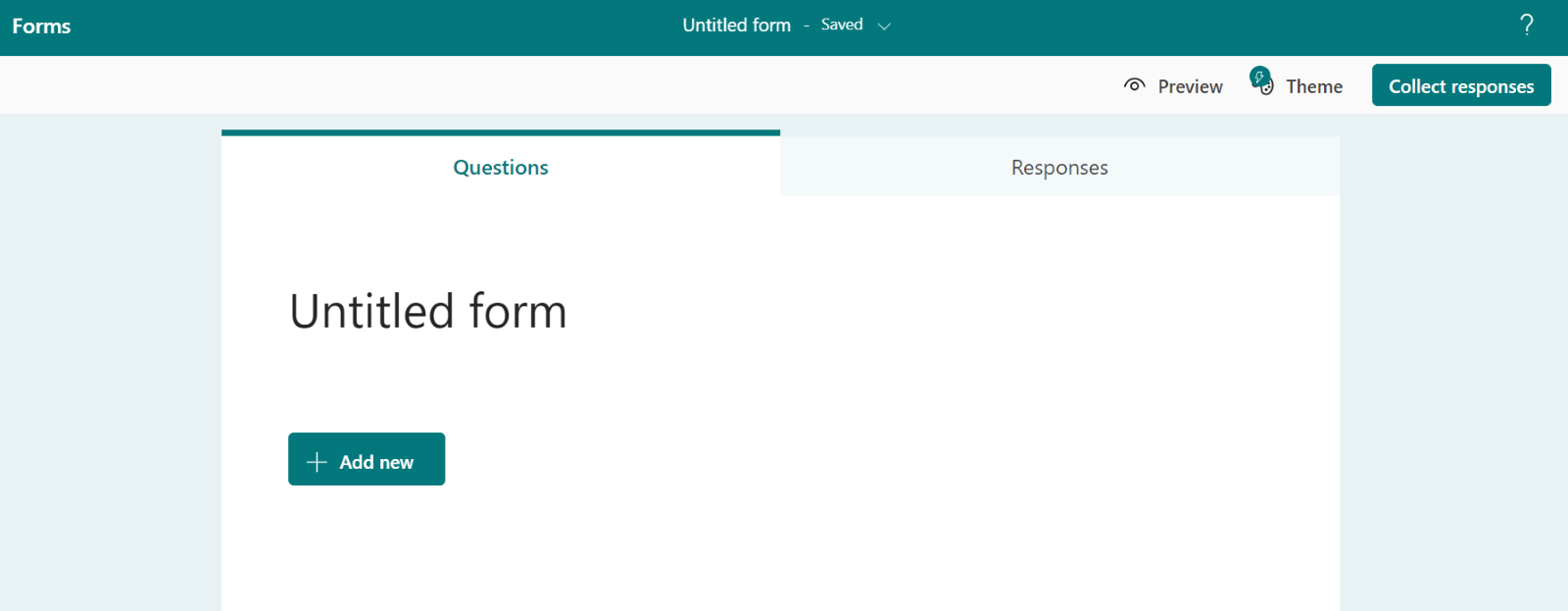
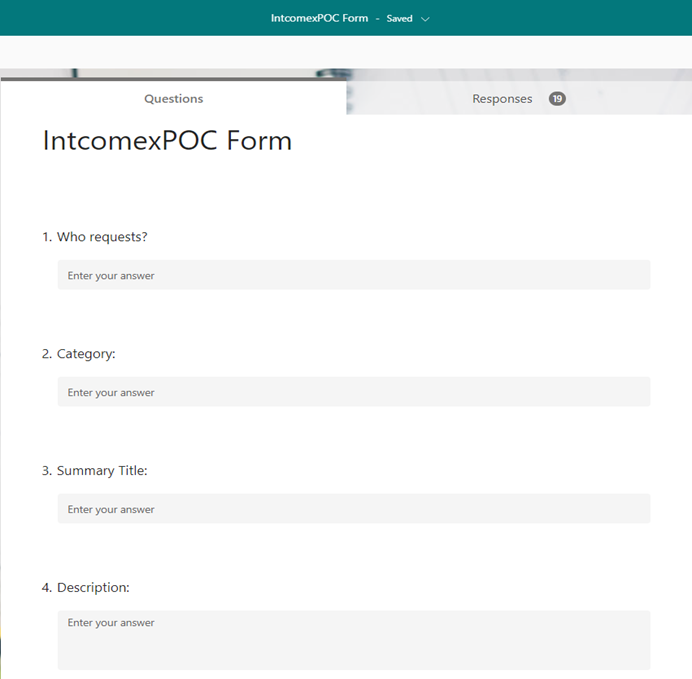
**JIRA-AZUREDEVOPS INTEGRATION**

**Microsoft forms:**

* Take a new Microsoft form
* Add required fields as per the requirement



* Fill the form
* Submit the response

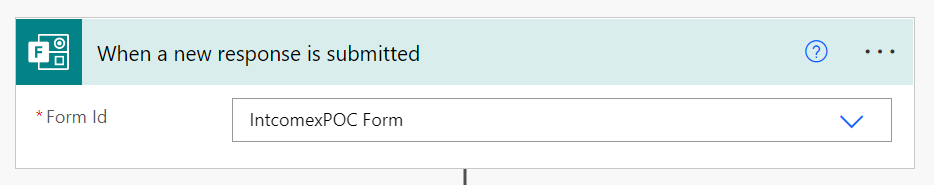


**Power Automate flows: Jira-AzureDevops**

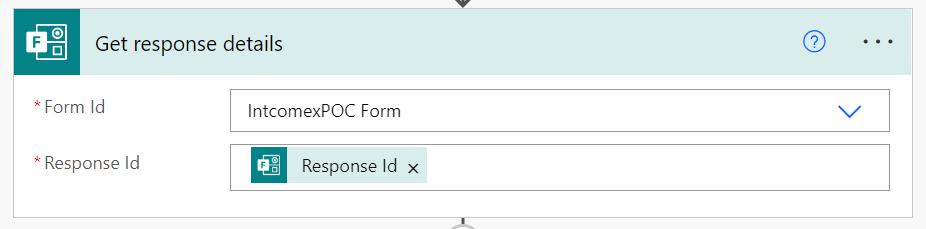
1. **Create an issue in Jira after the form is submitted and after the approval of Jira Issue, flow creates an epic in Azure Devops:**

* When a new response is submitted, the flow will create an issue in Jira.
* It will send an email for approval.
* It delays the flow for the scheduled time and after the approval, the flow will create an epic in AzureDevops.

**The following steps explain the flow:**

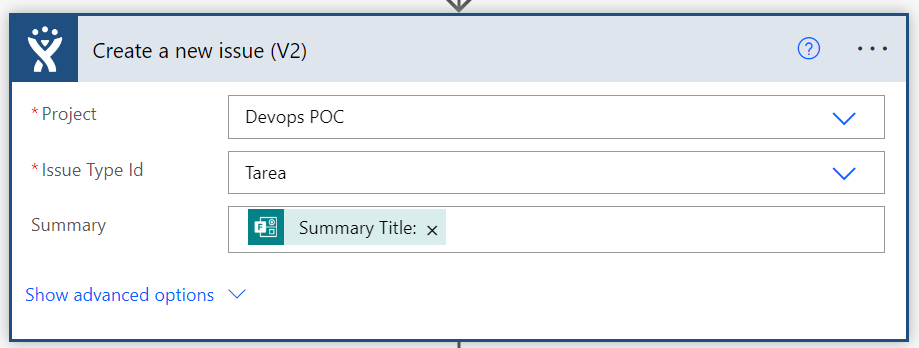


This operation triggers a flow when a new response is submitted.

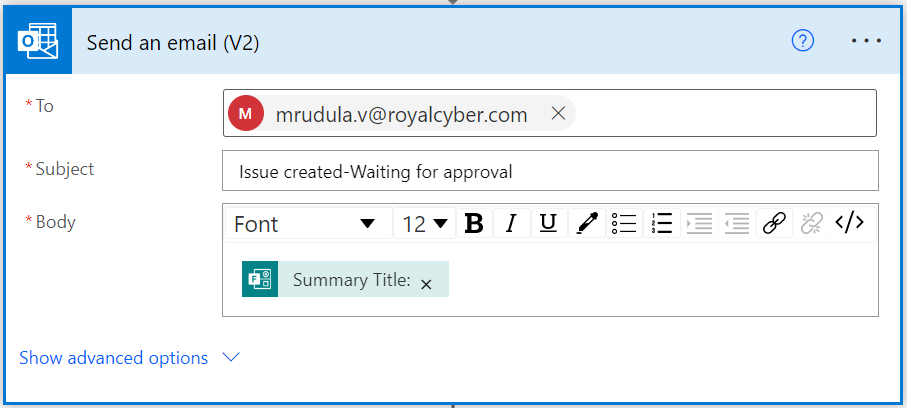


This action retrieves a form response, the Response Id takes the response when a form is

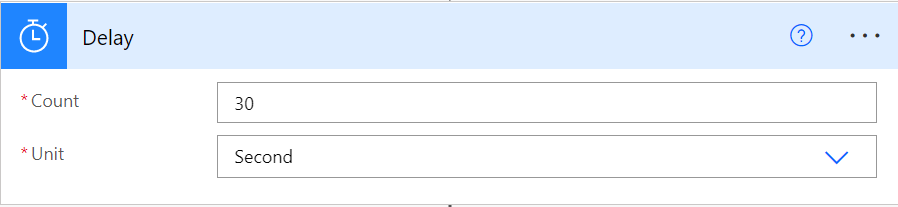
Submitted.



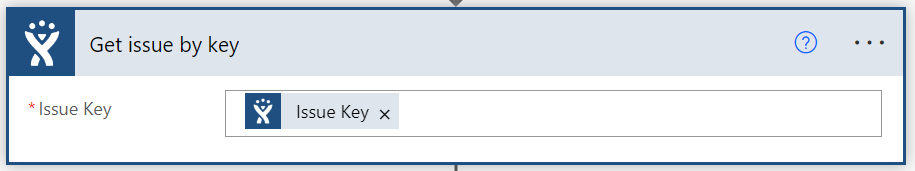
This operation is used to create a new issue. We can select the project and Issue type from Jira, since it is connected by establishing the connection between Power Automate and Jira. Summary takes the dynamic response from the form submitted.



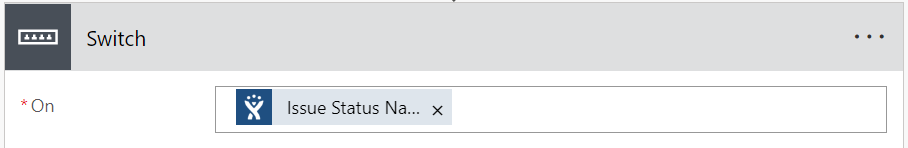
This operation sends an email to the approver, notifying that issue has been created and waiting for approval. Body of the email contains the dynamic content from the form response.



