for new jobs
pollThreshold	Specify the threshold of buffered activated jobs before polling for new jobs, i.e. pollThreshold * maxJobsActive
requestTimeout	Specify the timeout for a request
timeout	Specify the duration no other worker should work on job activated by this worker

Verify in Operate

If you now launch Operate, you will see that the Process Instance that you launched has now moved to the **Review Application** task.



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance ...	Called Process Instances
Application Process	2251799819652124	1	2023-07-06 11:09:54	--	None	None

Instance History:
- Application Process
  - Application received
  - Retrieve Credit Score
  - Score?
  - Review Application

Variables

Name	Value
applicantAge	21
applicantEmail	"camundo@camunda.com"

Complete the User Task

Since the User Task **Review Application** is currently active, it must be completed.

A Job Worker can subscribe to the Job Type **io.camunda.zeebe:userTask** to complete the Job manually. When the Job is completed, the User Task is completed and the process instance continues.

```
zbctl create worker io.camunda.zeebe:userTask --handler "echo
{\"approved\":true}"
```

If the worker creation was successful, you should see output similar to the below:

```
2023/09/13 18:12:45 Activated job 2251799814704618 with variables
{"applicantAge":21,"applicantName":"Camundo
Camunda","applicationId":"Application-
123","applicantEmail":"camundo@camunda.com"}
2023/09/13 18:12:45 Handler completed job 2251799814704618 with variables
{"approved":true}
```

Windows Command Line

If you are using the Windows Command Line you may need to take a different approach to implement the handler. An example has been provided below:

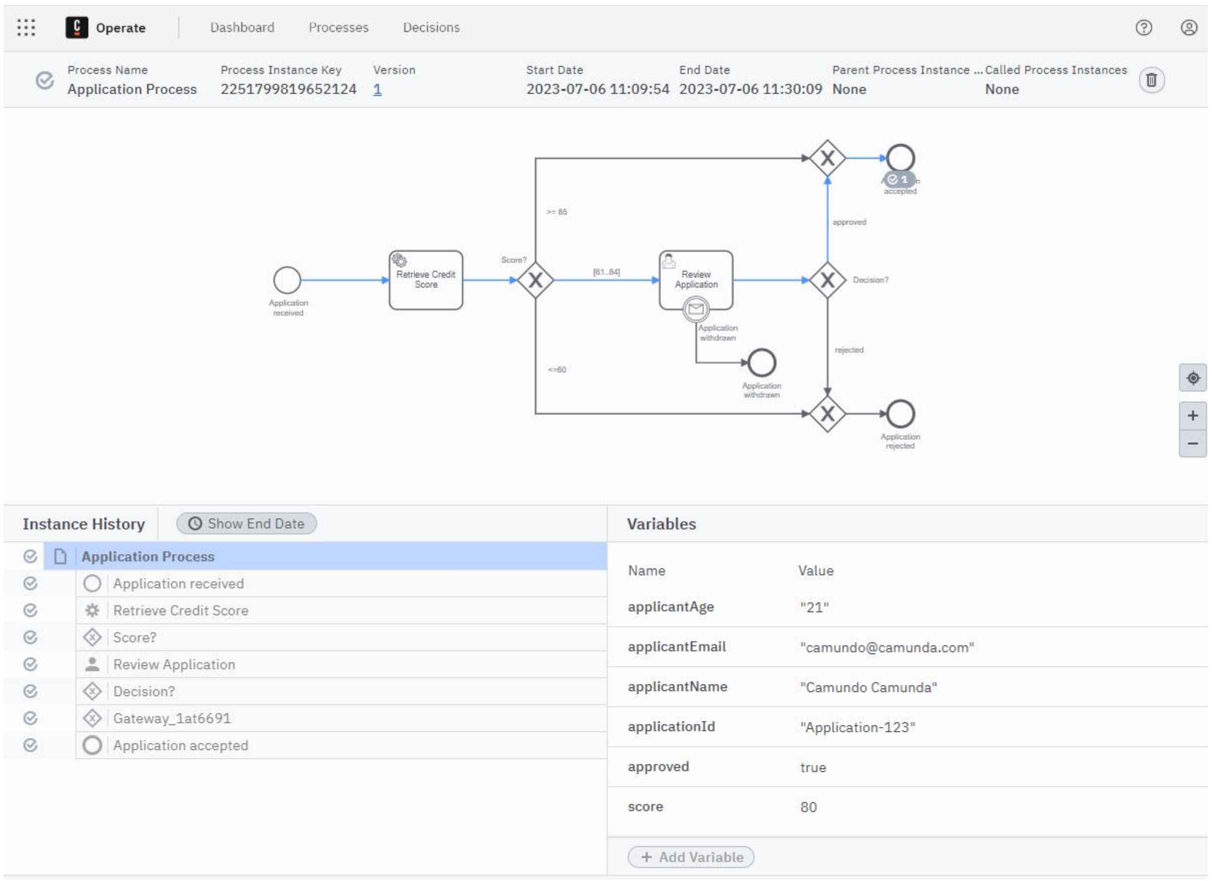
Create a `reviewApplication.bat` file which contains the following JSON:

```
@echo off
echo {"approved":true}
```

Create the **Job Worker** using:

```
zbctl create worker io.camunda.zeebe:userTask --
handler .\reviewApplication.bat
```

Finally, in Operate, you can see that the process instance has now completed.



