**Metaflow on Argo Workflows – Local Setup (Minikube + MinIO)**

# Part A – Setup Summary

## 1. Introduction

This document describes a local setup for running Metaflow workflows on Kubernetes using Argo Workflows. This setup is not production-ready but mirrors key production patterns for validation.

Metaflow pipelines are triggered from a local machine using the Metaflow CLI, while execution is handled by Argo Workflows inside a Kubernetes cluster. Workflow artifacts are stored in MinIO (S3-compatible storage), and execution metadata is tracked by the Metaflow service.

## 2. Purpose of the Setup

Validate that Metaflow workflows can be executed on Kubernetes using Argo Workflows in a local Minikube environment.

Goals:

* Confirm correct integration between Metaflow and Argo Workflows
* Enable containerized execution of workflow steps as Kubernetes pods
* Validate artifact storage (MinIO) and metadata tracking (Metaflow service + PostgreSQL)
* Provide visibility into workflow execution through Argo UI and Metaflow UI

## 3. High-Level Architecture

At a high level:

* Metaflow pipelines are defined and triggered from a local machine (Metaflow CLI).
* Metaflow service generates Argo WorkflowTemplates.
* Argo Workflows executes each pipeline step as a Kubernetes pod.
* Artifacts are stored in MinIO (S3-compatible).
* Run metadata is tracked and viewed via Metaflow UI and Argo UI.

## 4. Technology Stack

* Metaflow
* Argo Workflows
* Kubernetes (Minikube)
* MinIO (S3-compatible storage)
* PostgreSQL (Metaflow metadata)
* Helm, kubectl

## 5. Kubernetes Namespace Layout

* metaflow – Metaflow service, UI components, PostgreSQL
* argo – Argo Workflows controller and Argo UI
* minio – MinIO object storage
* default – Optional namespace for some services / executions (depends on deployment)

## 6. Workflow Execution Flow

1. A Metaflow pipeline can be triggered either from a local machine using the Metaflow CLI **or by submitting an existing WorkflowTemplate from the Argo UI**.
2. When triggered via the Metaflow CLI, the Metaflow service generates an Argo WorkflowTemplate for the flow.
3. Argo instantiates a workflow run from the WorkflowTemplate.
4. Each step of the workflow runs as an individual Kubernetes pod.
5. Steps execute according to defined dependencies; artifacts are stored in MinIO, and execution metadata is tracked by the Metaflow service.

## 7. Storage and Metadata Handling

Artifacts: stored in MinIO using S3 APIs.

Metadata: tracked by the Metaflow service and stored in PostgreSQL; surfaced through Metaflow UI.

## 8. Local vs Kubernetes Configuration Handling

Local CLI uses port-forwarded endpoints for MinIO and Metaflow service.

Workflow pods running in Kubernetes use in-cluster service DNS and credentials provided via Kubernetes Secrets.

Local-only environment variables are excluded from generated Argo WorkflowTemplates to avoid leaking localhost settings into pods.This separation avoids invalid localhost references inside Kubernetes pods.

## 9. Monitoring and Validation

* Argo UI: workflow list, DAG view, step logs
* Metaflow UI: flow list, run details, step metadata
* kubectl: pods, workflows, workflowtemplates validation
* MinIO UI: artifact buckets and objects

## 10. Current Status and Next Steps

Metaflow–Argo workflows run successfully in the local setup. Next steps include exploring event-driven workflow triggering using Argo Events with Kafka as the event source.

# Part B – Step-by-Step Runbook (Commands + Screenshot Placement)

## B1. Start Minikube and verify namespaces

Run:

minikube start  
kubectl get ns

**Screenshot: kubectl get ns output**



*Kubernetes namespaces used in the setup*

*This shows the different namespaces used in the cluster, including metaflow, argo and minio, which help logically separate components.*

## B2. Verify MinIO and set up port-forwarding (S3 API + UI)

Run:

kubectl get pods -n minio  
kubectl get svc -n minio



MinIO object storage pods

MinIO is used as the S3-compatible object store for workflow artifacts and intermediate data.