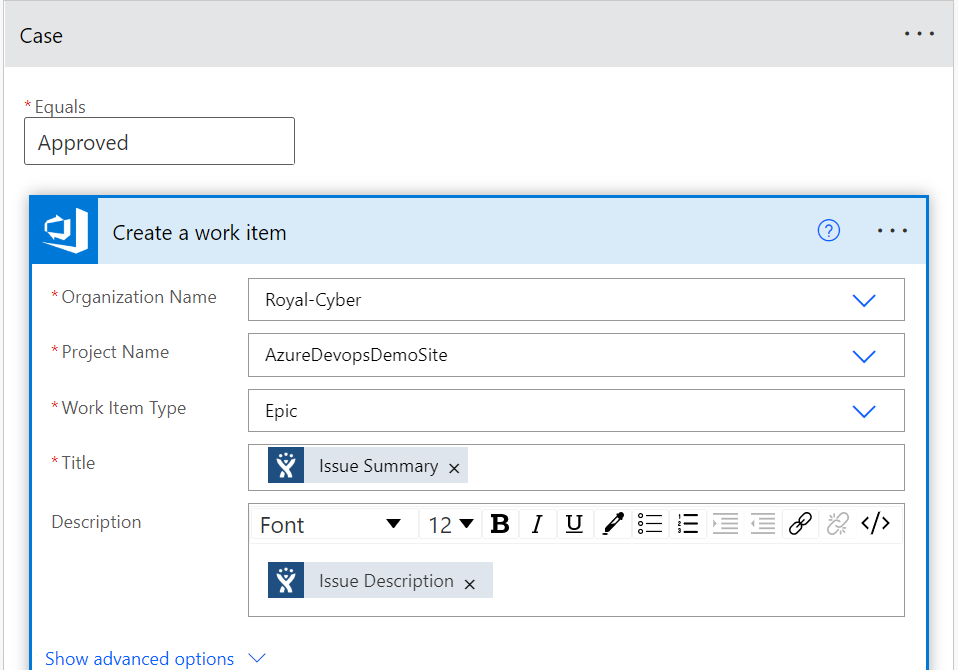
This operation delays the flows to review the Issues in Jira. We can define the count and unit as per the requirement.



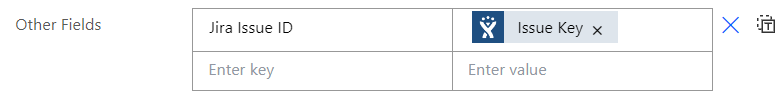
This operation is used to retrieve the issue object for a given issue Key.



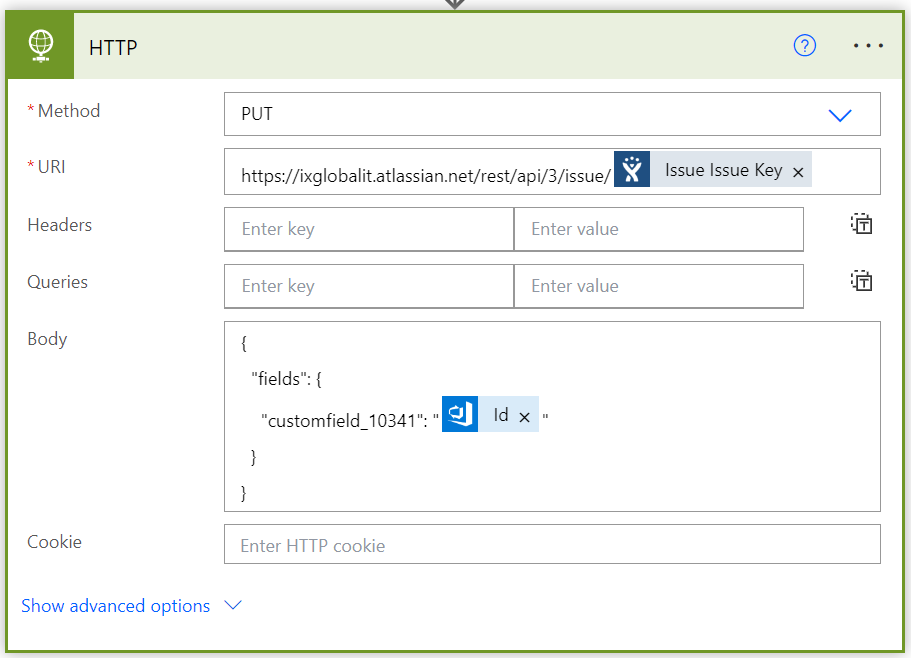
The switch case take the status of the Jira issue and we can create actions for different cases.



When the ‘Issue status’ response is ‘Approved’, this operation creates a new work item with the provided attributes in this case. The dynamic values received from the Jira issue will be updated to fields of the work item. Here we have chosen work item to be Epic and the title of the Epic will be the Jira Issue summary and the Description will take the response same as the Issue description.



While creating work item, we have given that Jira issue ID field in AzureDevops will take the value of Jira Issue key.

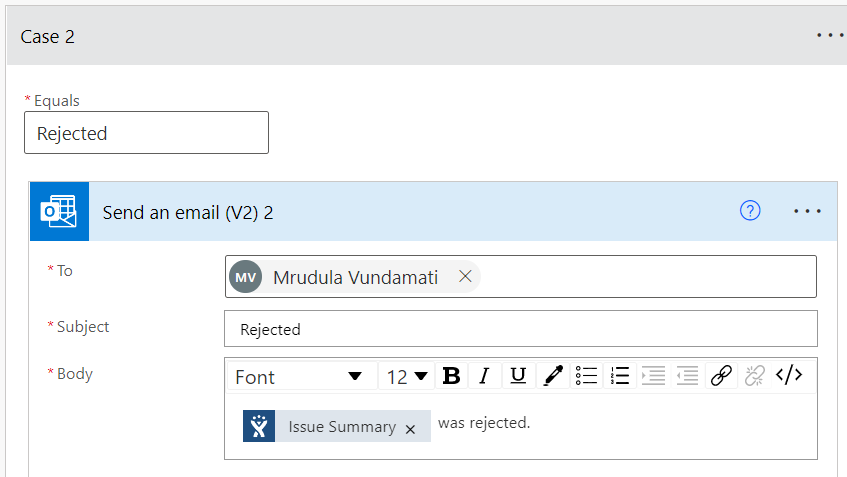


This operation updates the Jira Issue using HTTP action. Here the PUT method applies the changes to the Jira issue, which was linked to work item, by the provided URL of the Jira issues. The body contains the JSON text, which tells the HTTP action to update the Jira field (AzureDevops Epic ID) by the work item ID so that the work item and Jira are linked to each other.

HTTP actions enable you to interact with APIs and send web requests that perform various operations.

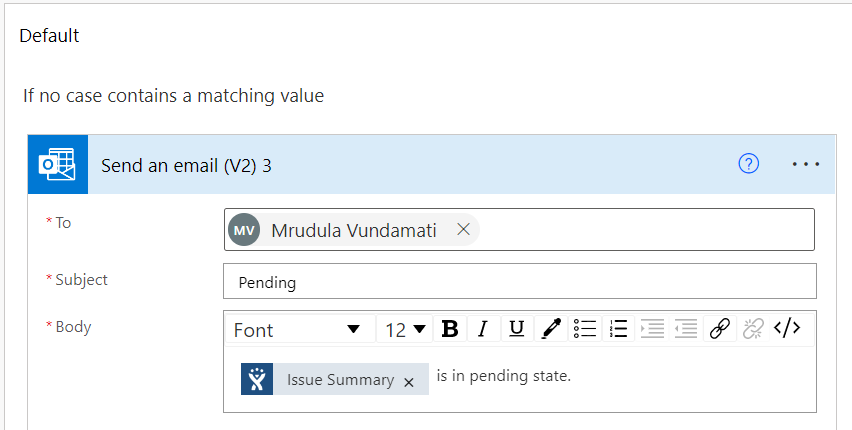
To send an API request, like POST, GET, PUT, or DELETE, use the Invoke web service action.

In the action's properties, you must populate the service's URL and the appropriate HTTP method.



In this when the Jira Issue status name is ‘Rejected’, this action will send a message by email to the responder, that the form was rejected.

In other case of the issue status, the form will be in pending state and it sends email to responder saying it is still in pending state.

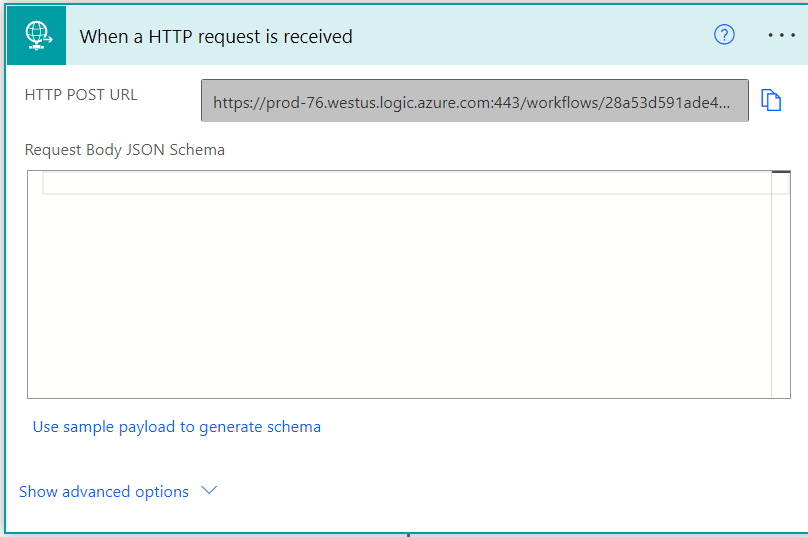


1. **Recurrence flow-AzureDevops (for the issues which are yet to approve):**

* The flow will run for scheduled intervals and checks whether any issue is updated.
* If the issue is approved and it will create an epic in Azure Devops
* Updates the epic id in Jira issue.

**The following steps explain the flow:**

When an HTTP request is received:



When first adding the ‘When an HTTP request is received’ trigger, to a flow it starts with an HTTP POST URL informing us that the URL will be generated after the Flow has been saved. This means that initially creating the Flow will not allow us to be able to provide/use the URL that is required to trigger the Flow.

After entering JSON Schema into the box and hit done, the schema will be created and displayed in the ‘Request Body JSON Schema’ section as shown above. The above step retrieves the data from Jira.

Get Issue by Key:



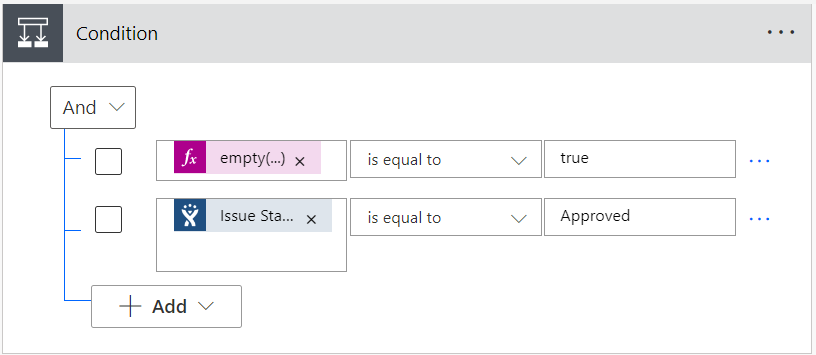
This is a Jira Connector, which gives the data about the Jira Issue Key from the key which is passed from the HTTP request connector.

Parse JSON:



In this step, Fields of Jira Issue will be parsed and gives the data of each field that is associated with a Jira Issue Key.

**Condition:**



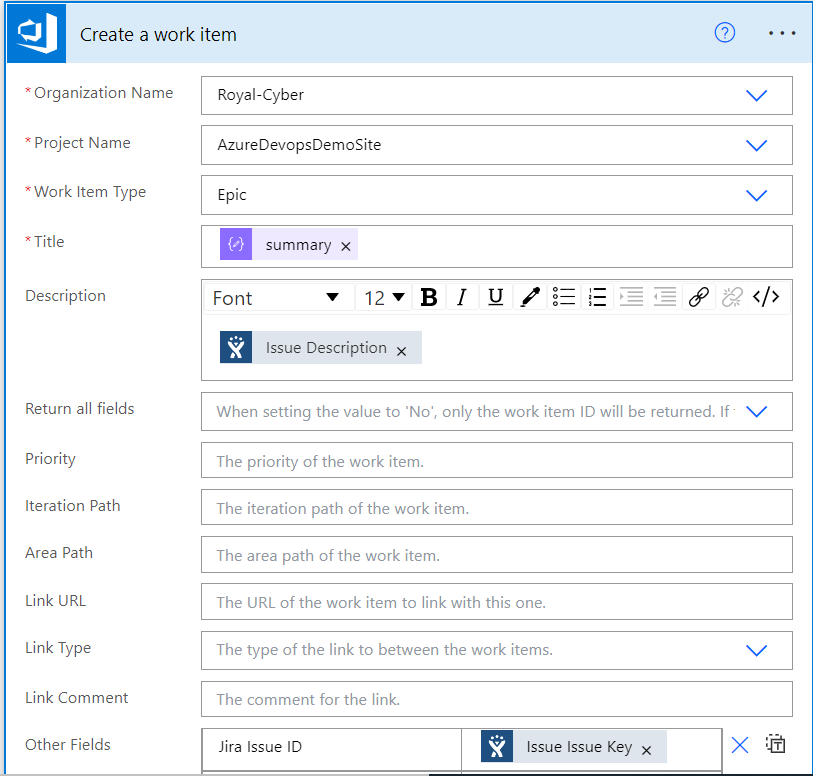
Then, there are two conditions these 2 must be validated.