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance ...	Called Process Instances
Application Process	2251799819652124	1	2023-07-06 11:09:54	2023-07-06 11:30:09	None	None

Instance History

- Application Process
  - Application received
  - Retrieve Credit Score
  - Score?
  - Review Application
  - Decision?
  - Gateway_1at6691
  - Application accepted

Variables

Name	Value
applicantAge	"21"
applicantEmail	"camundo@camunda.com"
applicantName	"Camundo Camunda"
applicationId	"Application-123"
approved	true
score	80

Created a Worker

Congratulations! You have successfully used `zbctl` to create a **Job Worker** and to also complete the *Review Application* User Task.

Managing Messages

Publish Message

You should now be able to perform the following tasks using `zbctl`:

- Deploy a Process Definition
- Start a Process Instance
- Cancel a Process Instance
- Activate, Complete and Fail a Job
- Create a Job Worker

In this final lesson, you will see how to publish a message using the **PublishMessage RPC**. This will allow an application in manual processing to be withdrawn.

What is Message Correlation?

Process instances can be notified of incoming messages through Start, Boundary and Intermediate Events. However, the message must be mapped to the correct process instance using a process called **Message Correlation**.

A message is not sent to a process instance directly. Instead, the message correlation is based on subscriptions that contain the `message name` and the `correlation key` (also known as the correlation value).

Prepare a Process Instance

Start a Process Instance

Start a new process instance using the following command:

```
zbctl create instance ApplicationProcess --variables
"{\"applicationId\":\"Application-001\",\"applicantName\":\"Camundo
Camunda\",\"applicantEmail\":\"camundo@camunda.com\",\"applicantAge\":21}"
```

If the instance creation was successful, you should see output similar to the below:

```
{
  "processDefinitionKey": "2251799813685249",
  "bpmnProcessId": "ApplicationProcess",
  "version": 1,
  "processInstanceKey": "2251799813685254"
}
```

Correlation Key

We will be using the Application Id as a **Correlation Key** so you should ensure that this value is unique when starting the process instance i.e. you will need to increment the Application Id if you work through this lesson several times.

Start the Job Worker

Start a **Job Worker** which will always return a score of 80 to the process instance by issuing the following command:

```
zbctl create worker creditscore --handler "echo {\"score\":80}"
```

If the worker creation was successful, you should see output similar to the below:

```
2023/09/13 16:54:13 Activated job 2251799814701216 with variables
{"applicantAge":21,"applicantName":"Camundo
Camunda","applicationId":"Application-
001","applicantEmail":"camundo@camunda.com"}
2023/09/13 16:54:13 Handler completed job 2251799814701216 with variables
{"score":80}
```

Publish a Message

We can now publish a **Message** to Zeebe using the **applicationId** as a **Correlation Key**.

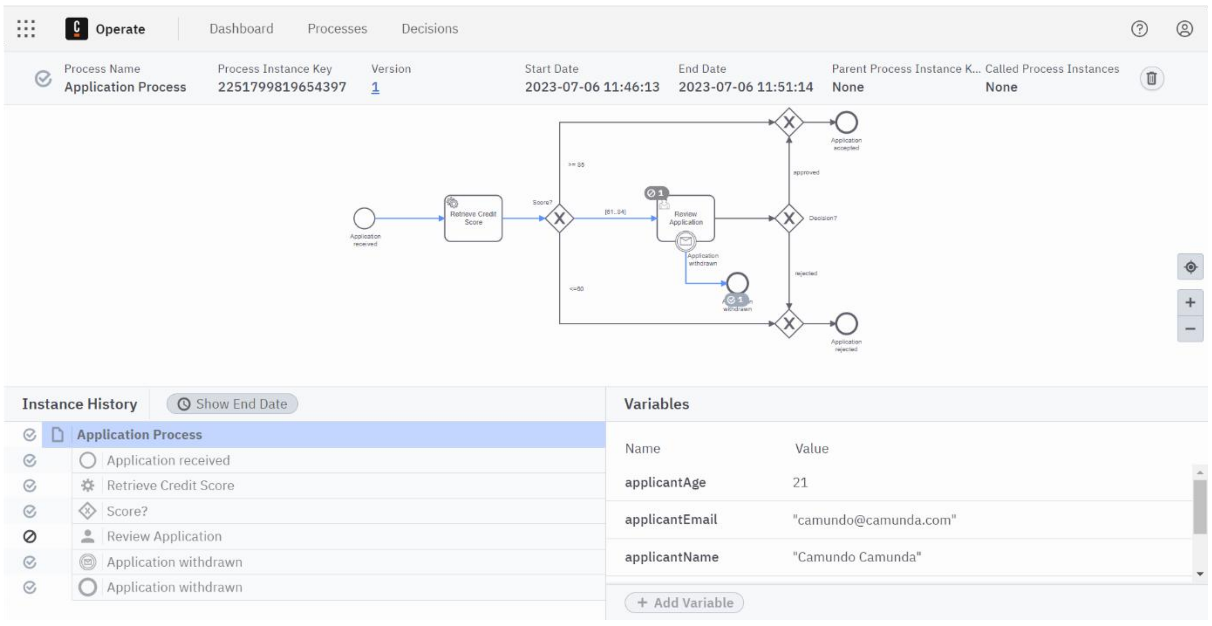
```
zbctl publish message "Application withdrawn" --correlationKey "Application-
001"
```

