Lab 6.1 - Setting Up Event Triggers with Argo

In this chapter, we begin our hands-on journey with Argo Events, starting from setting up its basic components and triggering simple workflows with HTTP requests. After a theoretical overview, the first lab focuses on practical implementation, allowing users to familiarize themselves with Argo Events by using simple curl commands to initiate workflows. Building on this foundational knowledge, the chapter advances to a lab that integrates Argo Events with an external system, Apache Pulsar. This step-by-step exploration not only enhances understanding of Argo Events in real-world scenarios but also demonstrates its capability to seamlessly integrate with other technologies, showcasing its versatility in managing event-driven architectures in cloud-native environments.

Objective

- Set up a laboratory environment that demonstrates the integration and functionality of Argo Events and Argo Workflows.
- Install and configure the necessary components to trigger a workflow through an event.

Prerequisites

- A Kubernetes cluster is available.
- kubectl is installed and configured to communicate with your Kubernetes cluster.
- Internet access for downloading necessary files and packages.

Install and Use Argo Events

1. Installing Argo Workflows

Because we trigger a workflow we need to install Argo Workflows first. For that create the respective namespace “argo” and apply the installation YAML to the new namespace:

```
kubectl create namespace argo
```

```
kubectl apply -n argo -f
https://github.com/argoproj/argo-workflows/releases/download/v3.5.2/in
stall.yaml
```

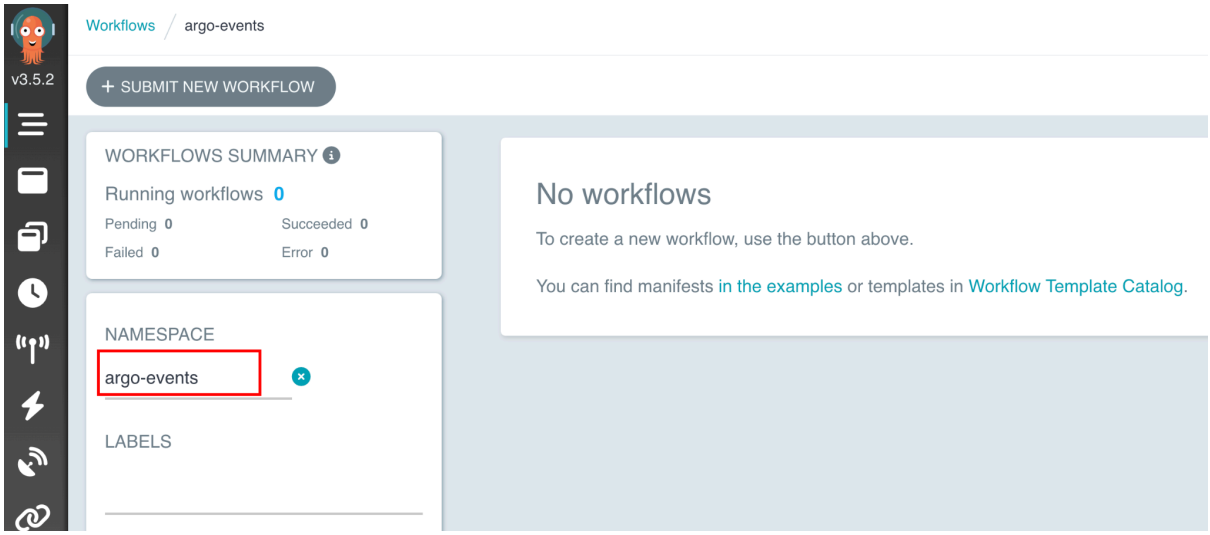
In the next step, we want to access Argo Workflows through the UI. For that we need to change the authentication mode of the Argo Server to “server”. It ensures that the server can properly authenticate requests:

```
kubectl patch deployment argo-server --namespace argo --type='json'
-p='[{"op": "replace", "path":
"/spec/template/spec/containers/0/args", "value": ["server",
"--auth-mode=server"]}]'
```

Lastly port forward the Argo server deployment to access it on https://localhost:2746 locally:

```
kubectl -n argo port-forward deployment/argo-server 2746:2746
```

You should see the following screen:



Start screen of Argo Workflows

Make sure to select “argo-events” as the namespace.

2. Install Argo Events and the Needed Components

Creates a new namespace named 'argo-events', which is used to separate and manage the resources related to Argo Events. After that install Argo Events in your cluster by applying the installation manifest from the given URL:

```
kubectl create namespace argo-events
```

```
kubectl apply -f
https://raw.githubusercontent.com/argoproj/argo-events/stable/manifest
s/install.yaml
```

The next command applies a validating webhook for Argo Events. Validating webhooks are used to ensure that incoming requests to the Kubernetes API server are valid:

```
kubectl apply -f
https://raw.githubusercontent.com/argoproj/argo-events/stable/manifest
s/install-validating-webhook.yaml
```

For setting up a native EventBus in the 'argo-events' namespace, which handles event transportation in Argo Events, apply the configuration with this command:

```
kubectl -n argo-events apply -f
https://raw.githubusercontent.com/argoproj/argo-events/stable/examples
/eventbus/native.yaml
```