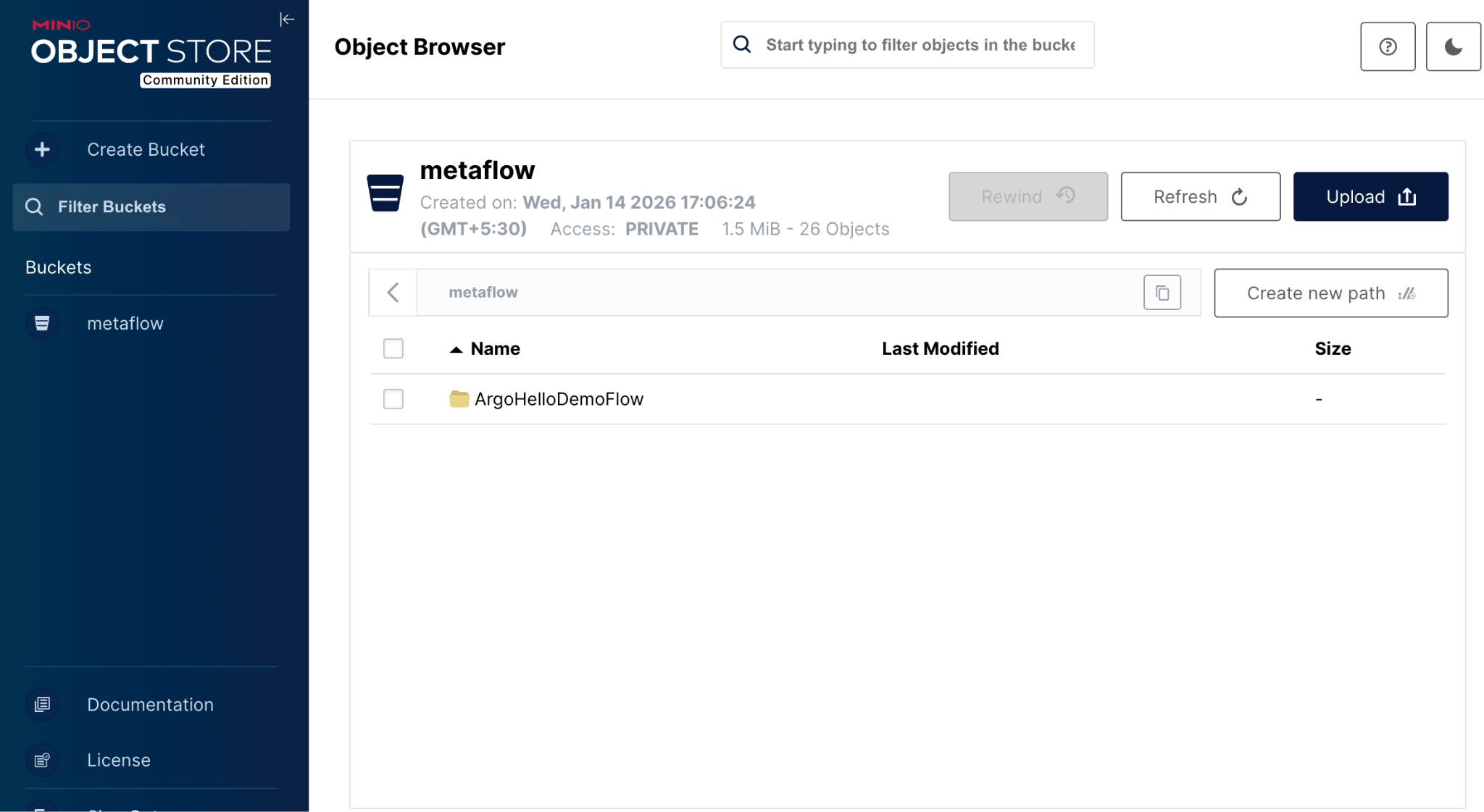
kubectl -n minio port-forward svc/minio 19000:9000  
kubectl -n minio port-forward svc/minio 9001:9001