If the publication was successful, you should see output similar to the below:

```
{
  "key": "2251799813685665"
}
```

Verify in Operate

Finally, in Operate, you can see that the User Task has been cancelled and that the process is now completed at **Application Withdrawn**.



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance K...	Called Process Instances
Application Process	2251799819654397	1	2023-07-06 11:46:13	2023-07-06 11:51:14	None	None

Instance History

- Application Process
  - Application received
  - Retrieve Credit Score
  - Score?
  - Review Application
  - Application withdrawn
  - Application withdrawn

Variables

Name	Value
applicantAge	21
applicantEmail	"camundo@camunda.com"
applicantName	"Camundo Camunda"

Published a Message

Congratulations! You have successfully published a **Message** using `zbctl`

Cleanup Environment

We have now met all of the business requirements that have been defined in our **Scenario**. We are now in a position where we can perform an end-to-end test of our process to ensure that everything is working as expected.

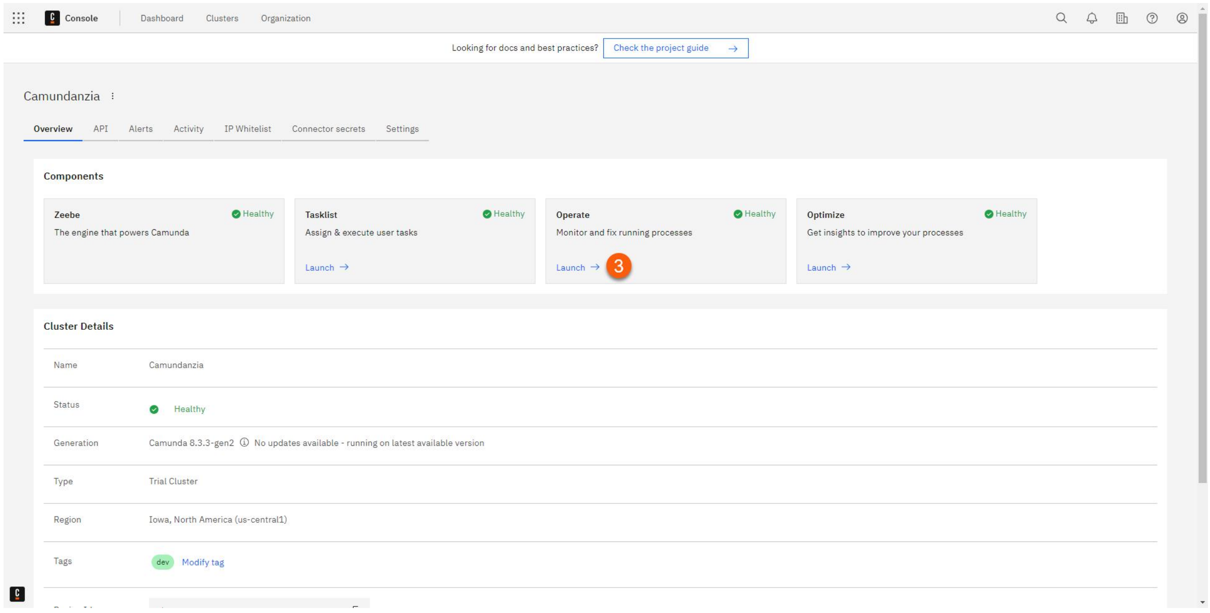
However, as we have been testing our process definition using `zbctl` it is likely that we have several incomplete process instances still running. Having incomplete (and possibly incorrect) process instances running in our environment may make it more difficult to confirm that everything is working as expected in an end-to-end test.
We will now **Cancel** and **Delete** any remaining instances so that we have a completely clean environment in which we can conduct our acceptance tests.