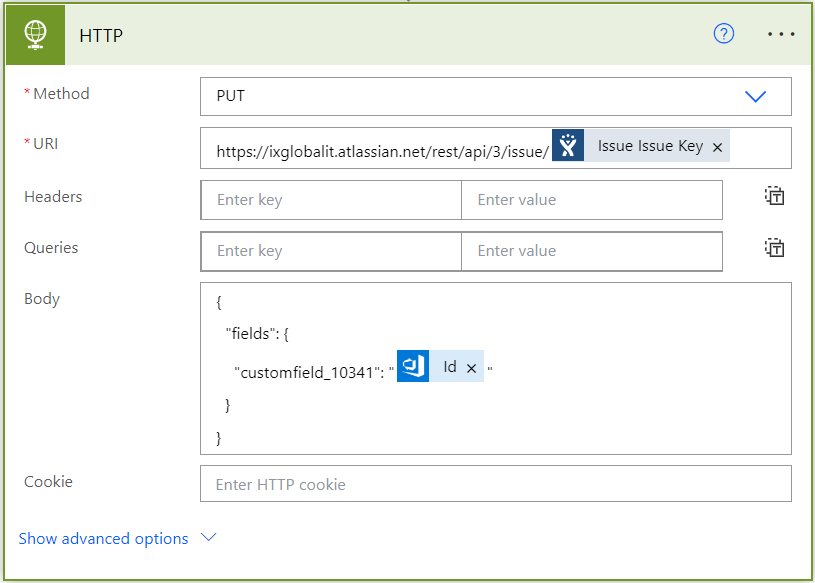
1. Empty (): This operation is to check whether any value is assigned or not, this should be true.
2. Jira Issue Status Name: This field is to check whether the status is in the approved state or not.

If the condition Yes then, there are 2 steps as follows:

**Create a work item:**

Create a new work item with the provided attributes.

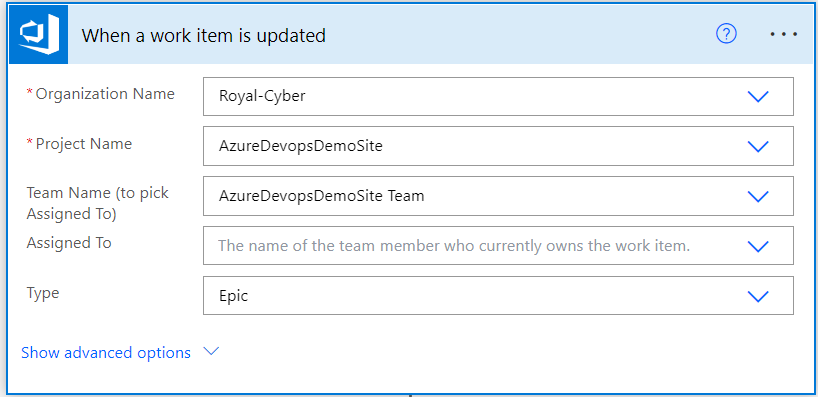


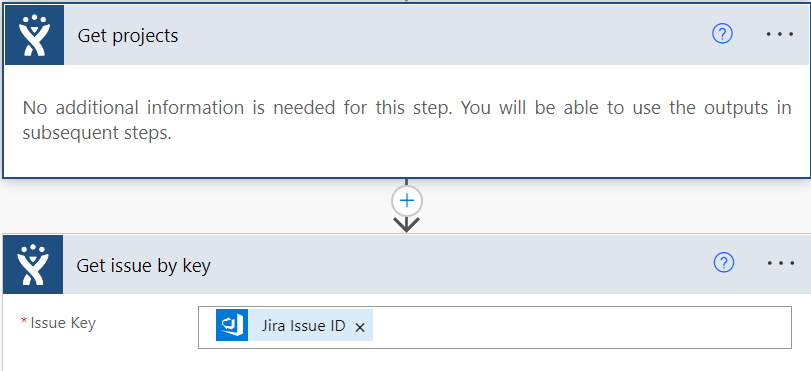


This operation updates the Jira Issue using HTTP action. Here the PUT method applies the changes to the Jira issue, which was linked to the work item, by the provided URL of the Jira issues.

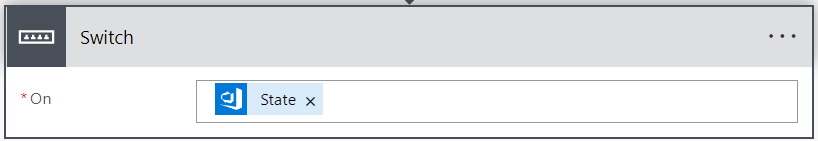
1. **When epic is updated in AzureDevops, flow updates the Jira issue:**

* When an epic is updated (the state changes), the flow will edit the status of Jira via HTTP request.

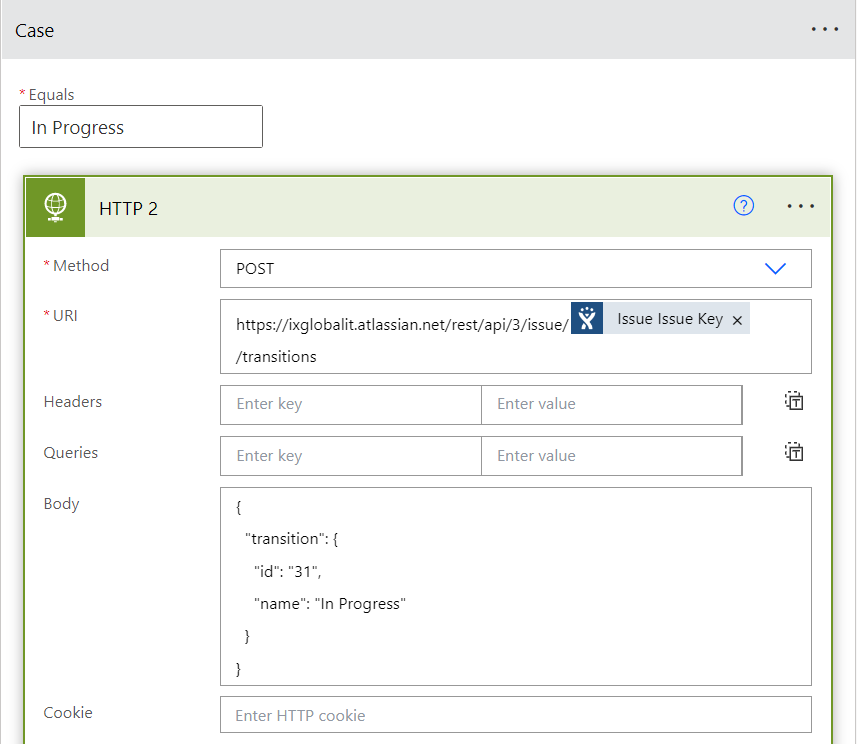
  
Triggers a flow when a work item which matches provided criteria is updated.

  
“Get projects" operation is used to retrieve a list of projects for your JIRA instance.

The “Get issue by key” operation is used to retrieve the issue object for a given issue Key. Here Issue key field is filled by the AzureDevops field “Jira Issue Id” since the Jira issue ID is the same as the Jira Issue key, it retrieves the same information.

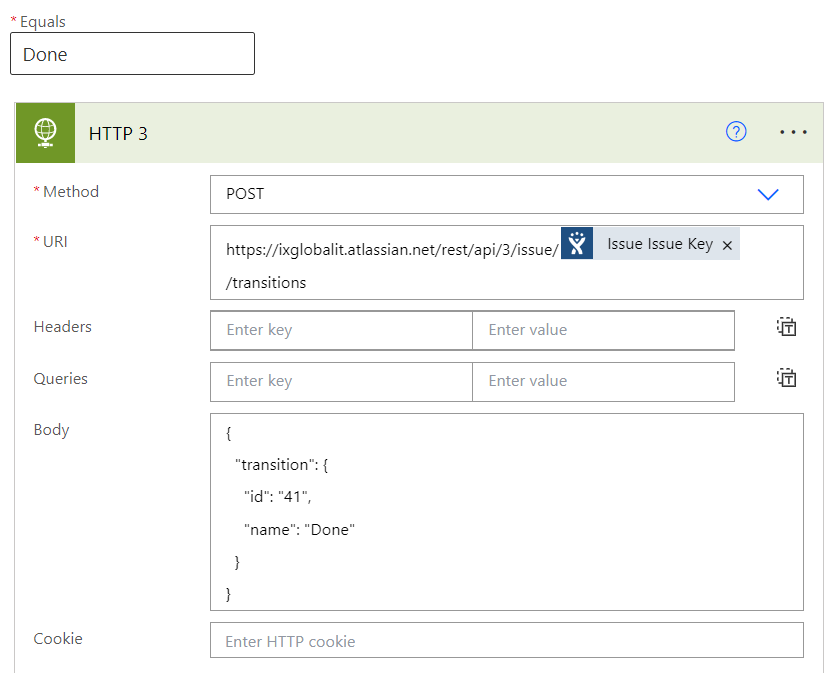


Here the “Switch” operation takes the value of “State” field of AzureDevops Work item.

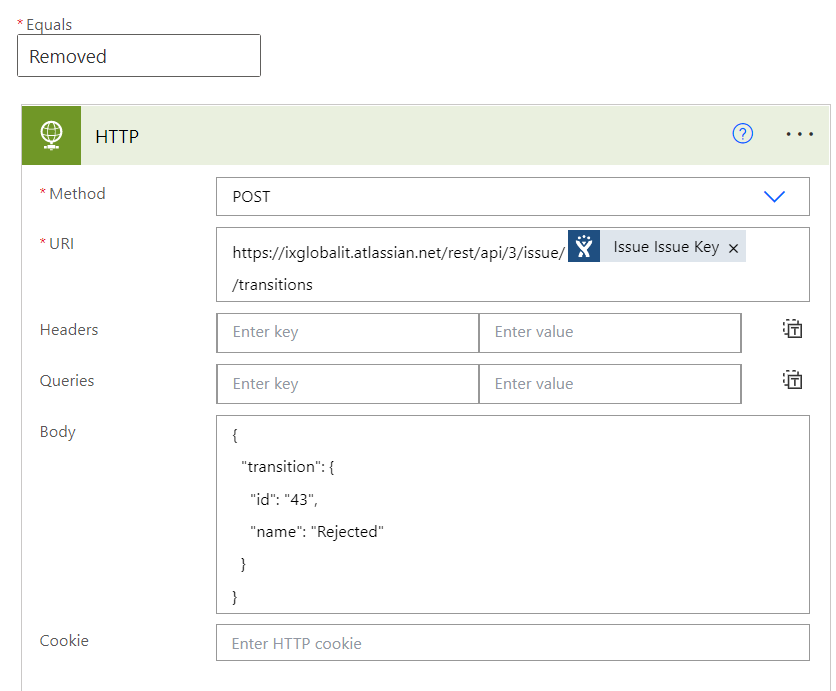


In this case, when the “State” of the work item is “In Progress”, the “HTTP” operation updates the status of the Jira issue by the POST method. The URL here used is for transitioning the Jira issue. The body is in Json format, the id indicates what would be the status name of the Jira issue.

