User Scenario: Creating a general workflow for change requests

We want to implement a user scenario requiring a workflow for a general change requests. For example, an end user should be enabled to launch a request to change or improve a business related issue such as process, a convention or any other business related artifact. For this we establish a general change request workflow that allows the end user to open a request, discuss the issue with others and then send the request for approval to the management.

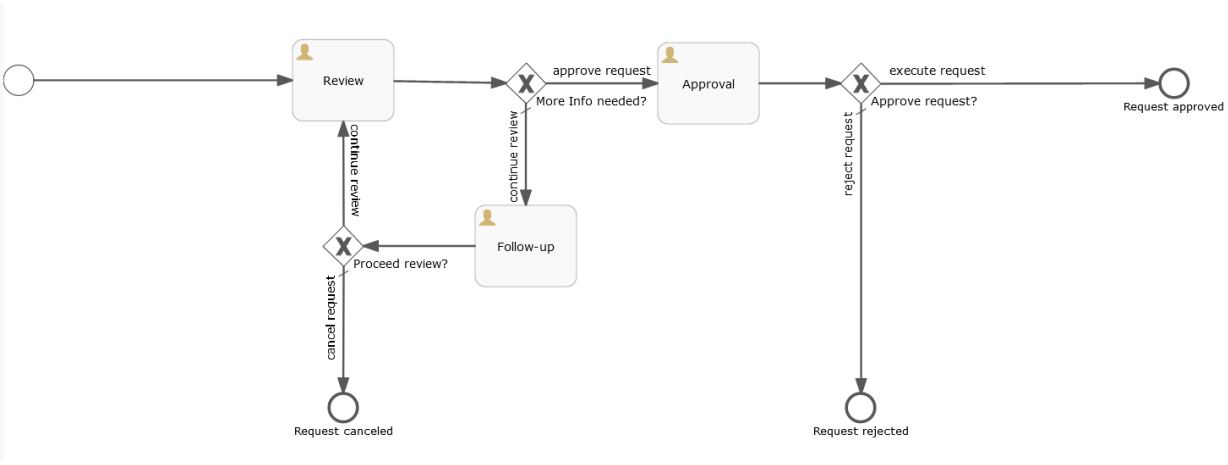
For our scenario we construct a workflow template from scratch. We intend to depend on use as few JavaScript code as possible in order to keep it simple. In a first approach there will be no JavaScript code whatsoever. As a consequence, we will neither be able to update the workflow's state display during the workflow execution nor to log the progress of the workflow. In a second approach, we use some JavaScript code in order to update the workflow's state display during the workflow execution and add the workflow's progress to the logfile.

We use three user task elements that process the user data and three gateways that handle the control workflow. The start element will prompt for the end user's input data. At the end of the workflow, we know whether the user's change request was successful and will be implemented by the company later or whether it was given up.

The end user's change request consists of an initial title, a short summary and a detailed description.

First approach with plain user task elements

As a prerequisite, start your workflow visual editor and create a new process called "General change request" and assign it a new unique ID/key.



- Start the workflow by adding a start element and give it a unique ID, e.g. "start"
    - As Form properties, add four variables of type String:
        - Id "title" with name "Request title"
        - Id "summary" with name "Summary"
        - Id "description" with name "Description"
        - Id "category" with name "Category"
          For this variable add the following term as 'Expression':
          `${cpd:conf('{"cpd_type":"wkc_category","kind":"trigger"}')}`

- Make sure that all of these variables have the “Required” and the “Write” checks enabled

- Next, create a new user task element and give it a unique ID, e.g. “review” and name it “Review”
  - As Form properties, add the following variables of type String:
    - Id “title” with name “Request title”
      Make sure to have the “Read” check enabled and the “Write” and “Required” checks disabled
    - Id “summary” with name “Summary”
      Make sure to have the “Read” and the “Write” check enabled and the “Required” check disabled
    - Id “description” with name “Description”
      Make sure to have the “Read” and the “Write” check enabled and the “Required” check disabled
    - Id “category” with name “Category”
      Make sure to have “Read” check enabled and the “Write” and “Required” check disabled
    - Id “comment” with name “Comment”
      Make sure to have the “Read” and the “Write” check enabled and the “Required” check disabled
  - As Form properties, add the following variable of type “enum”:
    - Id “action” with the following enum values:
      - Id “completed” with the name “Review completed” that will serve as the end user’s corresponding button name.
      - Id “-moreinfo” with the name “More Info needed” that will serve as the end user’s corresponding button name.
      - Make sure to have the “Required”, “Read”, and “Write” check enabled

  - Due to a restriction in the BPMN format, the “Documentation” property will be used to define a set of values that are picked up by Cloud Pak for Data in order to visualize the workflow for the end user during the workflow’s execution.
    All these different values will have to be stored in the “Documentation” property of the user task element as one string where each value is separated by a “$$$” sequence from the next value. The values that must be set are:
    - The task title: The title that will be shown to the end user during the workflow execution, e.g. “*Review ${artifact_name}*”
    - The task instructions: A set of instructions that will be shown to the end user for this task during the workflow execution, e.g. “*Please review this change request and then either send it to approval or continue the review*”
    - The step instructions: A set of instructions that will be shown to the workflow administrator for this workflow task during the workflow configuration, e.g. “*Add at least one assignee for review discussions of this request”*
    - The step title: The title that will be shown to the workflow administrator for this workflow task during the workflow configuration, e.g. “*Review ${artifact_name}”*

- Using the textual examples from above, the complete textual value for the "Documentation" property of this new user task element would be:
  "Review ${artifact_name}$$$Please review this change request and then either send it to approval or continue the review discussion$$$Add at least one assignee for review discussions of this request$$$Review ${artifact_name}"

- Next, create another user task element "Approval" by copying the newly created "Review" user task. This will ensure that the new user task "Approval" will have the same properties as the base "Review" user task and save you a lot of typing.
  - Make sure the new user task "Approval" has a unique ID (e.g. "approval")
  - Check that the new user task "Approval" has the same set of Form property variables as the "Review" user task, which served as a base for copying.
  - As the "Form" property, edit the variable "action" of type "enum" and change its values to these two:
    - Id "completed" with the name "Review completed" that will serve as the end user's corresponding button name.
    - Id "-moreinfo" with the name "More info needed" that will serve as the end user's corresponding button name.
      Note the minus character as part of the enum's ID. This ensures that once this action is chosen, the end user is required to add a justification comment.
  - Edit the "Documentation" property of the "Approval" user task and assign it new titles and descriptions, e.g.:
    "Approve ${artifact_name}$$$Please either approve or reject this change request$$$Add at least one assignee for an approval decision of this change request$$$Approve ${artifact_name}"