Cancel & Delete Process Instances

Launch Operate

Launch **Operate** by following the steps below:

1. From the **Console**, select **Clusters**
2. Select *Camundanzia* from the **Clusters** page
3. Click the **Launch** link next to **Operate** in the **Cluster Details** page



Zeebe	Healthy	The engine that powers Camunda
Tasklist	Healthy	Assign & execute user tasks
Operate	Healthy	Monitor and fix running processes
Optimize	Healthy	Get insights to improve your processes

Cluster Details

Name	Camundanzia
Status	Healthy
Generation	Camunda 8.3.3-gen2 ⓘ No updates available - running on latest available version
Type	Trial Cluster
Region	Iowa, North America (us-central1)
Tags	dev Modify tag

4. After a few seconds, the **Operate** Dashboard will be displayed

Cancel Process Instances

We can now **Cancel** the process instances in **Operate** by following the steps below:

1. Click on *Application Process* from the **Instances by Process** panel on the left-hand side of the screen
2. On the bottom left of the screen is a series of filters that will allow you to select *Running* or *Finished* process instances

3. Ensure that the *Running* filters are selected and the *Finished* filters are not selected
4. Then, for each **Process Instance** that is shown in the **Instances** panel, click the **Cancel Instance** button to the right-hand side of the process

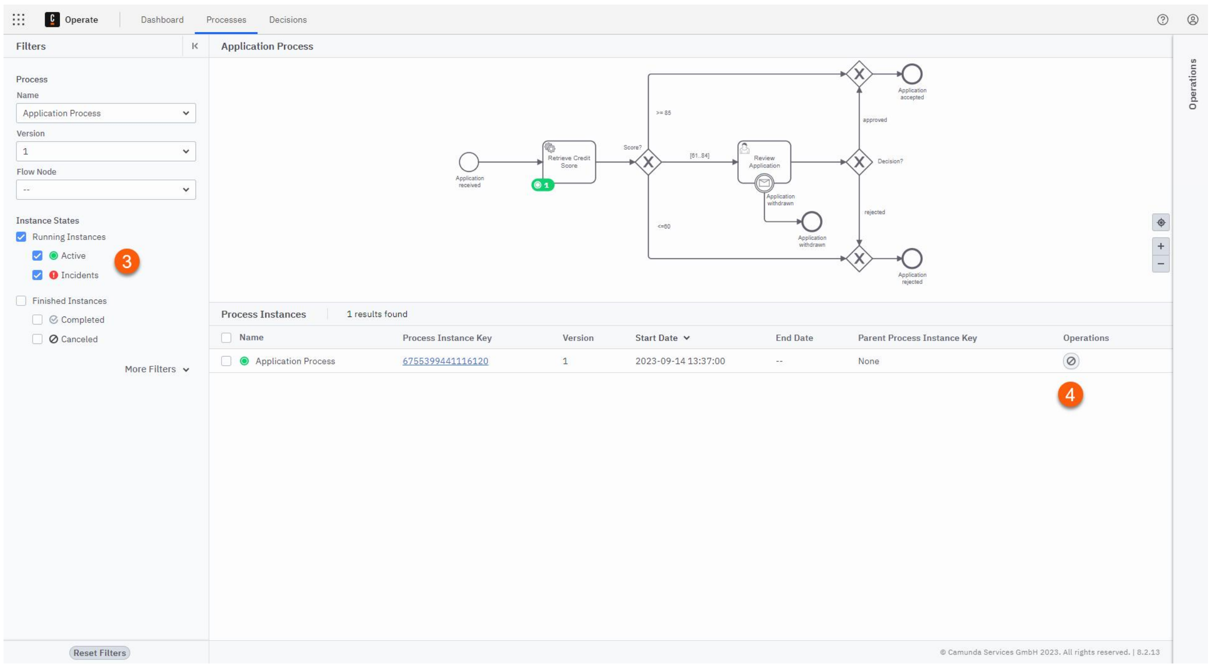
No Running Processes

This section can be skipped if there are no running processes shown in the **Instances** panel.

5. The **Process Instance** will then be cancelled and will disappear from the **Instances** panel

What happened to the Process Instance?

When you **Cancel** a process in Operate it changes to a stopped state. As the process is no longer running it is not picked up by *Running* filter.


Filters

Process
Name: Application Process
Version: 1
Flow Node: --

Instance States
- [✓] Running Instances
  - [✓] Active
  - [✓] Incidents
- [ ] Finished Instances
  - [ ] Completed
  - [ ] Canceled

Application Process

Process Instances — 1 results found

Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance Key	Operations
Application Process	6755399441116120	1	2023-09-14 13:37:00	--	None	⊘