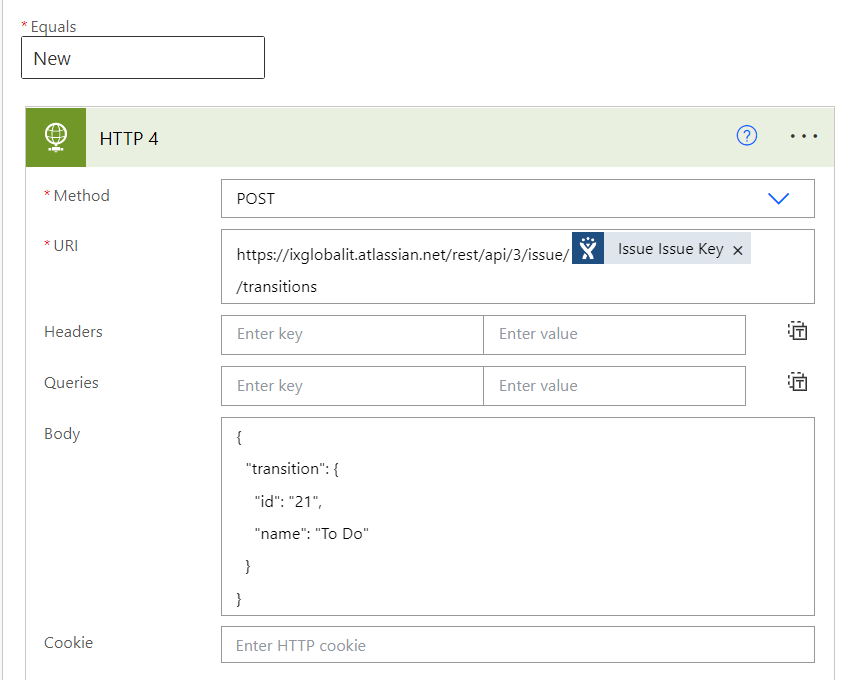
In Case of “Done” state:



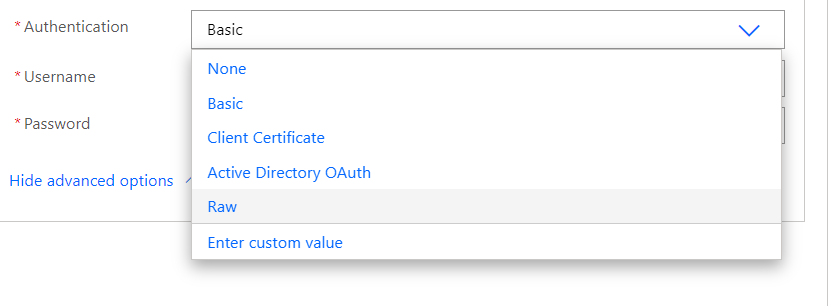
In Case of “Removed” state:



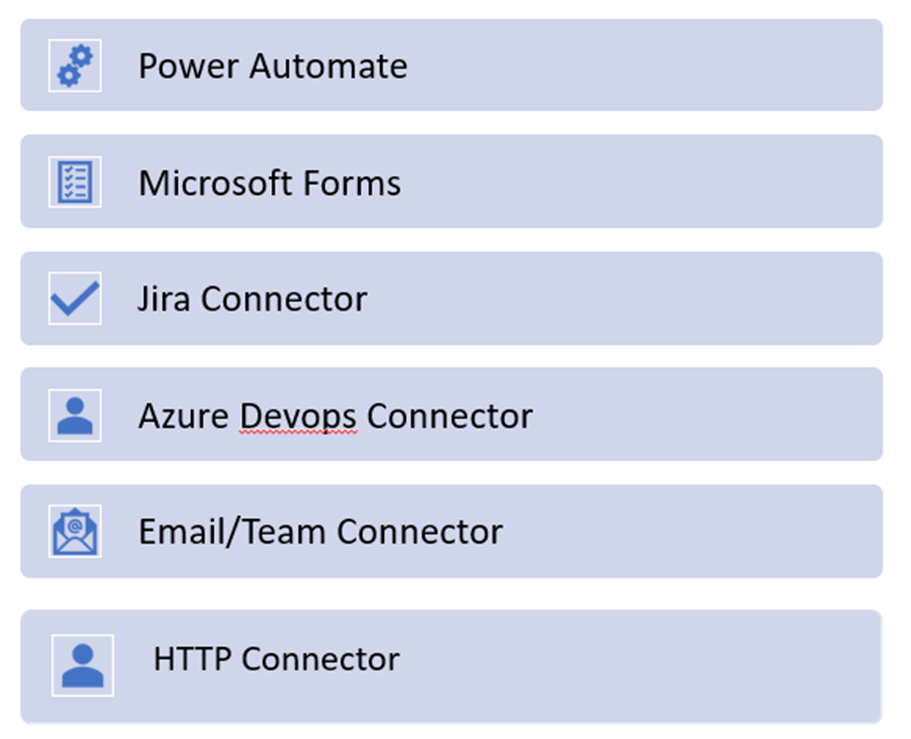
In Case of “New” State:



There are different types of authentications. Here we have used basic authentication, which requires email id and Personal access token.



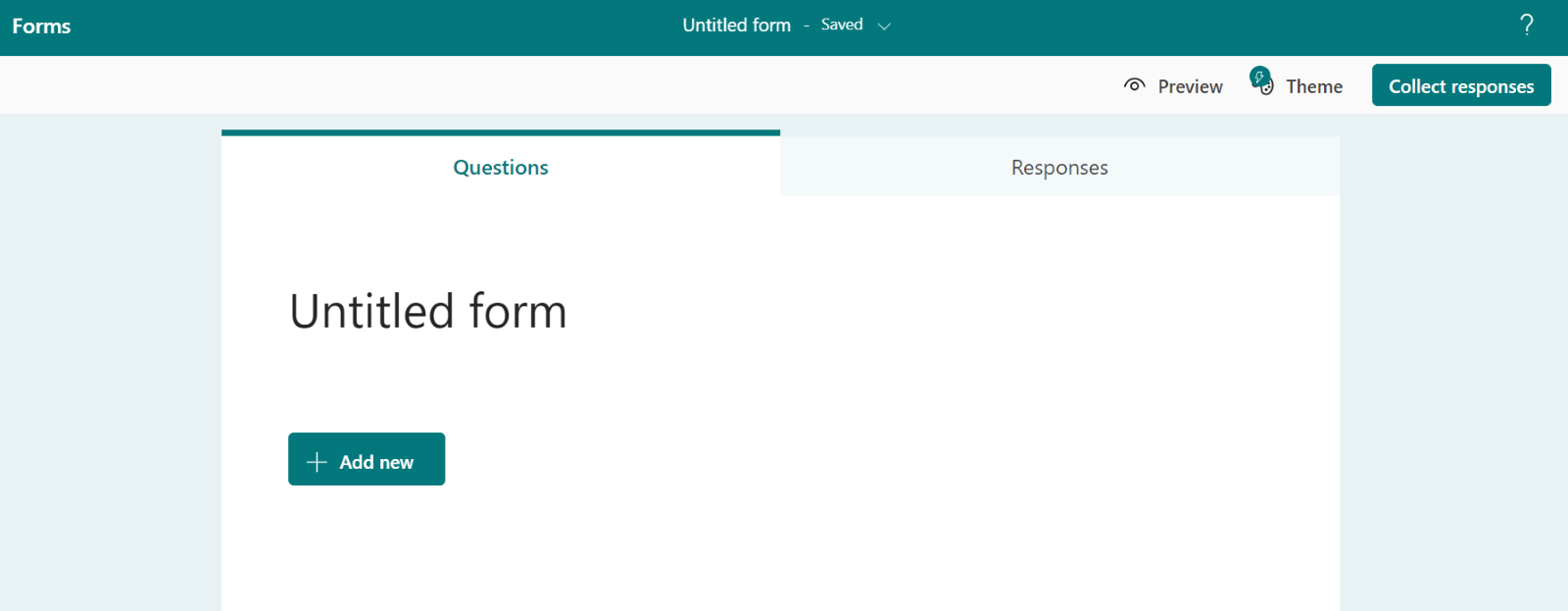
**Connectors required for Jira-AzureDevops Integration:**



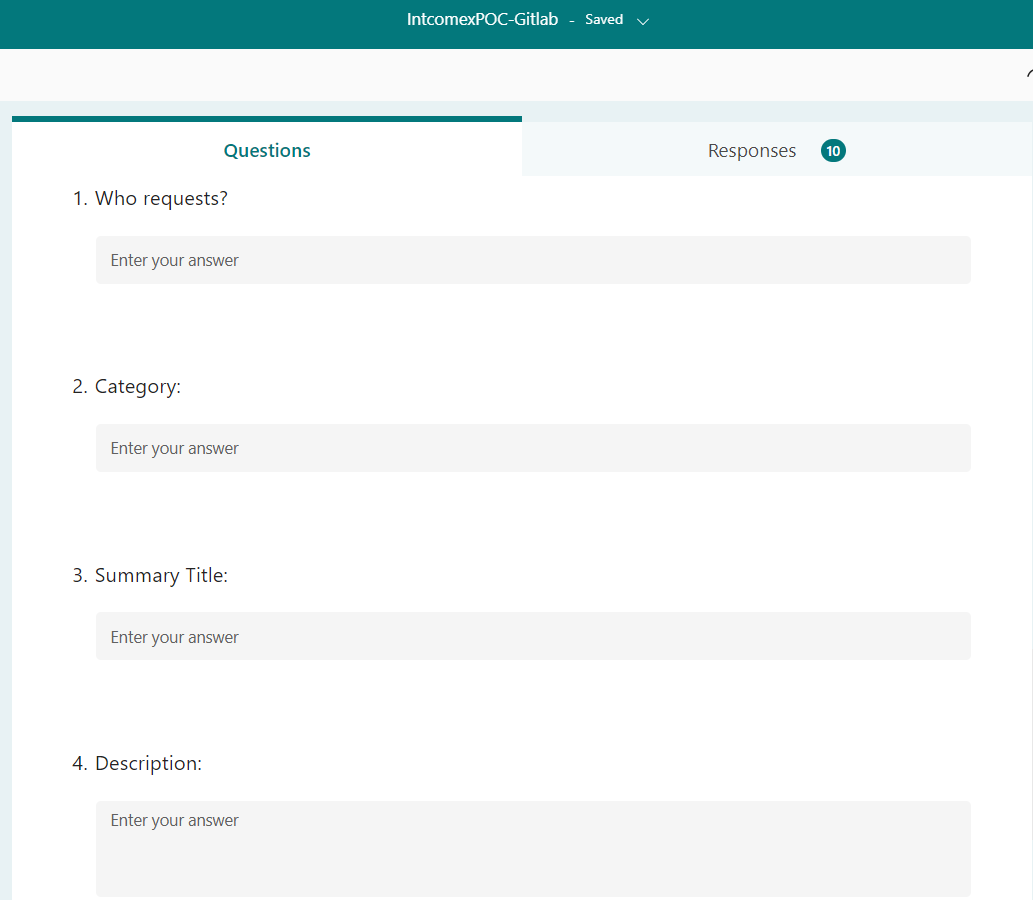
**JIRA-GITLAB INTEGRATION**

**Microsoft forms:**

* Take a new Microsoft form
* Add required fields as per the requirement



* Fill the form
* Submit the response

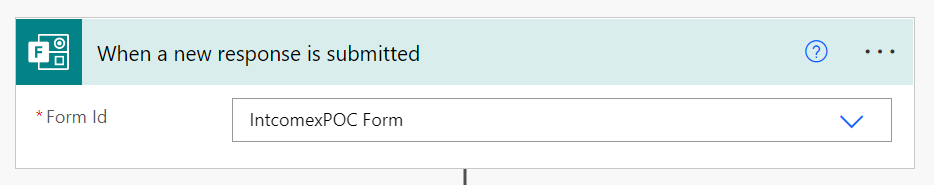


**Power Automate flows: Jira-Gitlab**

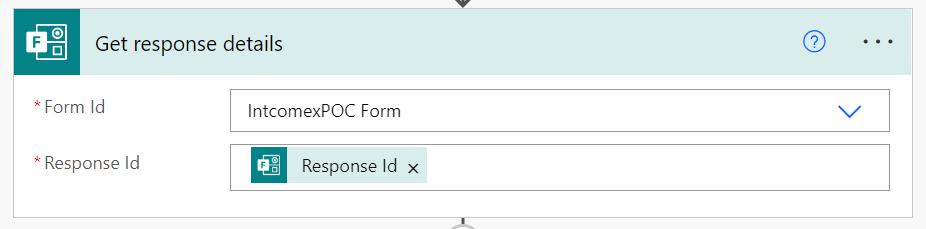
1. **Create an issue in Jira after the form is submitted and after the approval of Jira Issue, flow creates an issue in Gitlab:**

* When a new response is submitted, the flow will create an issue in Jira.
* It will send an email for approval.
* It delays the flow for scheduled time and after the approval, flow will create an issue in Gitlab.