- Next, create another user task element "Follow-up" by copying the "Review" user task. This will ensure that the new user task "Follow-up" will have the same properties as the base "Review" user task and save you a lot of typing.
  - Make sure the new user task "Follow-up" has a unique ID (e.g. "follow_up")
  - Check that the new user task "Follow-up" has the same set of Form property variables as the "Review" user task, which served as a base for copying.
  - As the "Form" property, edit the variable "action" of type "enum" and change its values to these two new values:
    - Id "continue_review" with the name "Continue review" that will serve as the end user's corresponding button name.
    - Id "-discard" with the name "Discard change request" that will serve as the end user's corresponding button name.
      Note the minus character as part of the enum's ID. This ensures that once this action is chosen, the end user is required to add a justification comment.
  - Edit the "Documentation" property of the "Follow-up" user task and assign it new titles and descriptions, e.g.:
    "Continue review ${artifact_name}$$$Please either continue to review and discuss this change request or cancel the request all together$$$Add at least one assignee for a the continued review of this change request$$$Follow-up ${artifact_name}"

Note that for the “Documentation” properties of all these three user task elements above the “$$$” sequence is separating the four individual values from each other and that the sequence of the individual values is always fixed and that values must be specified according to the sequence above (1. task title, 2. task instructions, 3. step instructions, 4. step title).
Also note, that you can use variables within these values. Variables will then be substituted by their corresponding value during the execution of the workflow. The variable ${artifact_name} will be replaced by the name of the artifact in progress during the workflow execution. A variable term must always have the form “${varname}” where “varname” is the name of the variable. The complete variable term will be replaced by the actual value for the variable “varname” during the execution of the workflow. For a complete list of all Variables, see list above.

As the next step we need to add the exclusive gateways. One exclusive gateway (“More info needed?”) will control the review and discuss cycle of the change request and one exclusive gateway (“Approved request?”) will either end the change request successfully or reject it. During the review and discussion cycle, a further exclusive gateway (“Proceed with review?”) will allow the end users to continue the cycle or cancel the change request all together.

As the next step we need to add the end events. You can create multiple end events as the end point of the flows out of the exclusive gateways or you can create one single end event and attach all exclusive gateways to it.

The “More info needed?” exclusive gateway by default leads to the “Follow-up” task, so make sure to set the flow to this task as the default flow. The “More info needed?” exclusive gateway flows to the “Approval” user task in the case that the “Review” user task selected “completed” as its action, so make sure that this flow has a “Flow condition” of:

```
${action == 'completed'}
```

The “Approve request?” exclusive gateway by default also leads to an end event so make sure to set the flow from this gateway to the end event as the default flow. The “Approve request?” exclusive gateway flows to the “RequestApproval” end task in the case that the “Approval” user task selected “approved” as its action, so make sure that this flow has a “Flow condition” of:

```
${action == "approved"}
```

The “Proceed with Review?” exclusive gateway by default also leads to an end event so make sure to set the flow from this gateway to the end event as the default flow. The “Proceed with Review?” exclusive gateway flows back to the “Review” user task in the case that the “Follow-up” user task selected “continue_review” as its action, so make sure that this flow has a “Flow condition” of:

```
${action == "continue_review"}
```

As the final step the flows between the individual elements need to be created. Make sure to give them unique IDs (e.g. “flow1”, “flow2”, …) and that they connect correctly.

Completing and saving your work

Before saving the new template, you should run the workflow editor’s validation functionality in order to detect any errors or problems that you might have created during your workflow template remodeling. It is recommended to fix any errors or issues that are reported during the validation step before using the template in Cloud Pak for Data.

Then save your workflow template using the workflow editor's save function. In the Save dialog, you should make sure that the template's name and key is what you have specified and intended before and that a new unique key is used for this new template.

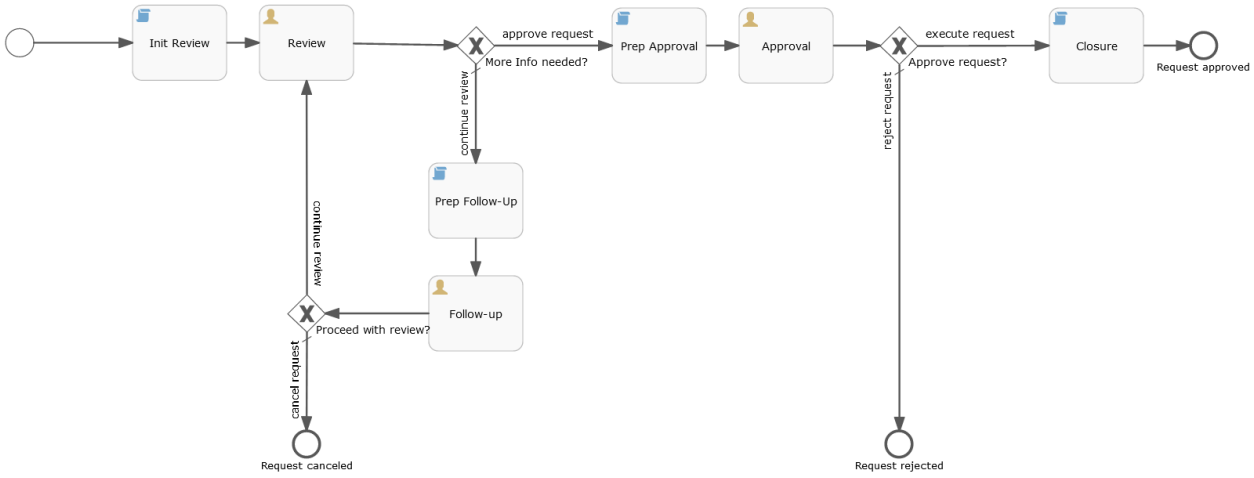
Running the new template in Cloud Pak for Data

Your new template can now be imported into Cloud Pak for Data and activated as a new workflow configuration by the workflow administrator:
Login to Cloud Pak for Data and navigate to Workflow Types, select "Workflow management" and create a new workflow type. After having specified the new workflow's type, you can then import your new workflow template.