© Camunda Services GmbH 2023. All rights reserved. | 8.2.13

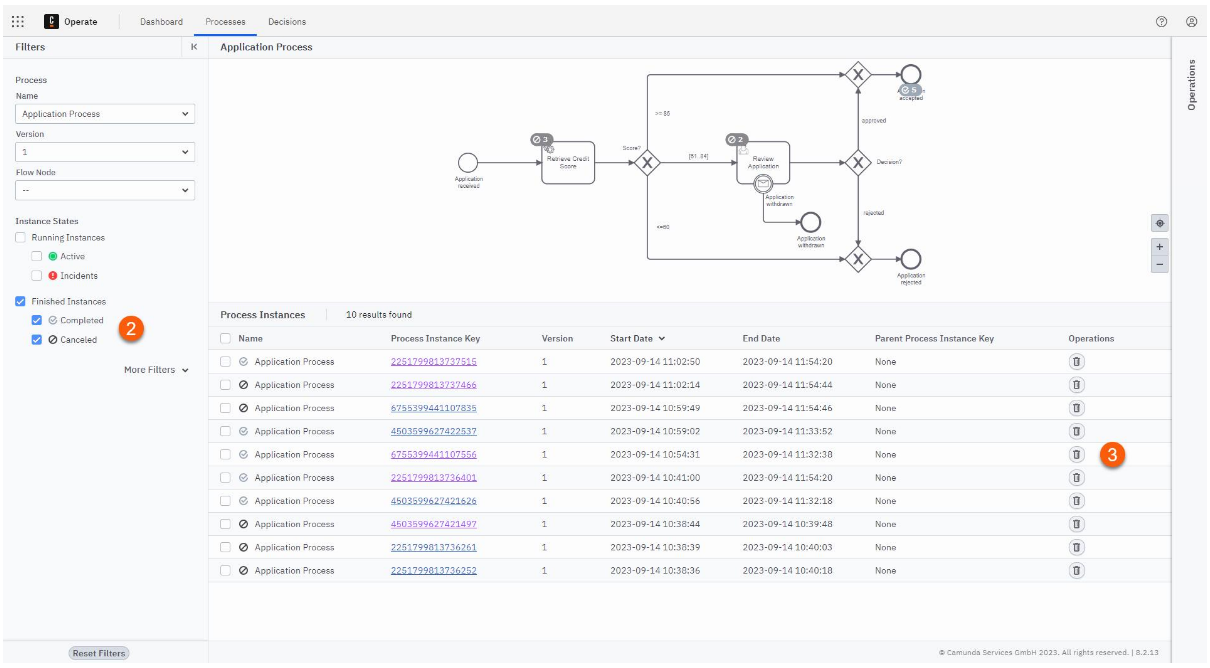
Delete Process Instances

We can now **Delete** the finished process instances in **Operate** by following the steps below:

1. On the bottom left of the screen is a series of filters that will allow you to select *Running* or *Finished* process instances
2. Ensure that the *Finished* filters are selected and the *Running* filters are not selected
3. Then, for each **Process Instance** that is shown in the **Instances** panel, click the **Delete Instance** button to the right-hand side of the process
4. The **Process Instance** will then be deleted and will disappear from the **Instances** panel

What happened to the Process Instance?

When you **Delete** a process in Operate it is completely removed from the **Zeebe Engine**.



Filters

Process
Name: Application Process
Version: 1
Flow Node: --

Instance States
- [ ] Running Instances
  - [ ] Active
  - [ ] Incidents
- [x] Finished Instances
  - [x] Completed
  - [x] Canceled

More Filters

Application Process

Process Instances — 10 results found

Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance Key	Operations
✓ Application Process	2251799813737515	1	2023-09-14 11:02:50	2023-09-14 11:54:20	None	🗑
⊘ Application Process	2251799813737466	1	2023-09-14 11:02:14	2023-09-14 11:54:44	None	🗑
⊘ Application Process	6755399441107835	1	2023-09-14 10:59:49	2023-09-14 11:54:46	None	🗑
✓ Application Process	4503599627422537	1	2023-09-14 10:59:02	2023-09-14 11:33:52	None	🗑
✓ Application Process	6755399441107556	1	2023-09-14 10:54:31	2023-09-14 11:32:38	None	🗑
✓ Application Process	2251799813736401	1	2023-09-14 10:41:00	2023-09-14 11:54:20	None	🗑
✓ Application Process	4503599627421626	1	2023-09-14 10:40:56	2023-09-14 11:32:18	None	🗑
⊘ Application Process	4503599627421497	1	2023-09-14 10:38:44	2023-09-14 10:39:48	None	🗑
⊘ Application Process	2251799813736261	1	2023-09-14 10:38:39	2023-09-14 10:40:03	None	🗑
⊘ Application Process	2251799813736252	1	2023-09-14 10:38:36	2023-09-14 10:40:18	None	🗑

Cleaned the Environment

Congratulations! You have now removed all instances of *Application Process* from the environment.

Test Process

We are now ready to test the *Application Process* end-to-end.

Review Acceptance Criteria

Let's start by reviewing the acceptance criteria that was defined earlier; we need to ensure that all of the requirements below can be met by our modifications and understanding.

1. An **Applicant** is able to submit a *Car Insurance Application*
2. The *Credit Score* of the **Applicant** can be retrieved automatically
3. A **Clerk** is able to review complex applications and decide whether the **Applicant** will be accepted or not
4. An **Applicant** is able to cancel their application

Test Scenarios

The following scenarios will be required to fully confirm that the acceptance criteria above has been met.

Test Application

Test Scenarios

Five scenarios will be required to confirm that the acceptance criteria above has been met.