Next we need to define an EventSource configuration that listens for webhook events in Argo Events, apply the following configuration using this command:

```
kubectl -n argo-events apply -f
https://raw.githubusercontent.com/argoproj/argo-events/stable/examples
/event-sources/webhook.yaml
```

For the Sensor to properly interact with Kubernetes resources, apply the necessary RBAC policies:

```
kubectl apply -n argo-events -f
https://raw.githubusercontent.com/argoproj/argo-events/master/examples
/rbac/sensor-rbac.yaml
```

Similarly, apply RBAC policies for Workflows to ensure they have the necessary permissions in Kubernetes:

```
kubectl apply -n argo-events -f
https://raw.githubusercontent.com/argoproj/argo-events/master/examples
/rbac/workflow-rbac.yaml
```

Set up a Sensor to trigger workflows based on webhook events by applying this Sensor configuration:

```
kubectl -n argo-events apply -f
https://raw.githubusercontent.com/argoproj/argo-events/stable/examples
/sensors/webhook.yaml
```

Expose the event-source pod via port forwarding to consume requests over HTTP:

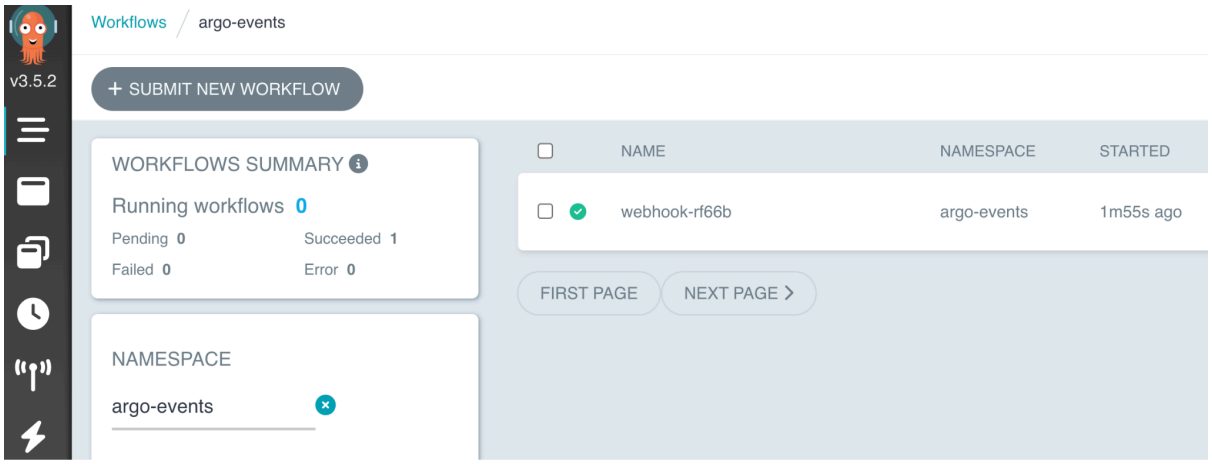
```
kubectl -n argo-events port-forward $(kubectl -n argo-events get pod
-l eventsource-name=webhook -o name) 12000:12000 &
```

Finally, simulate an external event that triggers the workflow. Send a test webhook event to the Event Source with this curl command:

```
curl -d '{"message":"this is my first webhook"}' -H "Content-Type:
application/json" -X POST http://localhost:12000/example
```

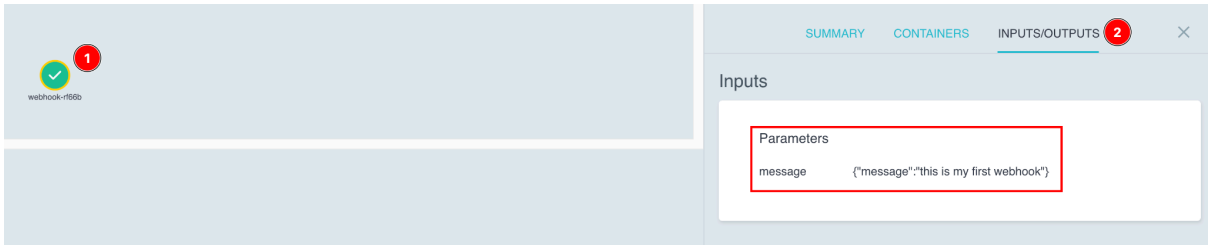
Refresh the UI of Argo Workflows. You should see a new workflow that is processing.

Refresh again after a few minutes and you should see the following screen with a completed workflow:



Workflow is shown in Argo Workflows UI

Click on the name of the workflow (e.g., webhook-rf66b). In the new window select the webhook (1), select Inputs/Outputs (2) and under parameters you see the message of the curl we have sent before.



Message of curl shown in the workflow

We successfully implemented a lab setup integrating Argo Workflows and Argo Events within a Kubernetes environment, demonstrating the efficiency of event-driven automation combined with workflow management.

The key achievement was using a webhook event to initiate an Argo Workflow, showcasing the system's capability to respond to events dynamically.

In our next lab, we plan to extend this setup to connect with an external system.