Second approach with additional JavaScript elements

The first approach in implementing this general change request workflow template avoided the use of script task elements completely. As a consequence, the workflow was neither able to update the workflow's state display during the workflow execution nor to write the progress of the workflow to a log file. Hence, in a second approach all this functionality will be added through some script task elements. These will contain JavaScript code in order to complement the missing parts.

It is recommended to add a new script task element as a preparation step for each user task element. Also, as the final step we want to be able to execute some code. So add four new script task elements and position three of them before the user task elements and one after the "Approve request" exclusive gateway.



As the next step modify the existing flows between the individual elements and add new flows where needed. Make sure to give the new flows unique IDs (e.g. "flow10", "flow11", …) and that they connect correctly.

Script Task Init Review

Make sure that this script task element has a new and unique ID (e.g. "init_review") and a value of "javascript" as the "Script format" property.

Add the following script to the script task's "Script" property:

```
// print/log input values
```

```
if (typeof title !== 'undefined') {
   print('Title: ' + title);
   execution.setVariable('artifact_name', title);
}
if (typeof summary !== 'undefined') {
   print('Summary: ' + summary);
}
if (typeof description !== 'undefined') {
   print('Description: ' + description);
}
if (typeof category !== 'undefined') {
   print('Category: ' + category);
}

// set workflow state
// workflowStateAndActionLogger.updateWorkflowState(execution, 'Change request submitted', 'submit',
false, comment, null);
print("Change request for {artifactName} started");

// reset global variables
execution.setVariable('action', null);
execution.setVariable('comment', null);
```

Script Task Prep Approval

Make sure that this script task element has a new and unique ID (e.g. “prep_approval”) and a value of “javascript” as the “Script format” property.

Add the following script to the script task’s “Script” property:

```
// set workflow state
// workflowStateAndActionLogger.updateWorkflowState(execution, 'Change request submitted for approval,
'submit', false, comment, null);
print("Change request for {artifactName} submitted for approval");

// reset global variables
execution.setVariable('action', null);
execution.setVariable('comment', null);
```

Script Task Closure

Make sure that this script task element has a new and unique ID (e.g. “closure”) and a value of “javascript” as the “Script format” property.

Add the following script to the script task’s “Script” property:

```
// set workflow state
// workflowStateAndActionLogger.updateWorkflowState(execution, 'Change request approved', 'submit', false,
comment, null);
print("Change request for {artifactName} approved and completed");

// reset global variables
execution.setVariable('action', null);
execution.setVariable('comment', null);
```

Script Task Prep Follow-Up

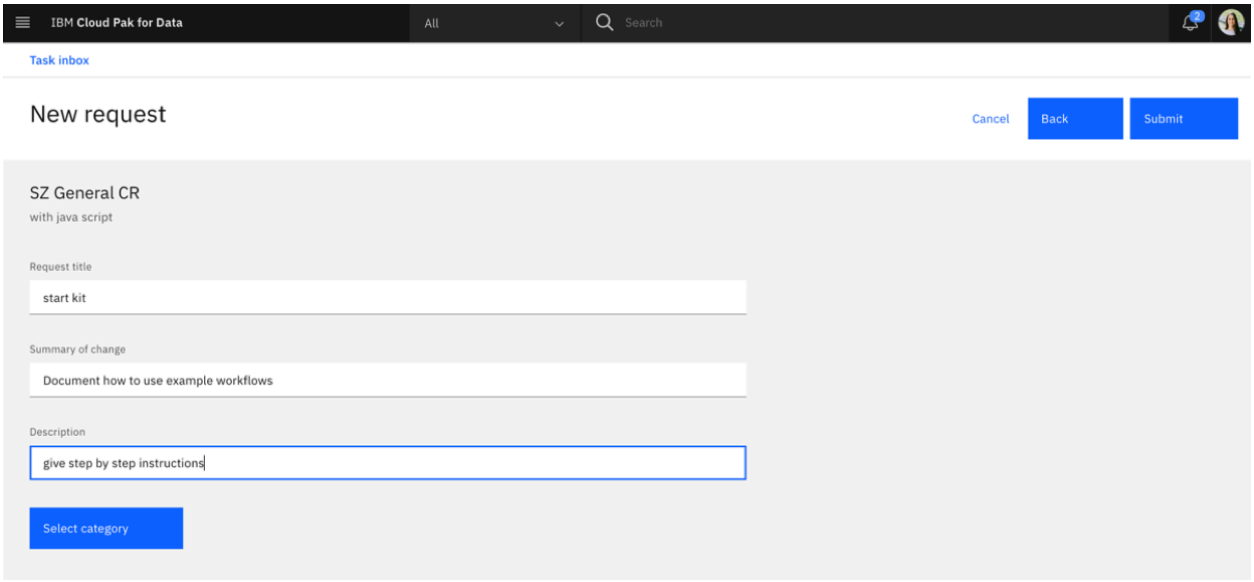
Make sure that this script task element has a new and unique ID (e.g. “prep_follow_up”) and a value of “javascript” as the “Script format” property.

Add the following script to the script task’s “Script” property:

```
// set workflow state
// workflowStateAndActionLogger.updateWorkflowState(execution, 'Change request submitted for follow-up',
'submit', false, comment, null);
print("Change request for {artifactName} submitted for follow-up discussions");

// reset global variables
execution.setVariable('action', null);
execution.setVariable('comment', null);
```

After importing the template and creating a default configuration you can start a new instance with “New request” in the Task inbox. This will ask for the form properties you defined in your process.