Test Scenario	Application Id	Applicant Name	Applicant Email	Applicant Age	Score	Expected Result
1 - Application Accepted	CMD000001	Accepted	001@camunda.com	21	90	Accepted
2 - Application Reviewed & Approved	CMD000002	Accepted	002@camunda.com	22	80	Accepted
3 - Application Reviewed & Rejected	CMD000003	Rejected	003@camunda.com	23	80	Rejected
4 - Application Rejected	CMD000004	Rejected	004@camunda.com	24	50	Rejected
5 - Application Withdrawn	CMD000005	Withdrawn	005@camunda.com	25	80	Withdrawn

Launch Process Instances

Start five instances of the *Application Process* by issuing the following commands:

```
zbctl create instance ApplicationProcess --variables
"{\"applicationId\":\"CMD0000001\",\"applicantName\":\"Accepted\",\"applicantE
mail\":\"001@camunda.com\",\"applicantAge\":21,\"score\":90}"
zbctl create instance ApplicationProcess --variables
"{\"applicationId\":\"CMD0000002\",\"applicantName\":\"Accepted\",\"applicantE
mail\":\"002@camunda.com\",\"applicantAge\":22,\"score\":80,\"approved\":true}
"
```

```
zbctl create instance ApplicationProcess --variables
"{\"applicationId\":\"CMD0000003\",\"applicantName\":\"Rejected\",\"applicantE
mail\":\"003@camunda.com\",\"applicantAge\":23,\"score\":80,\"approved\":false}
"
zbctl create instance ApplicationProcess --variables
"{\"applicationId\":\"CMD0000004\",\"applicantName\":\"Rejected\",\"applicantE
mail\":\"004@camunda.com\",\"applicantAge\":24,\"score\":50}"
zbctl create instance ApplicationProcess --variables
"{\"applicationId\":\"CMD0000005\",\"applicantName\":\"Withdrawn\",\"applicant
Email\":\"005@camunda.com\",\"applicantAge\":25,\"score\":80}"
```

Process Variables

In order to simplify our testing, we will populate all process variables when each new instance is started. In the real-world, a **Job Worker** would be developed that encapsulates the *Retrieve Credit Score* business logic; a **Clerk** would also be responsible for making the decision on whether the application has been approved or rejected.

Launch Job Worker for Retrieve Credit Score

Launch a **Job Worker** to complete the *Retrieve Credit Score* tasks by issuing the following command:

```
zbctl create worker creditscore --handler "echo"
```

Echo?

The `--handler` is a required parameter for creating a new worker; we have used "echo" as the simplest possible implementation here as we have already pre-populated the correct variables on each process instance.

If the worker creation was successful, you should see output similar to the below:

```
2023/09/15 08:46:33 Activated job 2251799813808253 with variables
{"score":80,"applicantAge":25,"applicantName":"Withdrawn","applicationId":"CMD
0000005","applicantEmail":"005@camunda.com"}
2023/09/15 08:46:33 Activated job 2251799813808227 with variables
{"score":80,"approved":false,"applicantAge":23,"applicantName":"Rejected","app
licationId":"CMD0000003","applicantEmail":"003@camunda.com"}
2023/09/15 08:46:33 Activated job 4503599627493422 with variables
{"score":50,"applicantAge":24,"applicantName":"Withdrawn","applicationId":"CMD
0000004","applicantEmail":"004@camunda.com"}
2023/09/15 08:46:33 Activated job 6755399441178676 with variables
{"score":90,"applicantAge":21,"applicantName":"Accepted","applicationId":"CMD0
000001","applicantEmail":"001@camunda.com"}
2023/09/15 08:46:33 Handler completed job 6755399441178676 with variables {}
2023/09/15 08:46:33 Activated job 6755399441178688 with variables
{"score":80,"approved":true,"applicantAge":22,"applicantName":"Accepted","appl
icationId":"CMD0000002","applicantEmail":"002@camunda.com"}
2023/09/15 08:46:33 Handler completed job 4503599627493422 with variables {}
2023/09/15 08:46:33 Handler completed job 2251799813808227 with variables {}
2023/09/15 08:46:33 Handler completed job 2251799813808253 with variables {}
```

```
2023/09/15 08:46:34 Handler completed job 6755399441178688 with variables {}
```

Publish a Message

Publish a Message to Zeebe using the applicationId as a Correlation Key.

```
zbctl publish message "Application withdrawn" --correlationKey "CMD0000005"
```

If the publication was successful, you should see output similar to the below:

```
{
  "key": "2251799813685665"
}
```

Complete the Review Application User Task

Launch a **Job Worker** to complete the *Review Application* tasks by issuing the following command:

```
zbctl create worker io.camunda.zeebe:userTask --handler "echo"
```

Echo?