**The following steps explain the flow:**

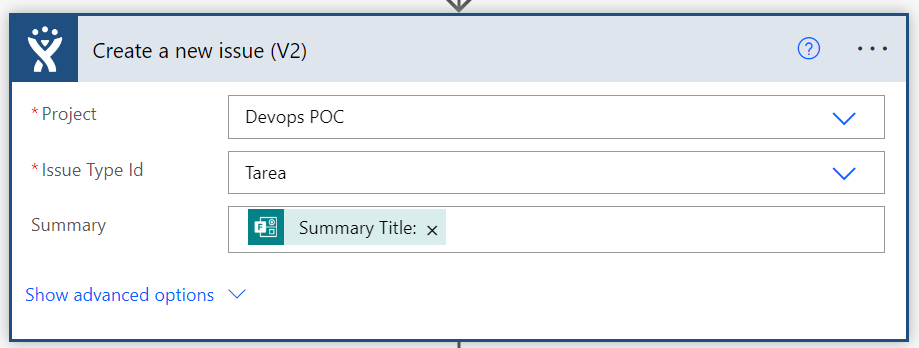


This operation triggers a flow when a new response is submitted.

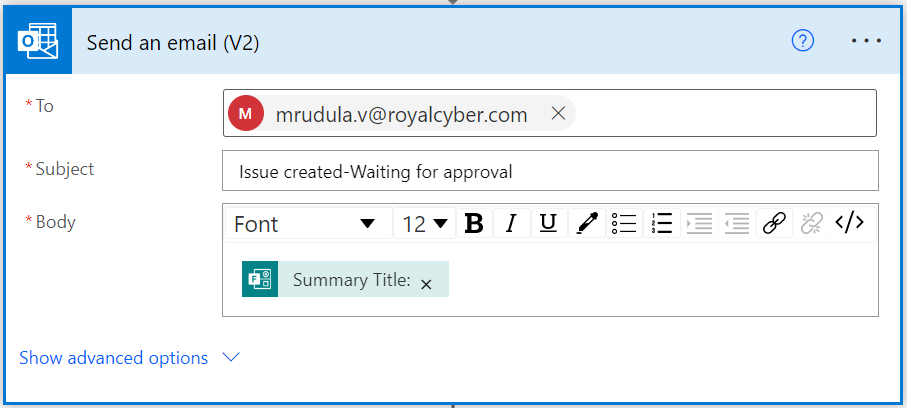


This action retrieves a form response, the Response Id takes the response when a form is

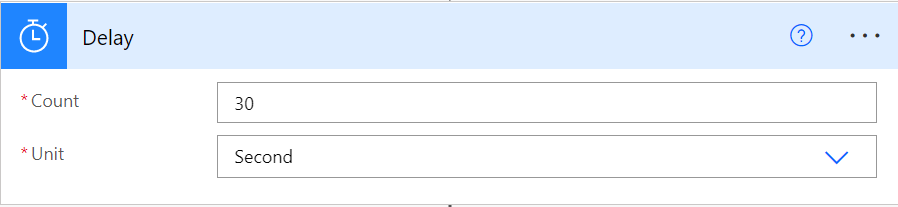
submitted.



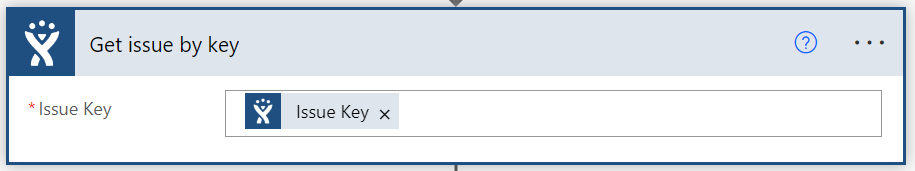
This operation is used to create a new issue. We can select the project and Issue type from Jira, since it is connected by establishing the connection between Power Automate and Jira. Summary takes the dynamic response from the form submitted.



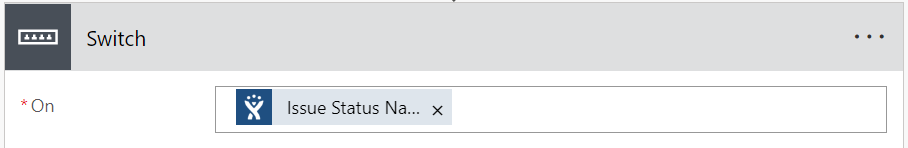
This operation sends an email to the approver, notifying that issue has been created and waiting for approval. Body of the email contains the dynamic content from the form response.



This operation delays the flows to review the Issues in Jira. We can define the count and unit as per the requirement.

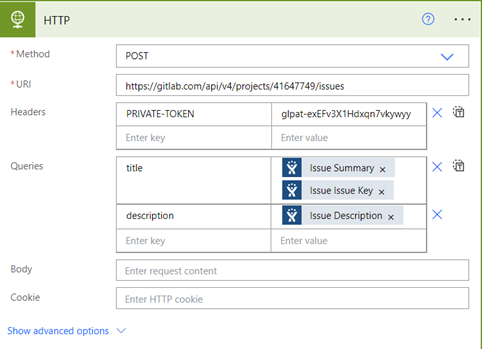


This operation is used to retrieve the issue object for a given issue Key.



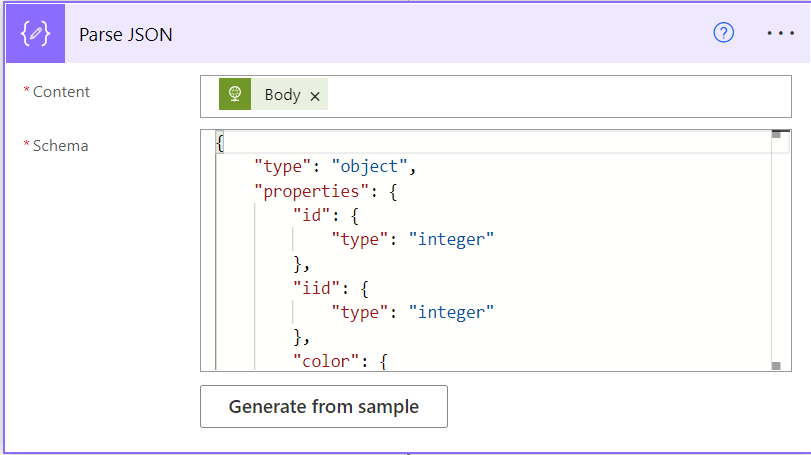
The switch case take the status of the Jira issue and we can create actions for different cases.

When Issue status name is Aprroved, the HTTP action will create an issue in GitLab.



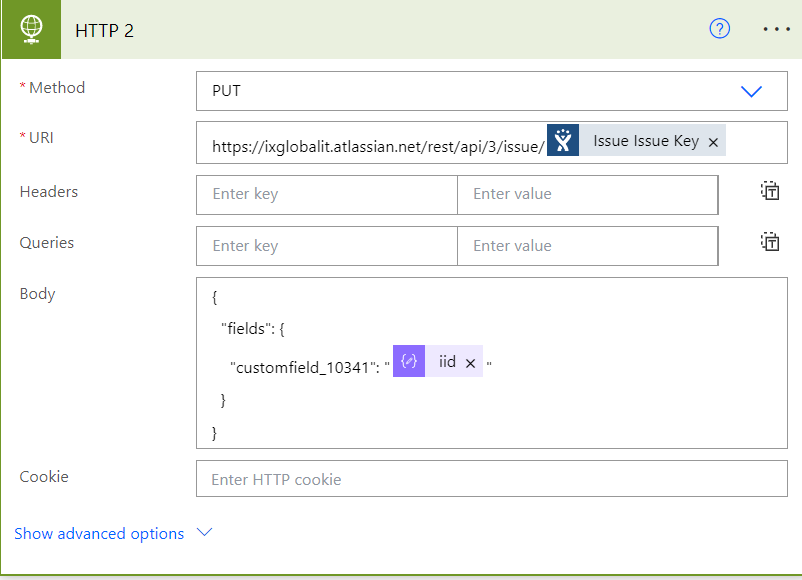
This Connector will create a new issue in GitLab by querying the issue summary, key and description. Here the POST method creates issue and URL provided is the webpage URL, where we need to create the issues in Gitlab. In headers field, we will provide Private-token of Gitlab.

This Connector will create a new issue in GitLab by querying the issue summary, key and description.



By the output of the above step and using the contents of the body to get “iid” which is the issue id of GitLab.

**HTTP Connector:**



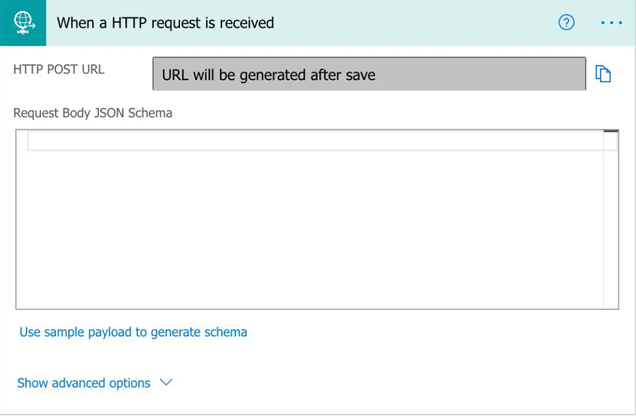
With the Parsed data we get the GitLab issue id and that need to be assigned in the Jira Issue Custom field by using an HTTP connector and providing the URL of the Jira with the issue key.

1. **Recurrence flow-Gitlab (for the issues which are yet to approve):**

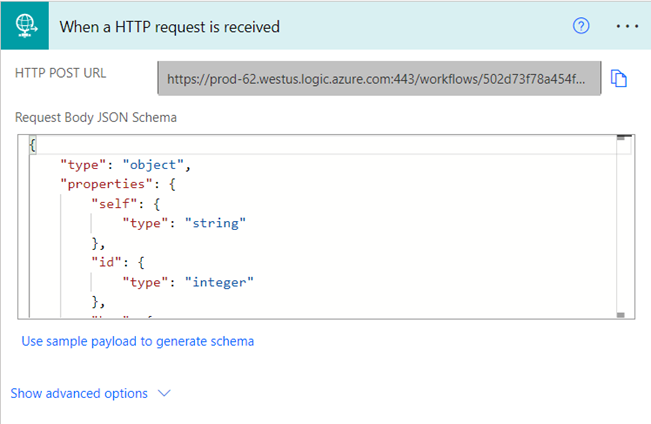
* The flow will run for scheduled intervals and checks whether any issue is updated.
* If the issue is approved and it will create an issue in Gitlab
* Updates the issue id in Jira issue.

Recurrence flow will be used as a scheduler to create a GitLab issue if there is a delay in Jira approval and assigns the Gitlab issue id in the Jira Custom Field.

When an HTTP request is received:

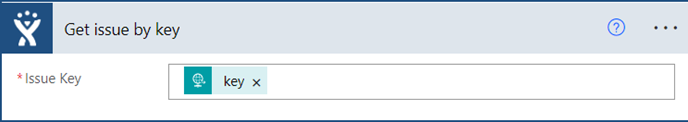


When first adding the ‘When an HTTP request is received’ trigger, to a flow it starts with an HTTP POST URL informing us that the URL will be generated after the Flow has been saved. This means that initially creating the Flow will not allow us to be able to provide/use the URL that is required to trigger the Flow.