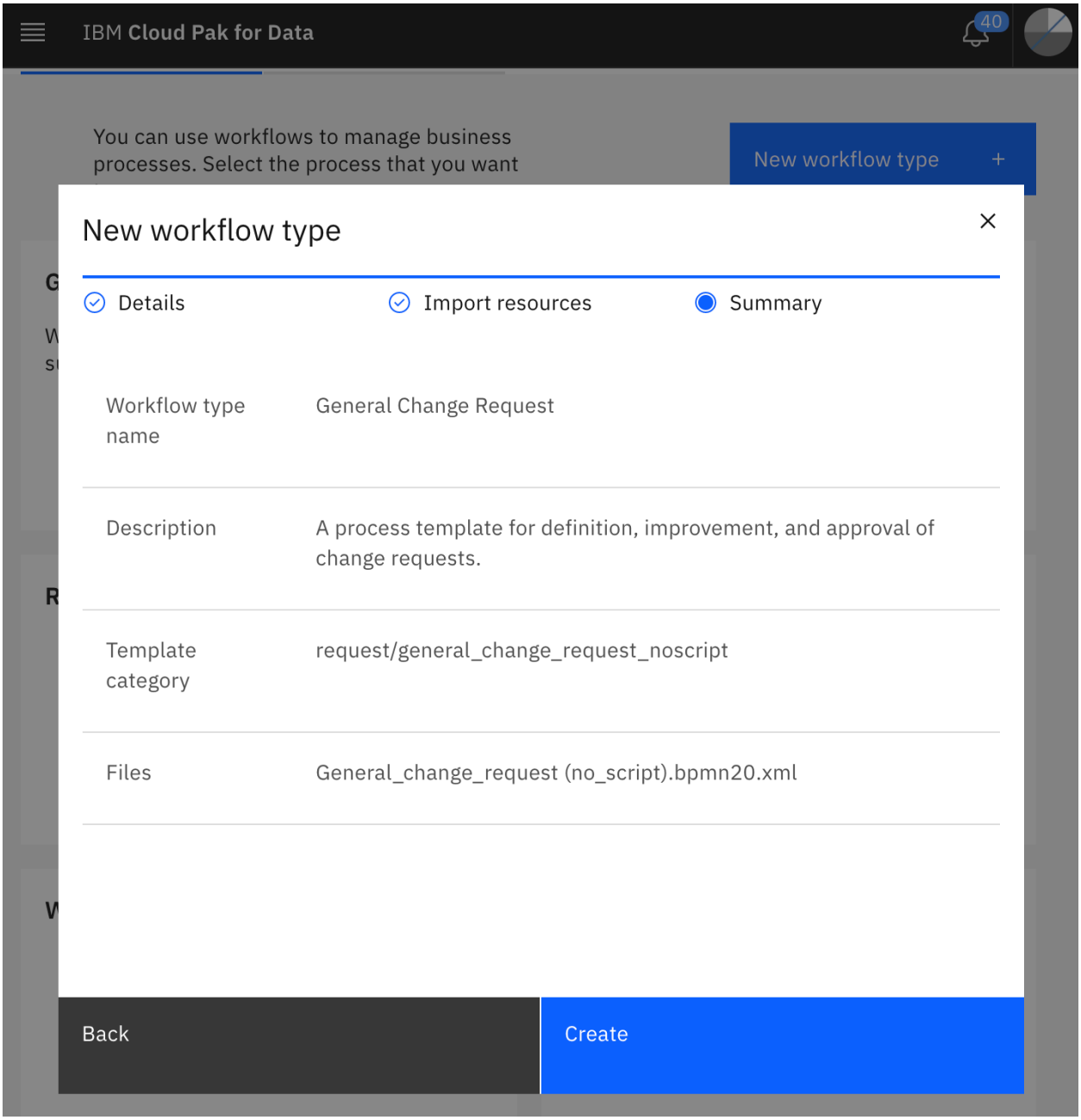


Appendix I:

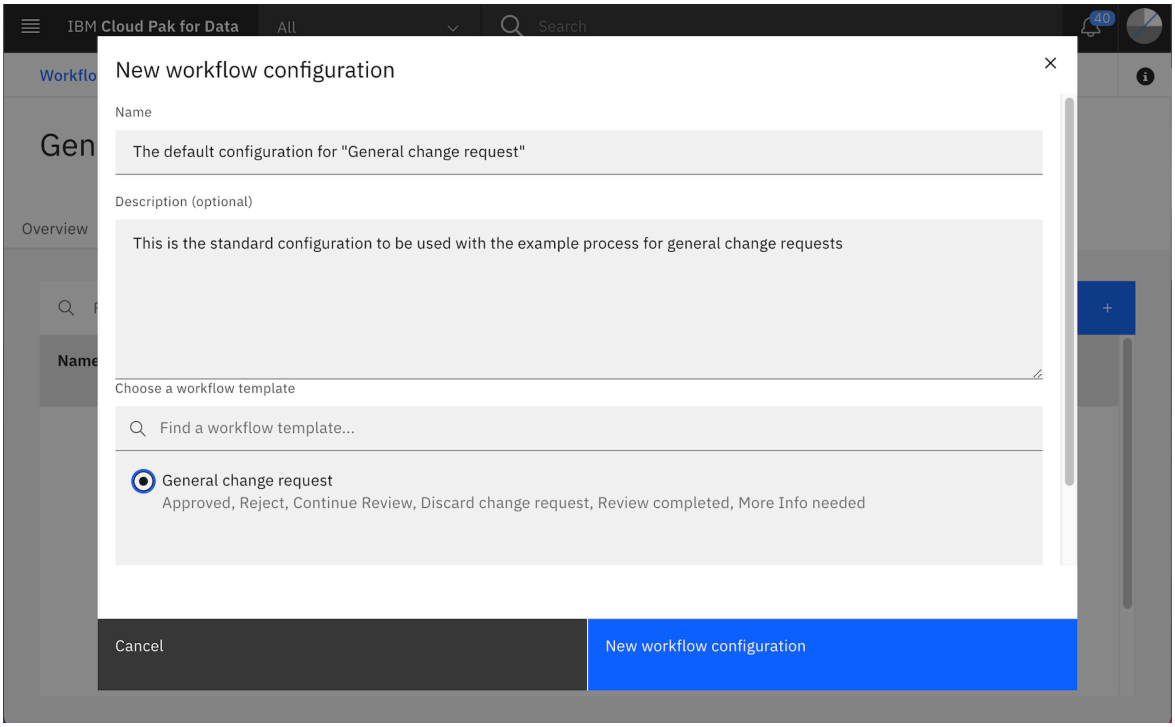
This appendix gives a more detailed walkthrough on upload and execution. For information on the deployment of the workflow file and the creation of a workflow configuration see Appendix I of the first scenario. In the following, we focus on how to start a new workflow instance (Step 2 below).