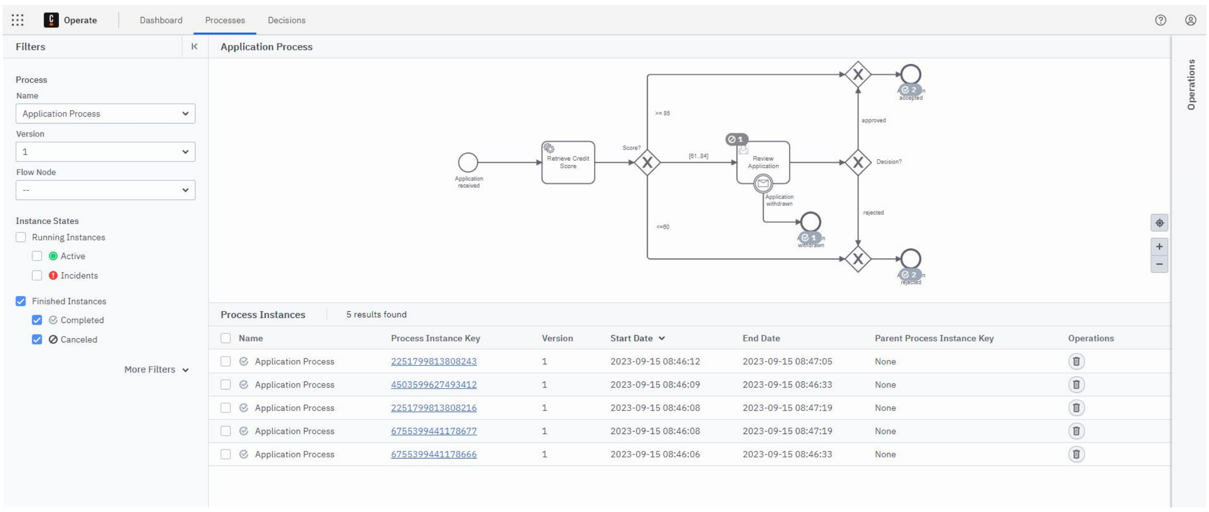
The `--handler` is a required parameter for creating a new worker; we have used "echo" as the simplest possible implementation here as we have already pre-populated the correct variables on each process instance.
If the worker creation was successful, you should see output similar to the below:

```
2023/09/15 08:47:19 Activated job 2251799813808276 with variables
{"score":80,"approved":false,"applicantAge":23,"applicantName":"Rejected","app
licationId":"CMD0000003","applicantEmail":"003@camunda.com"}
2023/09/15 08:47:19 Activated job 6755399441178734 with variables
{"score":80,"approved":true,"applicantAge":22,"applicantName":"Accepted","appl
icationId":"CMD0000002","applicantEmail":"002@camunda.com"}
2023/09/15 08:47:19 Handler completed job 2251799813808276 with variables {}
2023/09/15 08:47:19 Handler completed job 6755399441178734 with variables {}
```

Review Test Scenarios

We can now review the results from the test scenarios to confirm that everything is working as expected.

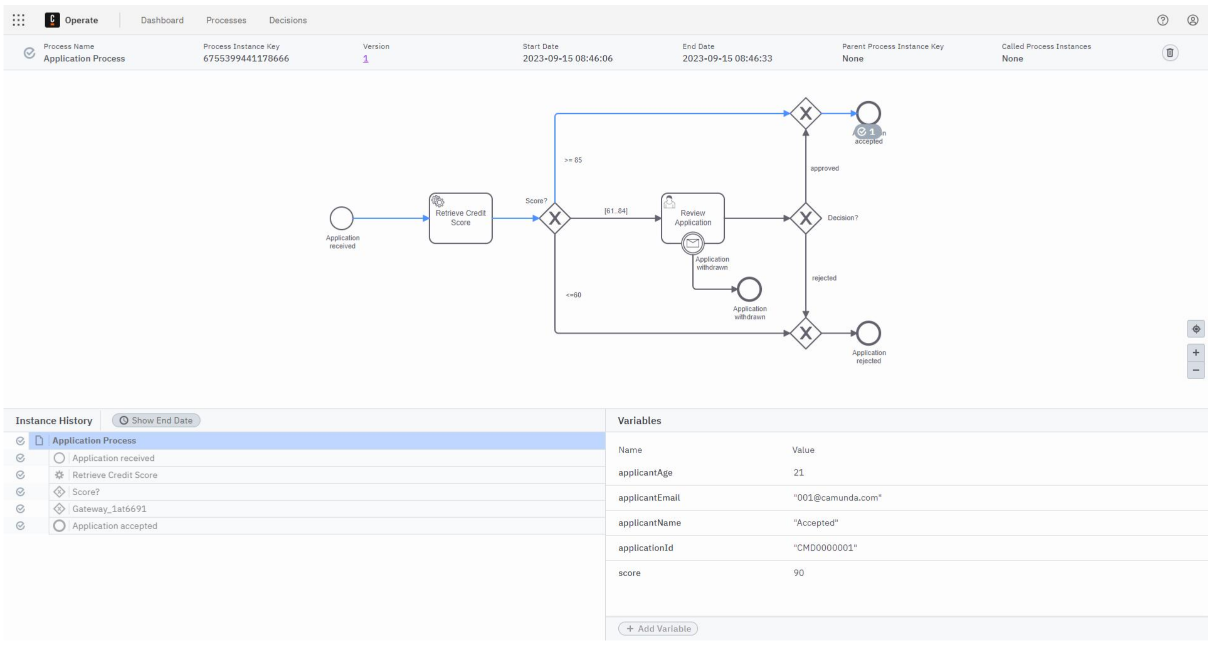
1. Launch **Operate** by following the steps in the previous lesson. Alternatively, refresh the window if you still have **Operate** open.
2. Click on *Application Process* from the **Process Instances by Name** panel on the left-hand side of the screen
3. Check the **Finished Instances** filter on the left hand side of the screen to show completed process instances. If everything has worked as expected you should now see five completed process instances on the screen.