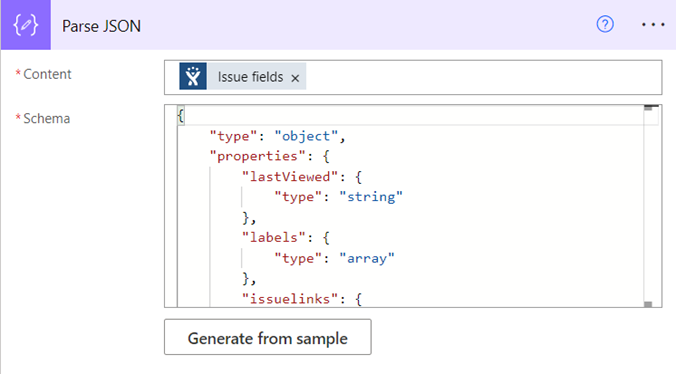
After entering JSON Schema into the box and hit done, the schema will be created and displayed in the ‘Request Body JSON Schema’ section as shown above. The above step retrieves the data from Jira.

Get Issue by Key:

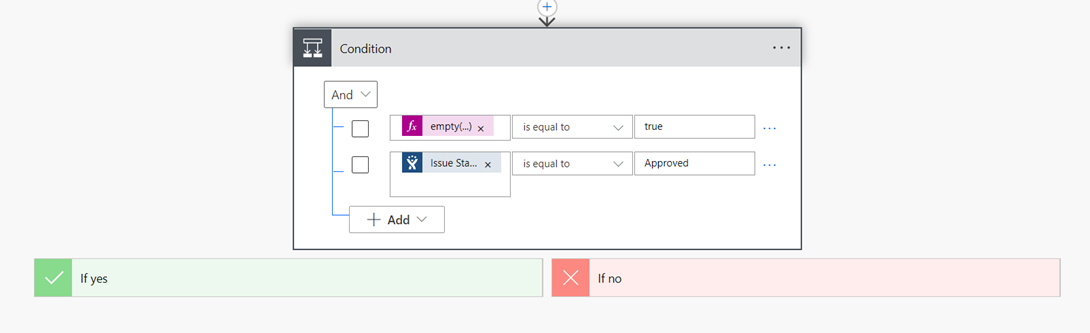


This is a Jira Connector, which gives the data about the Jira Issue Key from the key which is passed from the HTTP request connector.

Parse JSON:



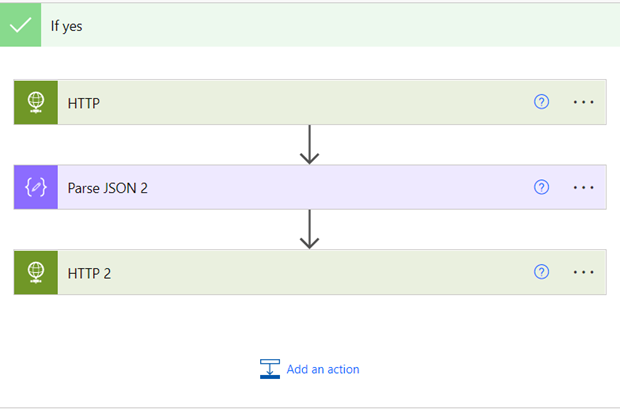
In this step, Fields of Jira Issue will be parsed and gives the data of each field that is associated with a Jira Issue Key.

Condition:

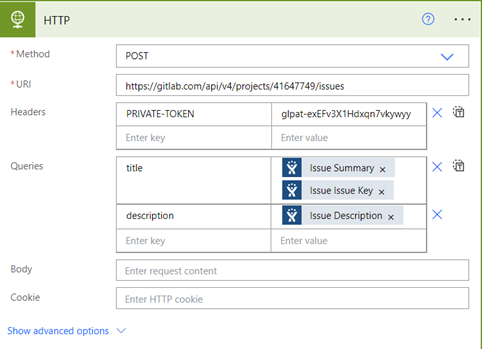
Then, there are two conditions these 2 must be validated.

1. Empty (): This operation is to check whether any value is assigned or not, this should be true.
2. Jira Issue Status Name: This field is to check whether the status is in the approved state or not.

If the condition Yes then, there are 3 steps as follows:

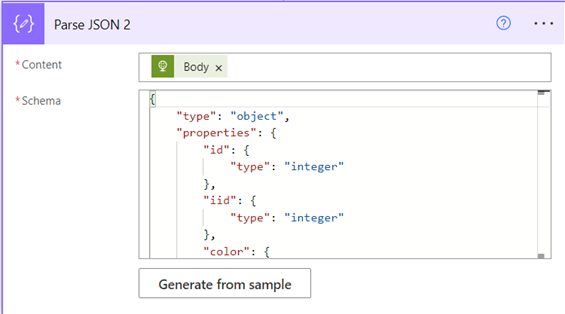


HTTP Connector:



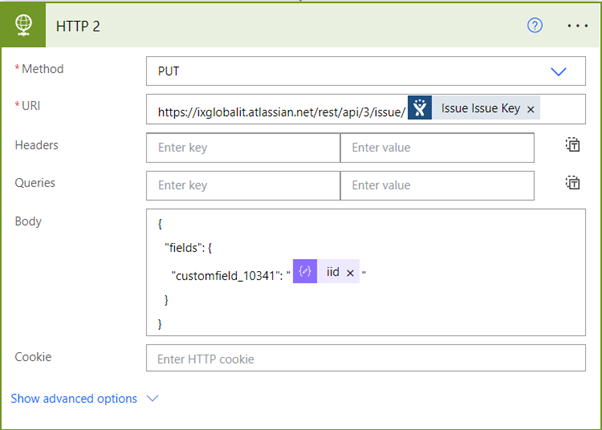
This Connector will create a new issue in GitLab by querying the issue summary, key and description, these details will get from the Parse JSON step.

Parse JSON 2:



Again, by the output of the above step and using the contents of the body to get “iid” which is the issue id of GitLab.

HTTP 2:



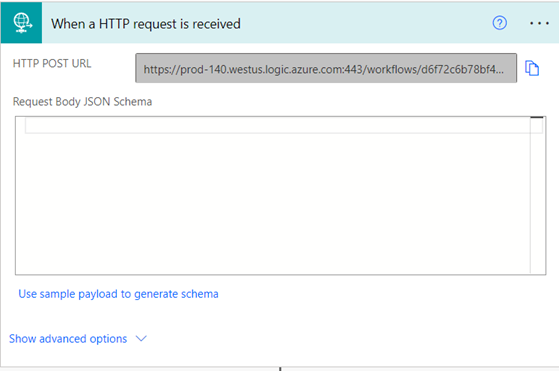
Finally, with the Parsed data we get the GitLab issue id and that need to be assigned in the Jira Issue Custom field by using an HTTP connector and providing the URL of the Jira with the issue key.

1. **When issue is updated in Gitlab, the flow updates Jira issue:**

* When an issue is updated (the state changes) in Gitlab, the flow will edit the status of Jira via HTTP request.

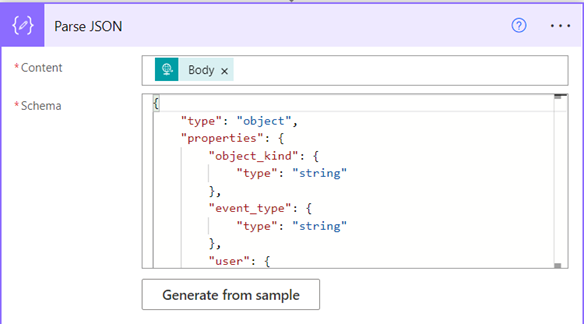
Whenever there is an update to Gitlab Status then it will automatically trigger this flow and reflects the same in the Jira issue status name.

When an HTTP request is received:



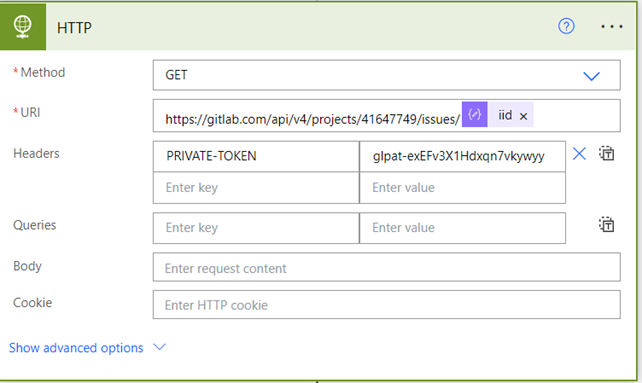
Adding the ‘When an HTTP request is received’ trigger, means this API call will get automatically triggered whenever there is an update in the Jira issue status.

Parse JSON:



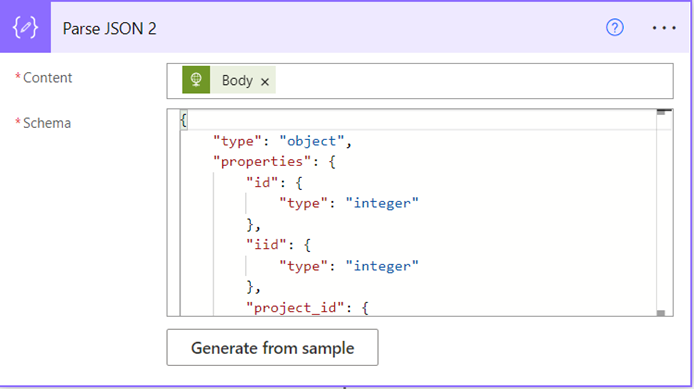
Parse JSON is the data operation, where it runs and generates the data from the HTTP request as this will help us to give the issue id of GitLab.

HTTP Connector:



This HTTP connector gets triggered and returns the GitLab issue by the GET method and the id of the Jira Issue.

Parse JSON 2:

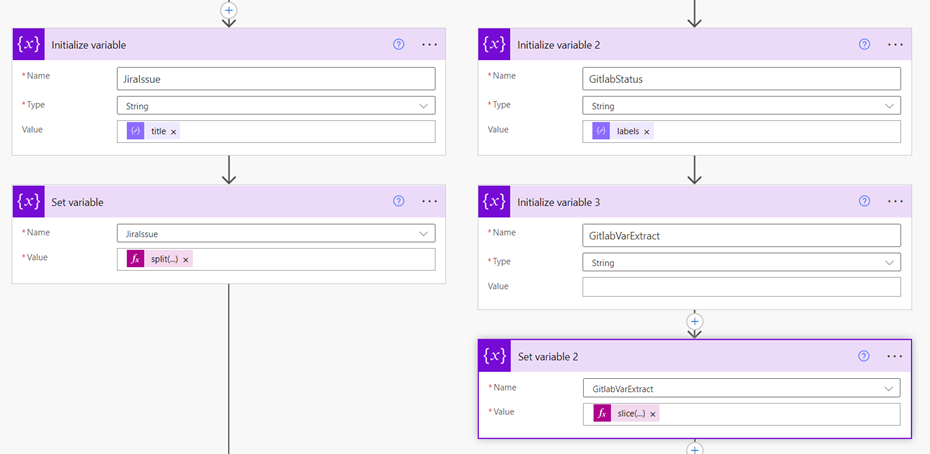


Again, to get the desired field where the status is updated, we need to parse the data in a JSON format from the API call.

Compose:



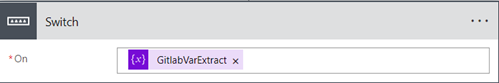
Compose is another data operation, which allows us to do some functions and get the Issue title as the issue title got the Jira issue id.



At this step, we get the Jira Issue id and the label details where the GitLab issue status is associated.