Step 1: Deployment & configuration

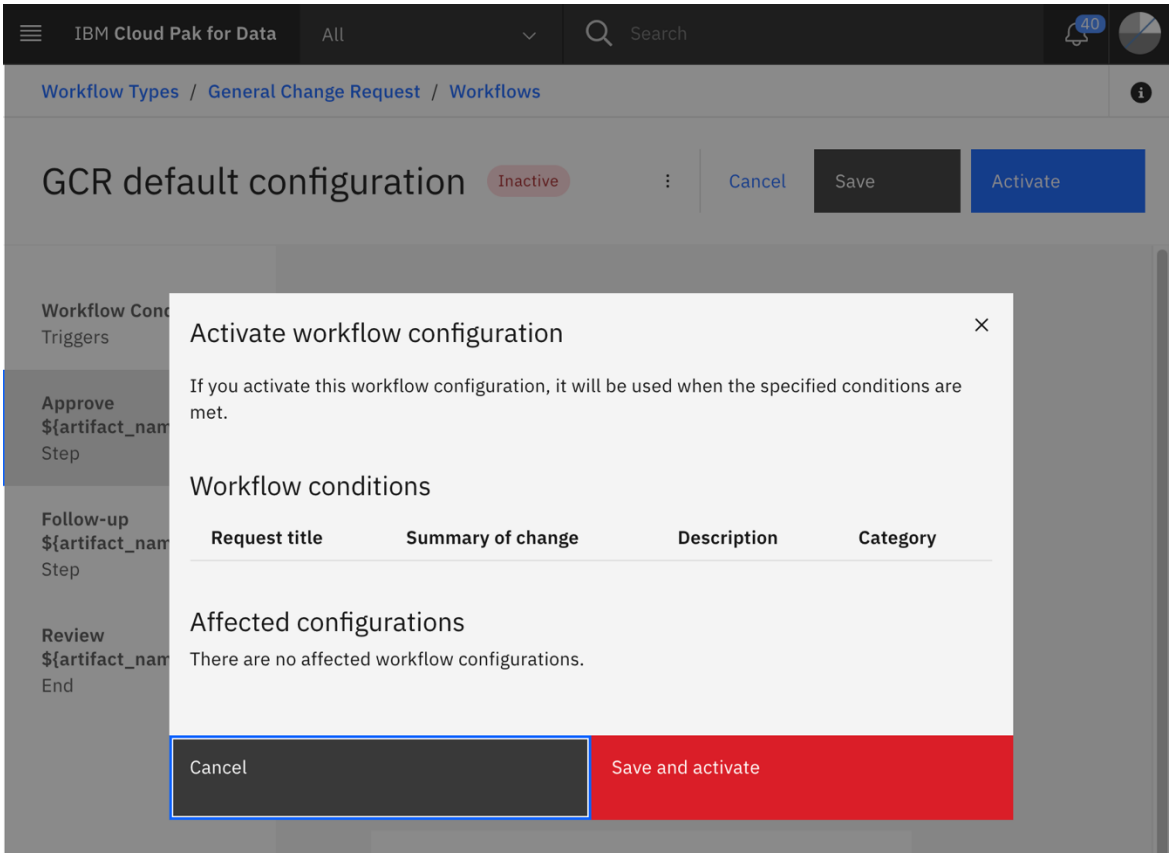
Follow the steps in Appendix I of scenario 1 for the non-javascript version of your general change request bpmn file. After you prepared the upload, you should see a similar summary as in the following screenshot.



Along the lines of Appendix I in Scenario 1 create a default configuration.

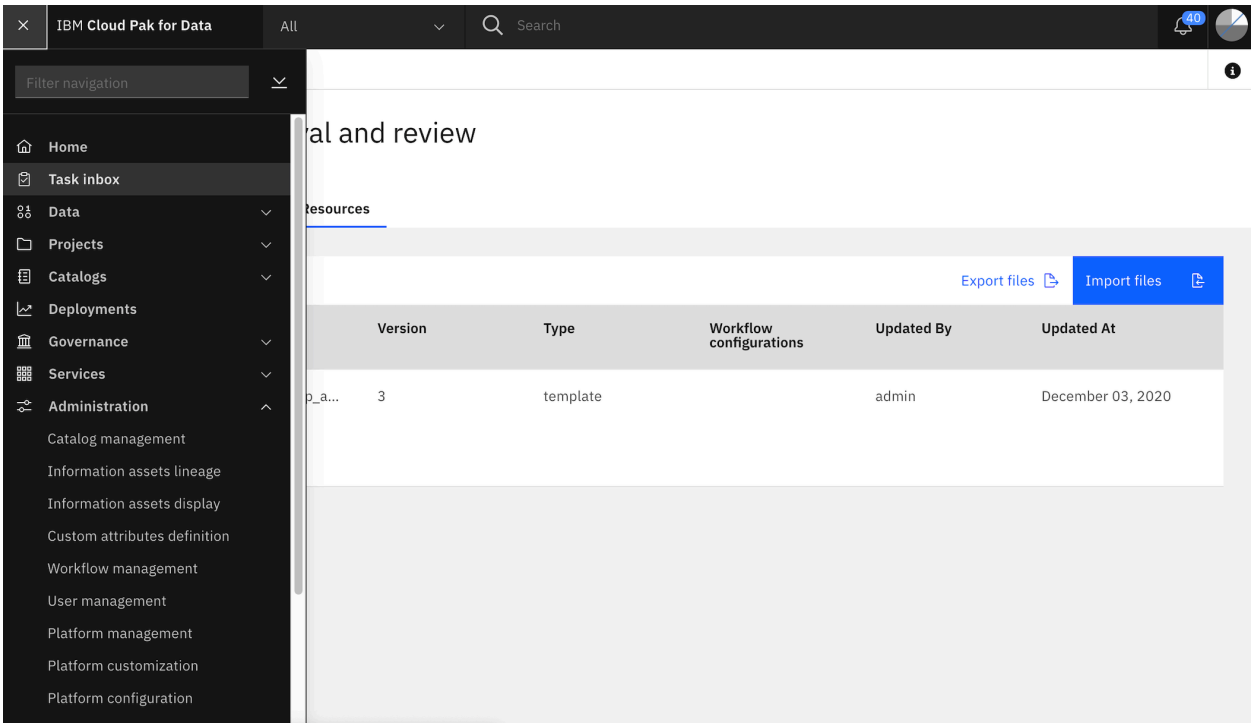


Select “Workflow requester” as possible assignee for all steps, then save and activate the configuration.