Process Instances | 5 results found

Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance Key	Operations
Application Process	2251799813808243	1	2023-09-15 08:46:12	2023-09-15 08:47:05	None	
Application Process	4503599627493412	1	2023-09-15 08:46:09	2023-09-15 08:46:33	None	
Application Process	2251799813808216	1	2023-09-15 08:46:08	2023-09-15 08:47:19	None	
Application Process	6755399441178677	1	2023-09-15 08:46:08	2023-09-15 08:47:19	None	
Application Process	6755399441178666	1	2023-09-15 08:46:06	2023-09-15 08:46:33	None	

4. Click on each of the **Process Instance Keys** in the bottom part of the screen to open up the **Process Instance View**
5. You should now be able to confirm that each of the scenarios above has been executed successfully.



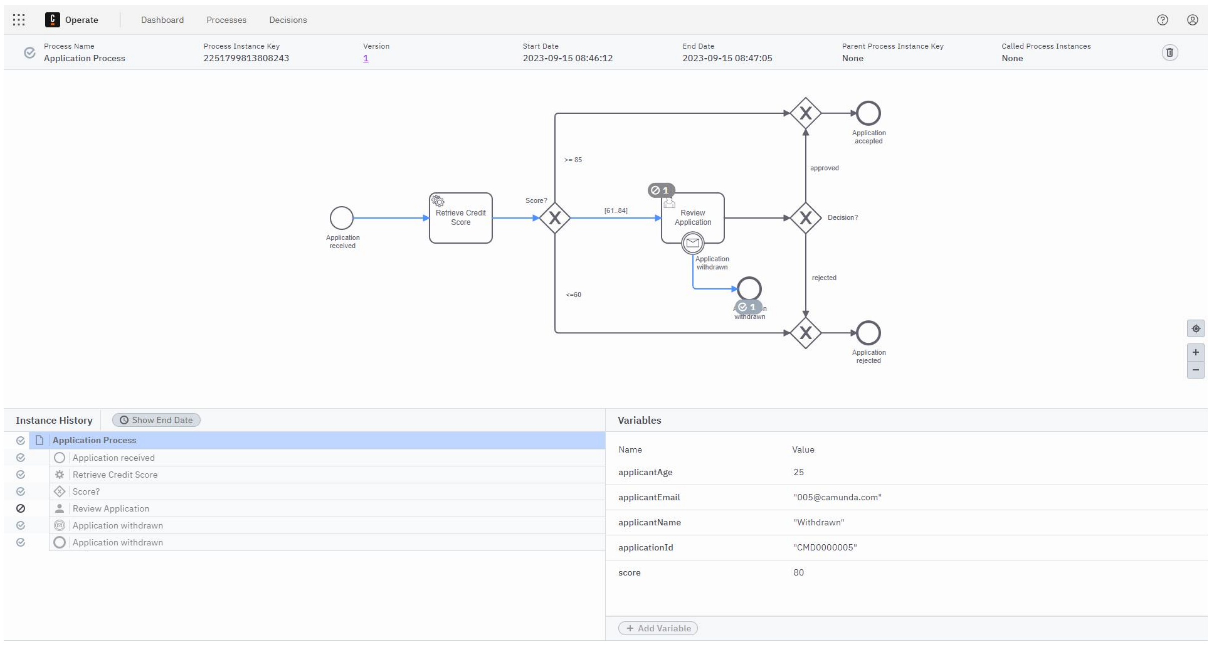
Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance Key	Called Process Instances
Application Process	6755399441178666	1	2023-09-15 08:46:06	2023-09-15 08:46:33	None	None

Instance History

- Application Process
  - Application received
  - Retrieve Credit Score
  - Score?
  - Gateway_1at6691
  - Application accepted

Variables

Name	Value
applicantAge	21
applicantEmail	"001@camunda.com"
applicantName	"Accepted"
applicationId	"CMD0000001"
score	90



Process Name	Process Instance Key	Version	Start Date	End Date	Parent Process Instance Key	Called Process Instances
Application Process	2251799813808243	1	2023-09-15 08:46:12	2023-09-15 08:47:05	None	None

Instance History

- Application Process
  - Application received
  - Retrieve Credit Score
  - Score?
  - Review Application
  - Application withdrawn
  - Application withdrawn

Variables

Name	Value
applicantAge	25
applicantEmail	"005@camunda.com"
applicantName	"Withdrawn"
applicationId	"CMD0000005"
score	80

Tested Car Insurance Request

Congratulations! You have now confirmed that the *Application Process* meets the acceptance criteria which was specified at the start of the course.