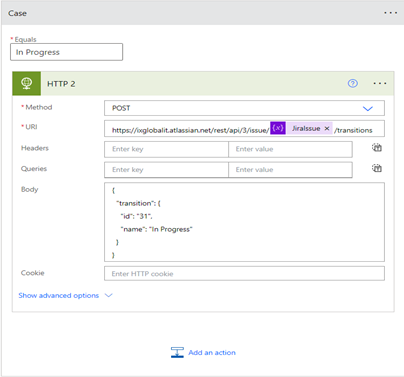
* To get the Jira Issue id, we use the split function.
* To get the Gitlab Issue status, we need to extract the data from the label and need to assign it to proper issue id.

Switch:



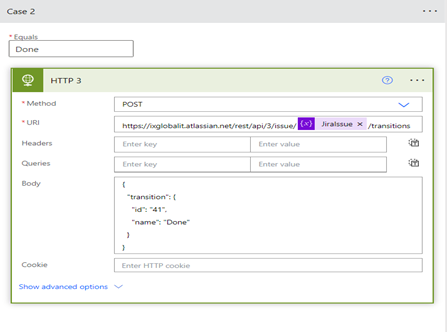
On the GitLab Issue Status, there are 4 cases as follows and update the same in Jira Issue:

Case 1: If GitLab status is “In Progress”

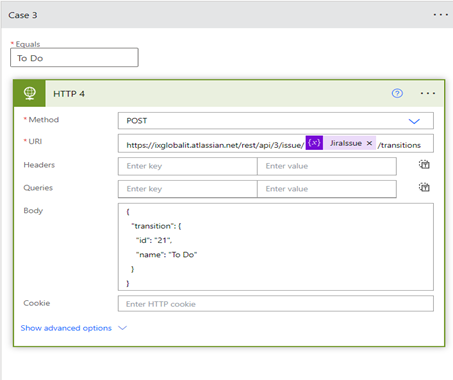


In this case, when the “GitLabVarExtract” of the work item is “In Progress”, the “HTTP” operation updates the status of the Jira issue by the POST method. The URL here used is for transitioning the Jira issue. The body is in Json format, the “JiraIssue” indicates what would be the id of the Jira issue.

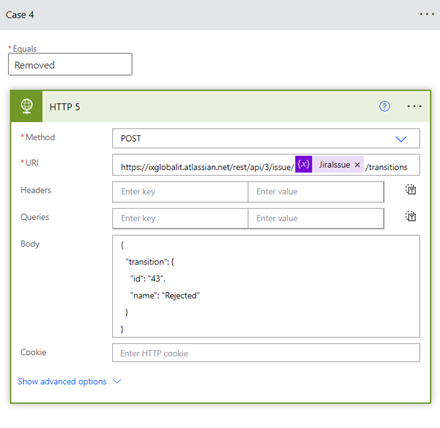
Case 2: If GitLab status is “Done”



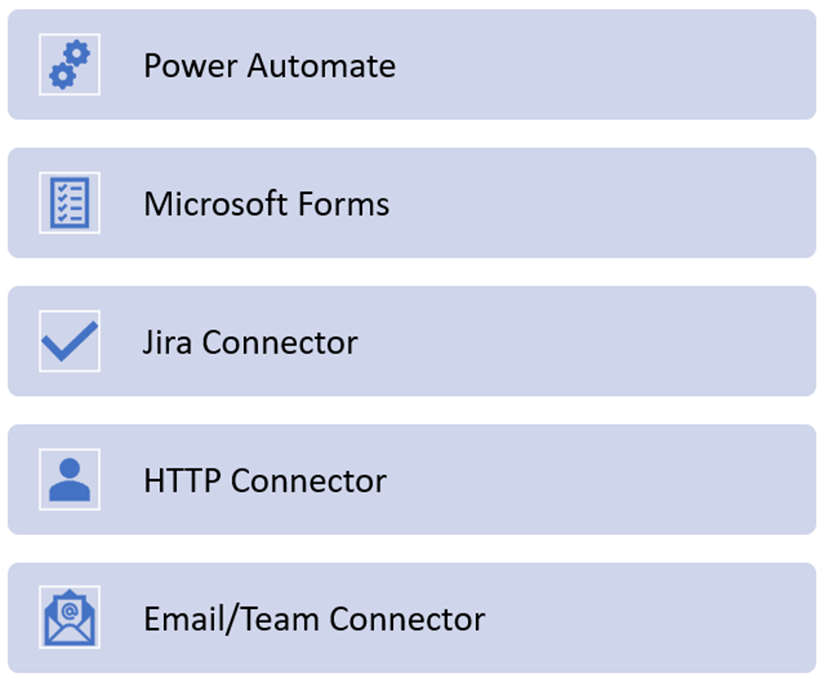
Case 3: If GitLab status is “To Do”



Case 4: If GitLab status is “Removed”



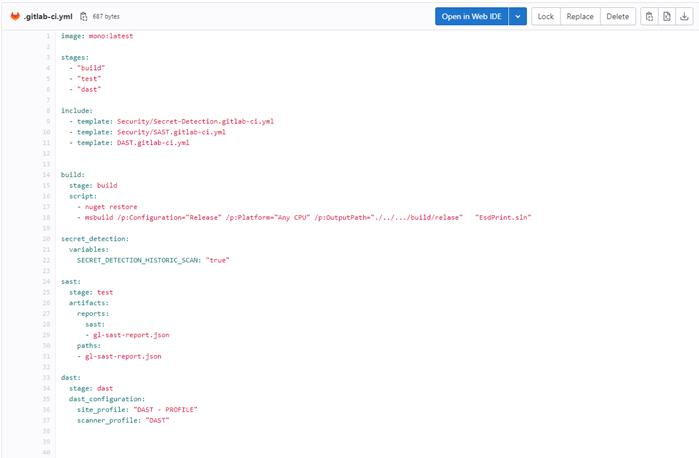
**Connectors required for Jira-Gitlab Integration:**



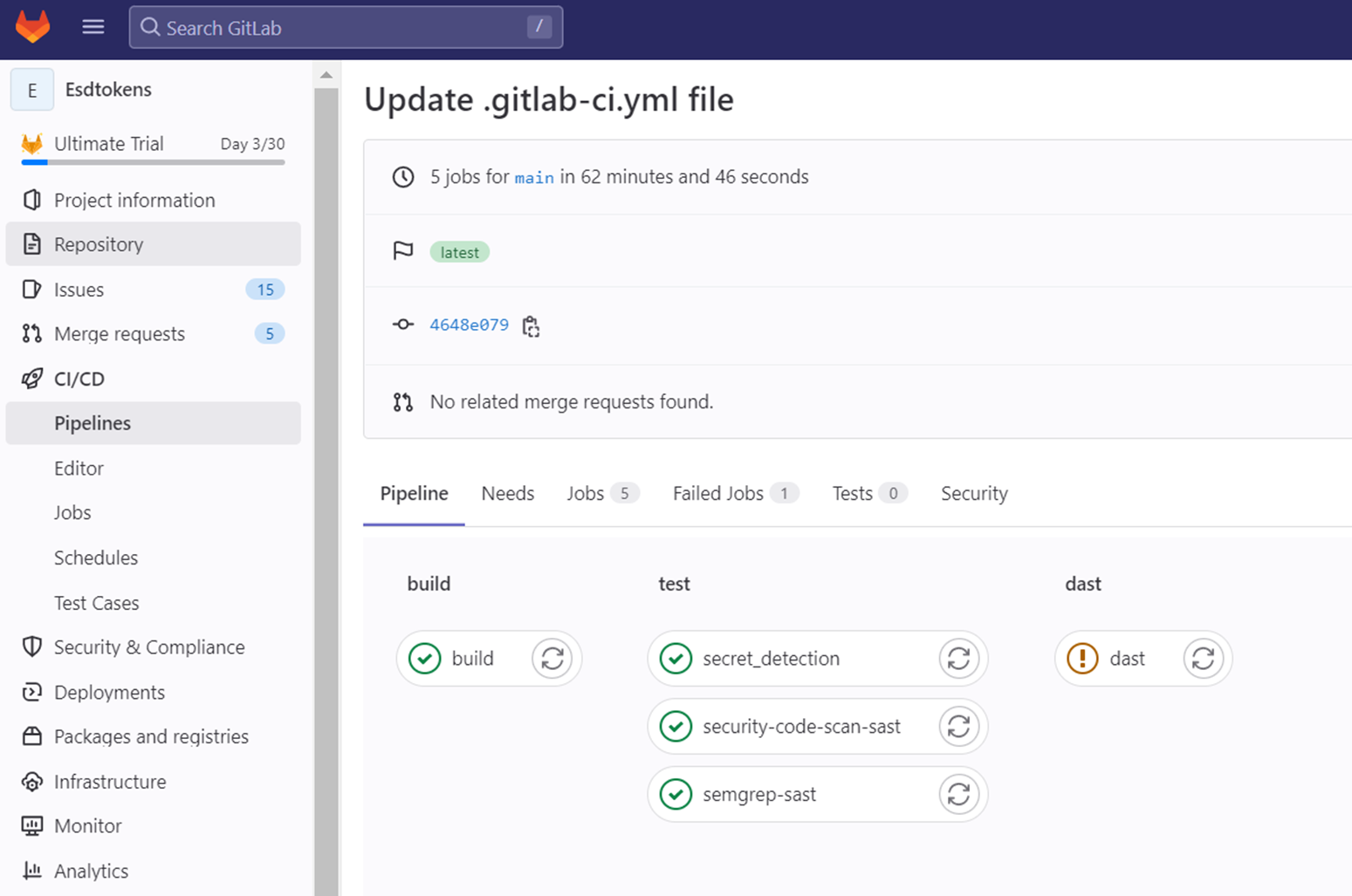
**GITLAB CI-CD**

**Implementation of GitLab CI/CD:**

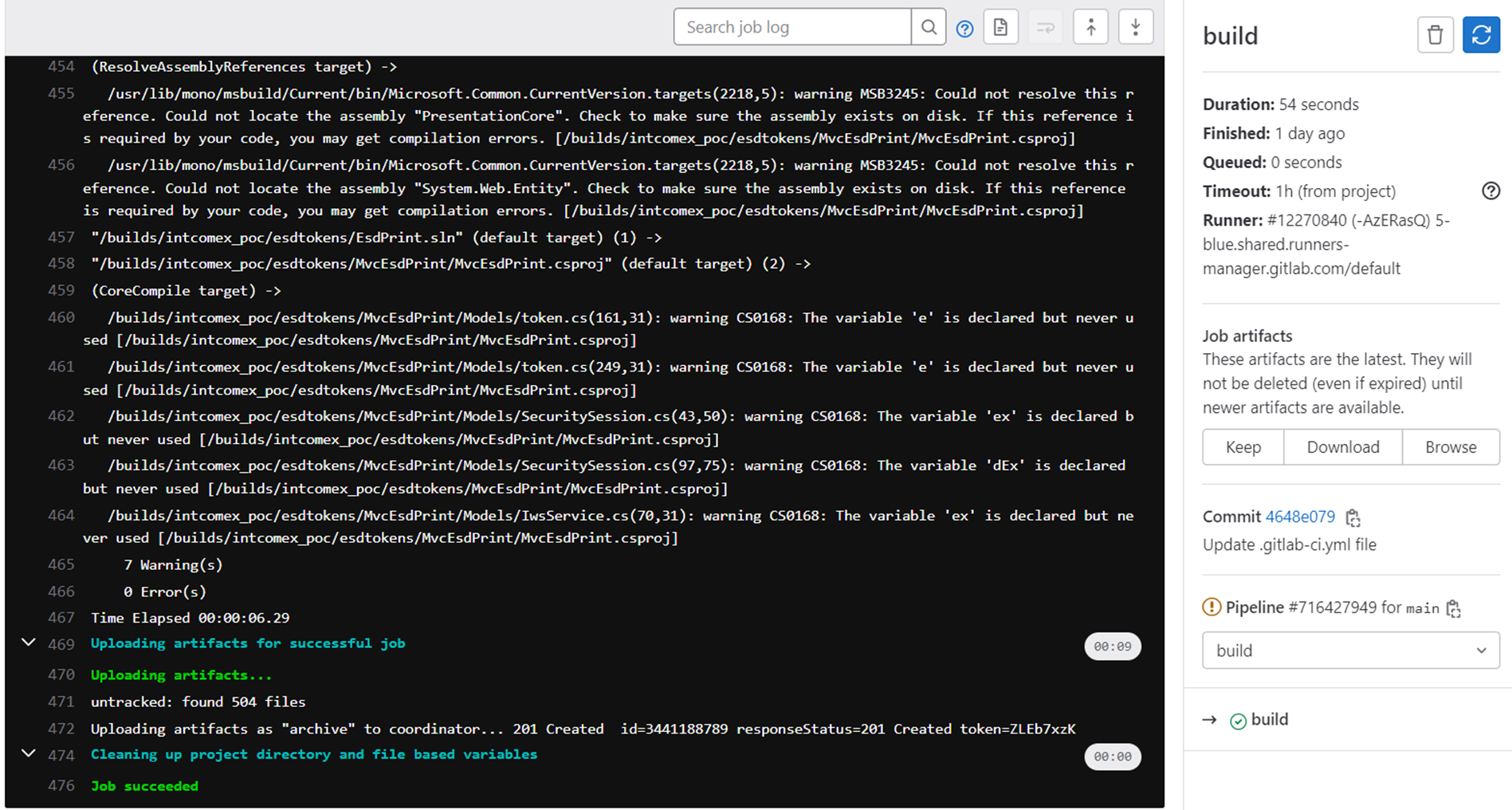
* Setting up the GitLab Account
* Adding SSH key for cloning the project
* Registering Runner to run the Pipelines
* Configuring yaml file to develop CI-CD
* Implementing Security Scans to protect our code from vulnerabilities
* Validating the Deployment