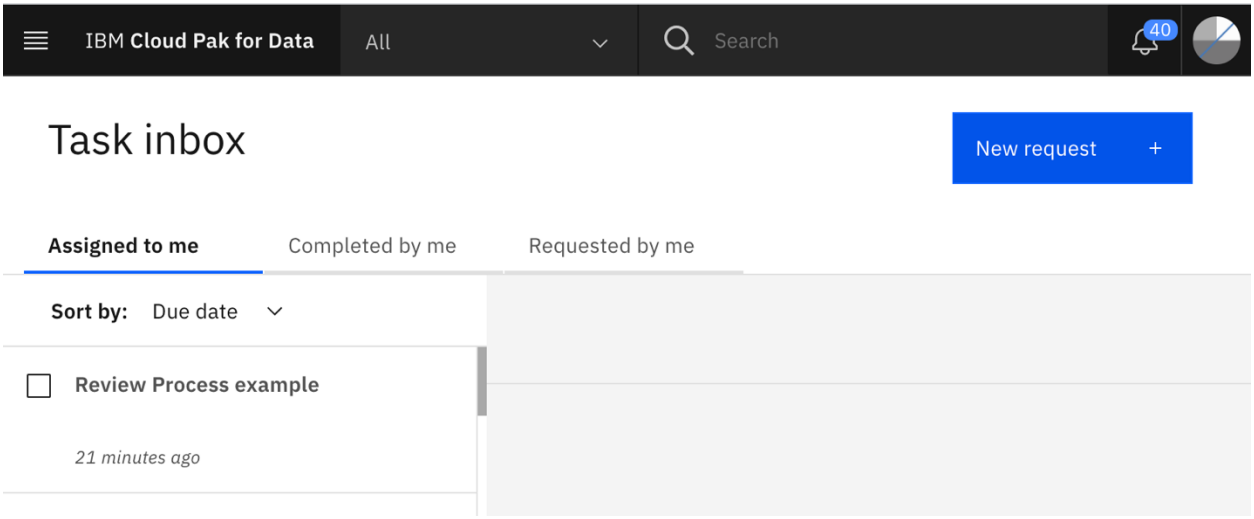


Step 2: Starting a workflow instance

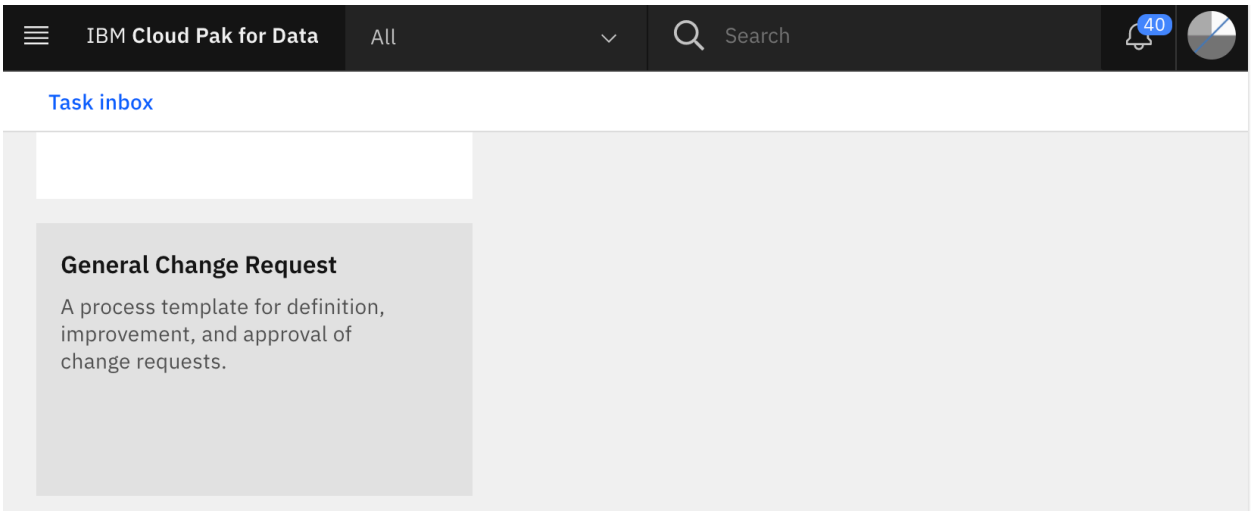
Workflow instances are started via the Task inbox. You can jump to it from the home page or using the main navigation menu.



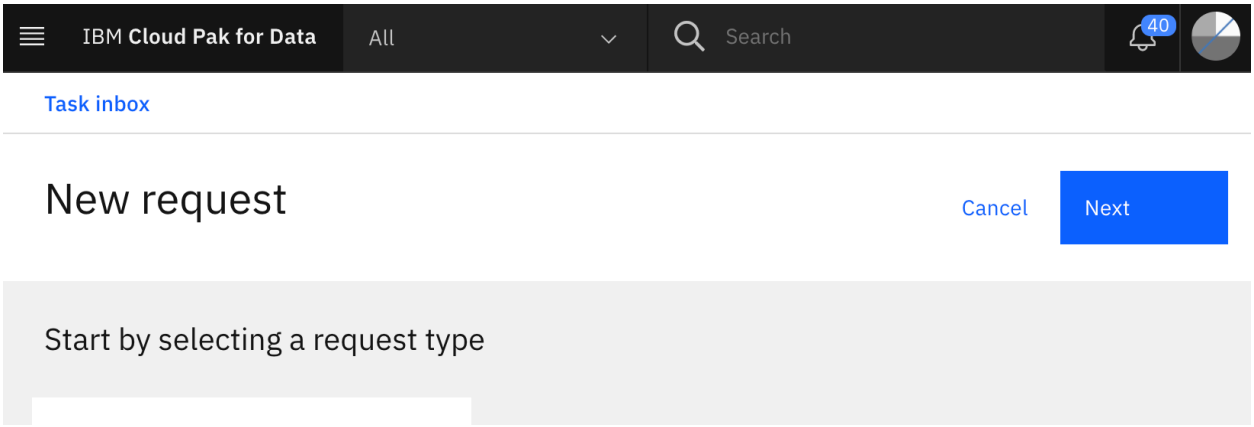
The top right corner of the Task inbox provides the “New request” button



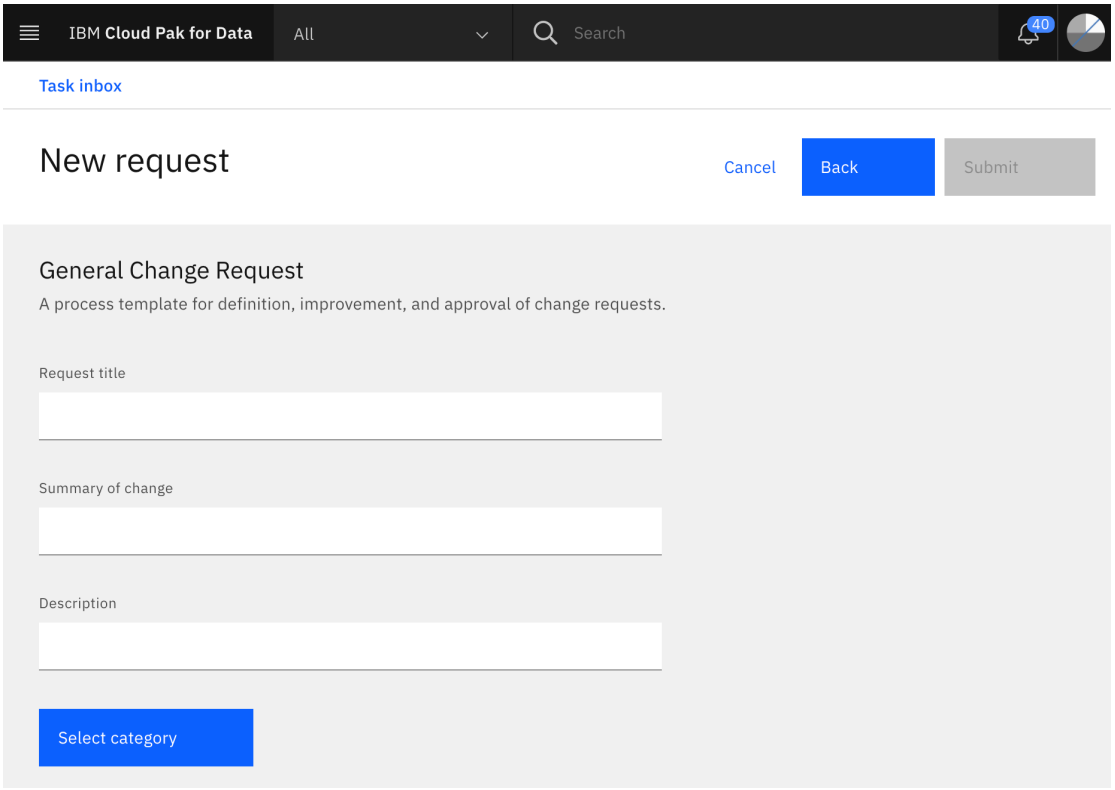
Press the “New request” button, scroll down the page and select our general change request template for request creation.



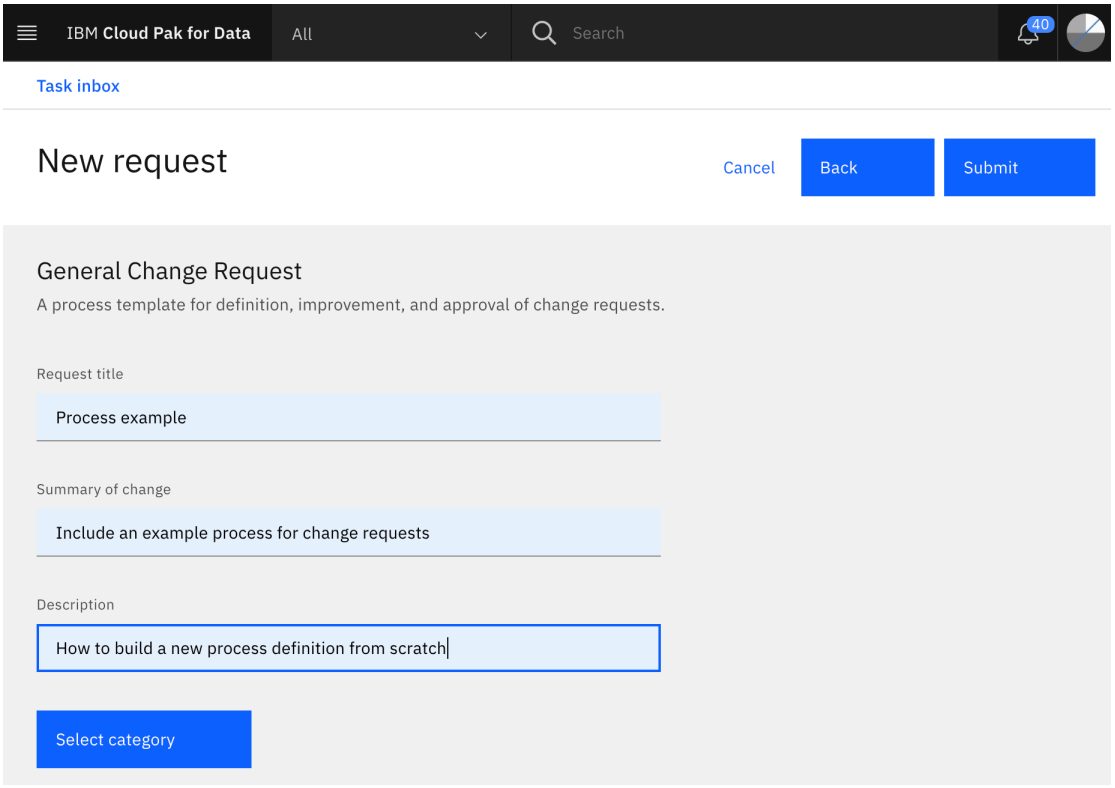
Now the button on the top of the page shows “Next” …