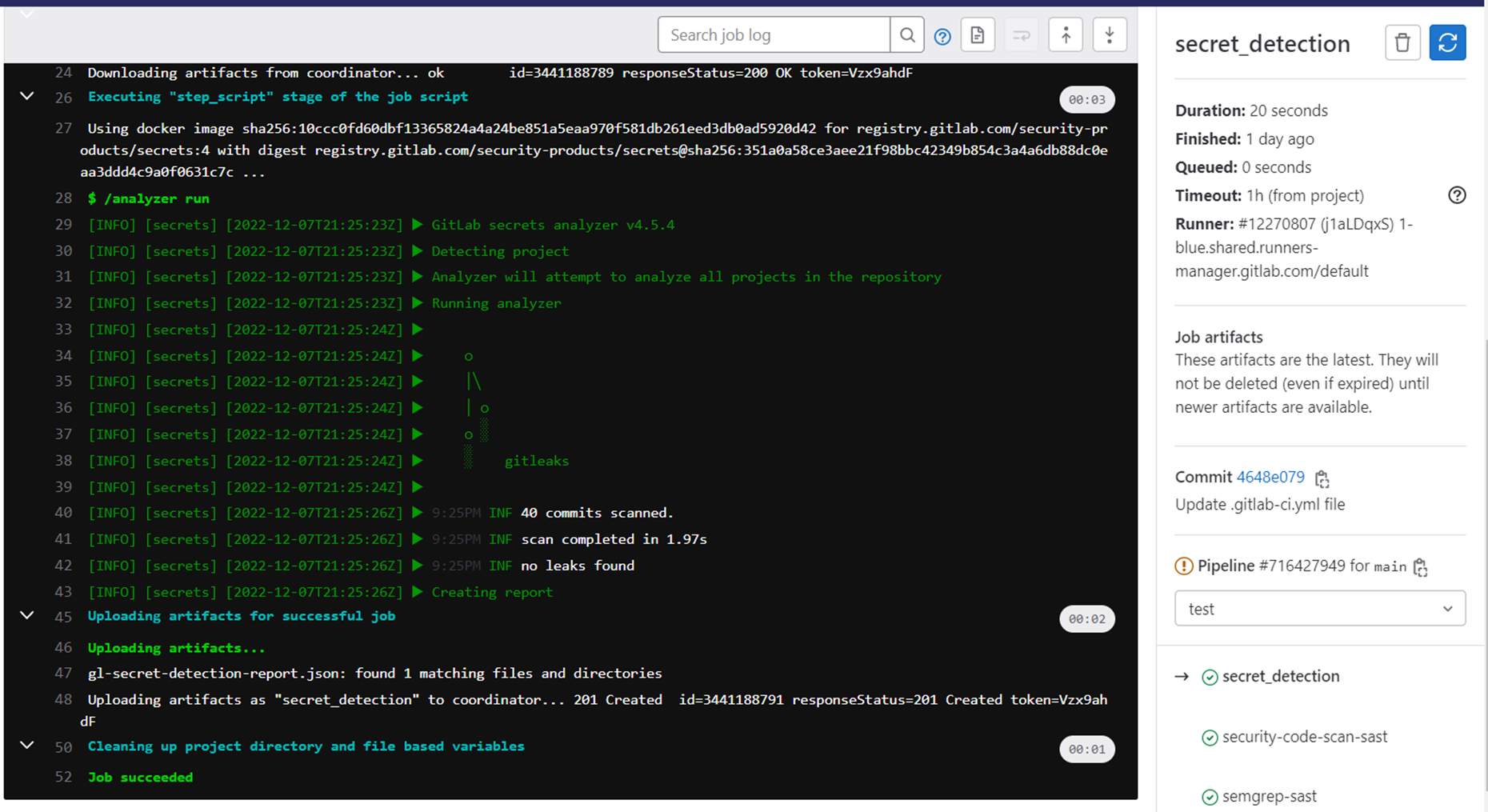
CI-CD yaml file for GitLab



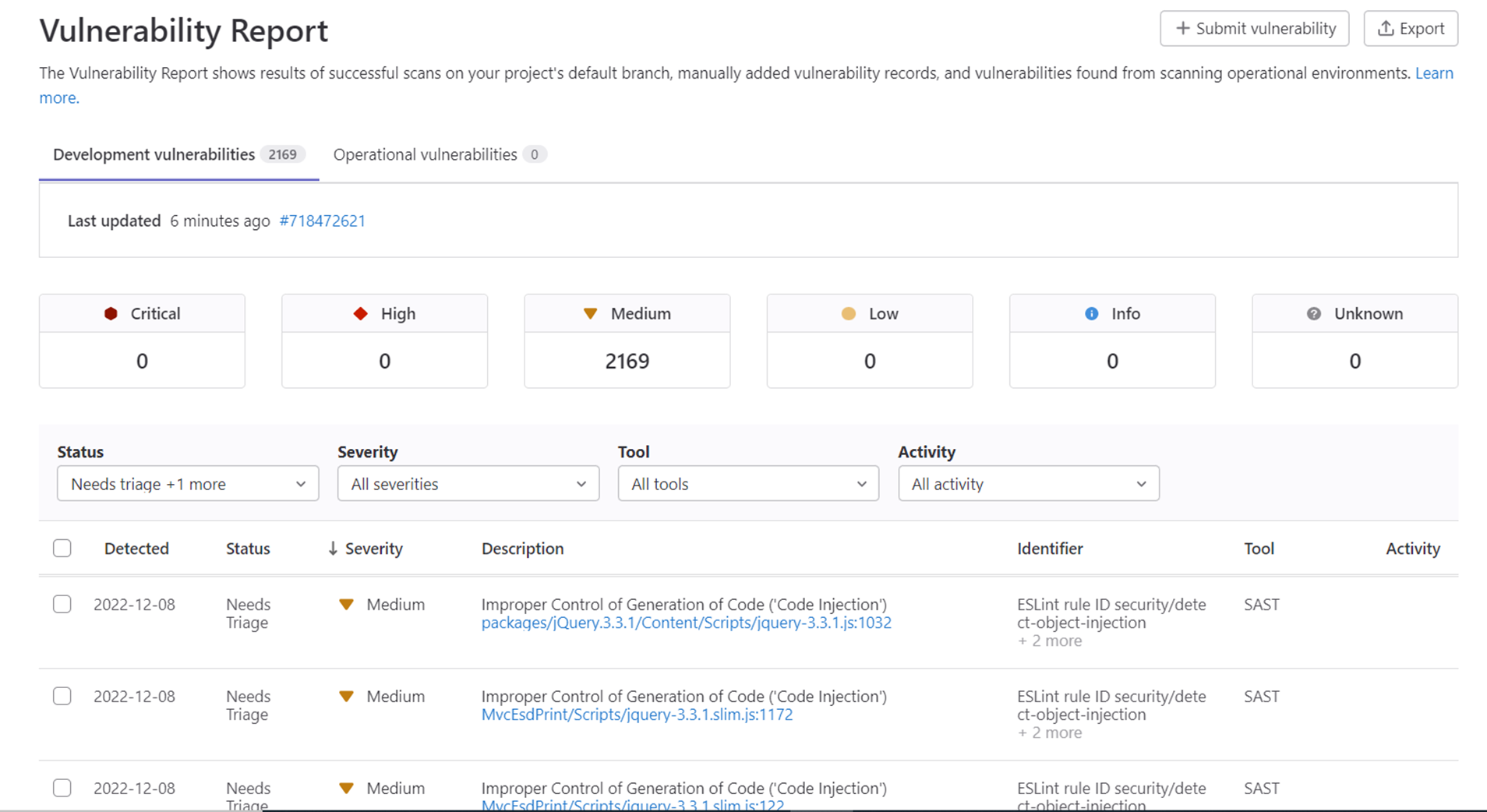
Pipeline Stages in CI-CD



Build is Succeeded



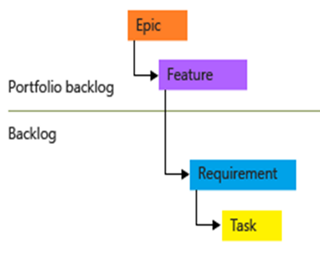
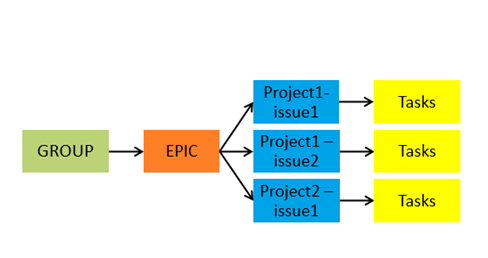
Security Scans run Successfully



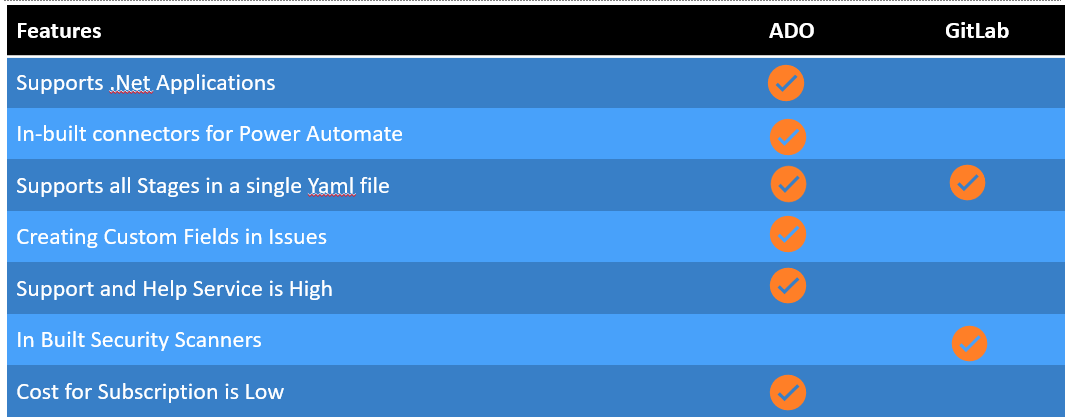
Vulnerability Report for all Security Scans

**Epics Workflow**

**AzureDevops:** **GitLab:**

**AzureDevops vs GitLab Features**



Thank you