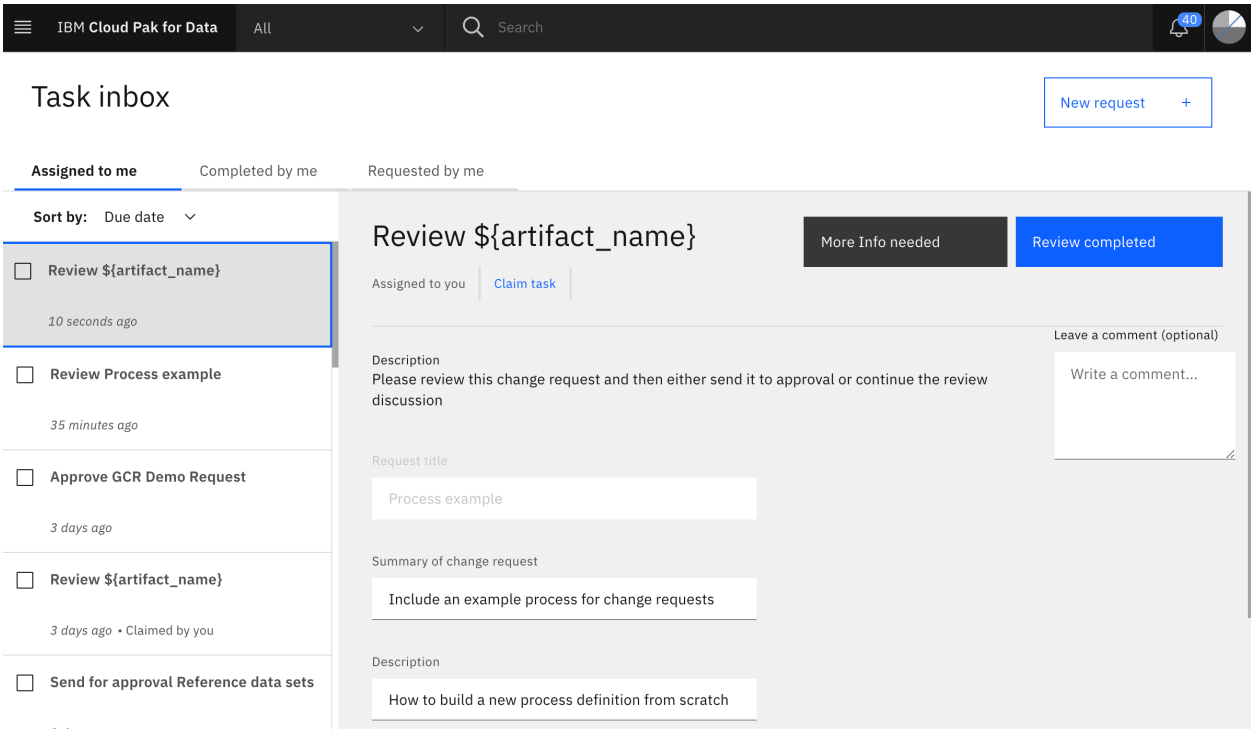
... which leads you to the screen where you input the values for the form properties:



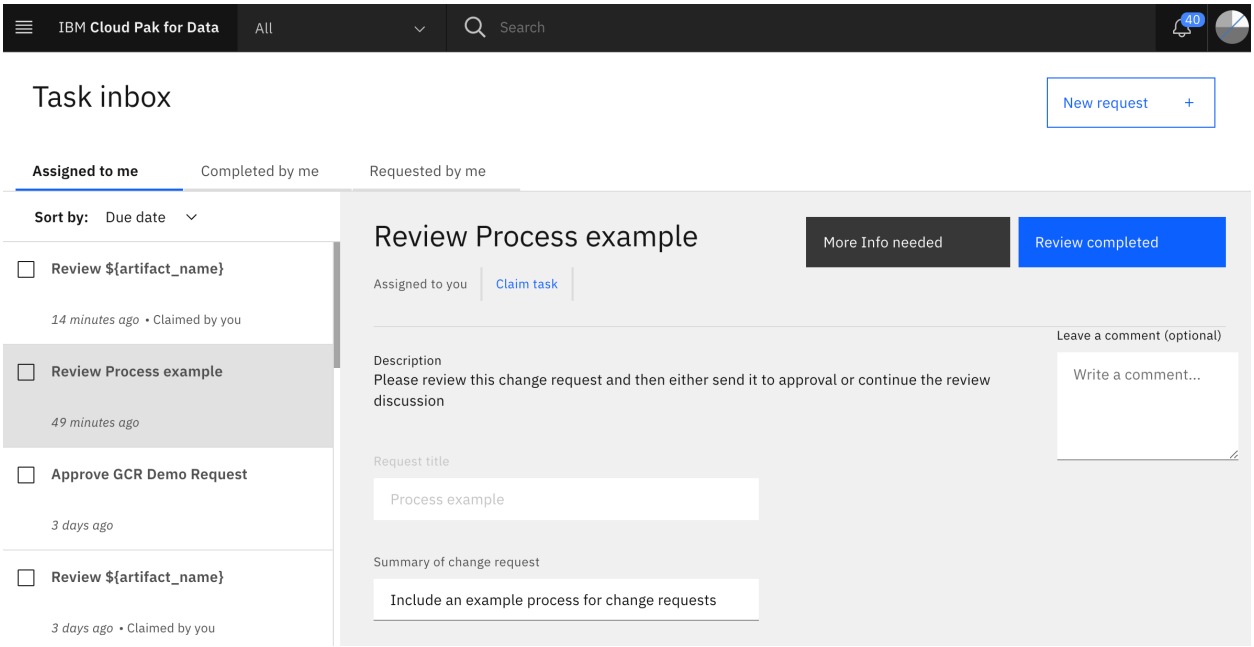
Submission of the form is enabled as soon as all required properties are entered.



Now a new Task appears in the task list of the task inbox.



As explained above, the general change request without javascript does not set execution variables. Since the artifact_name variable is not set, the term ${artifact_name} is not substituted. If you replace ${artifact_name} with e.g. ${title} in the bpmn file (and provide unique definitions:targetNamespace and process:id to allow for reimport), the task display will substitute the variable-term with the title of the change request:



In the javascript version of the change request, the artifact_name variable is set in the init_review step.