MinIO UI: <http://localhost:9001>

**Screenshot: MinIO UI showing bucket (e.g., metaflow)**

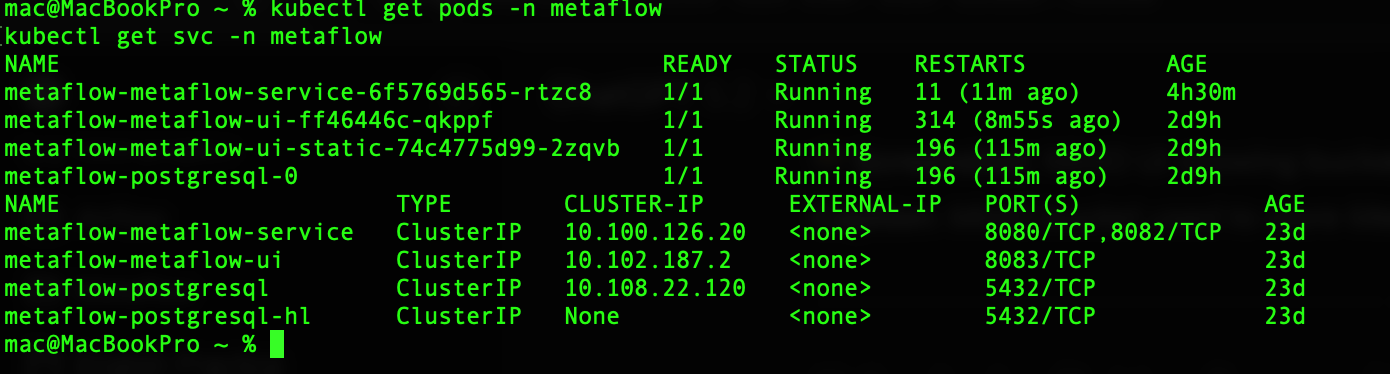


*MinIO bucket used to store Metaflow artifacts.*

## B3. Verify Metaflow services and set up port-forwarding

Run:

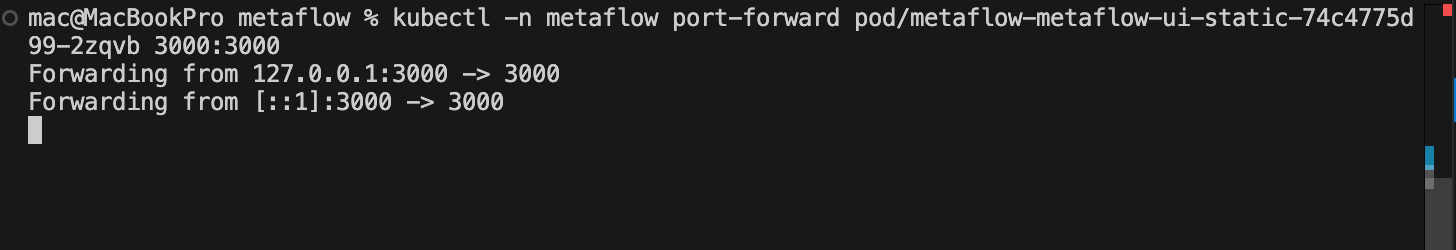
kubectl get pods -n metaflow  
kubectl get svc -n metaflow

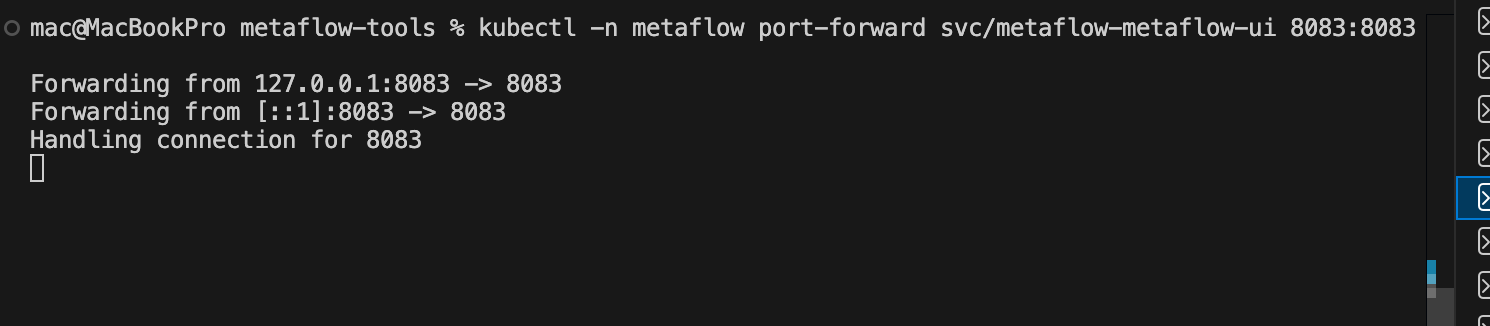


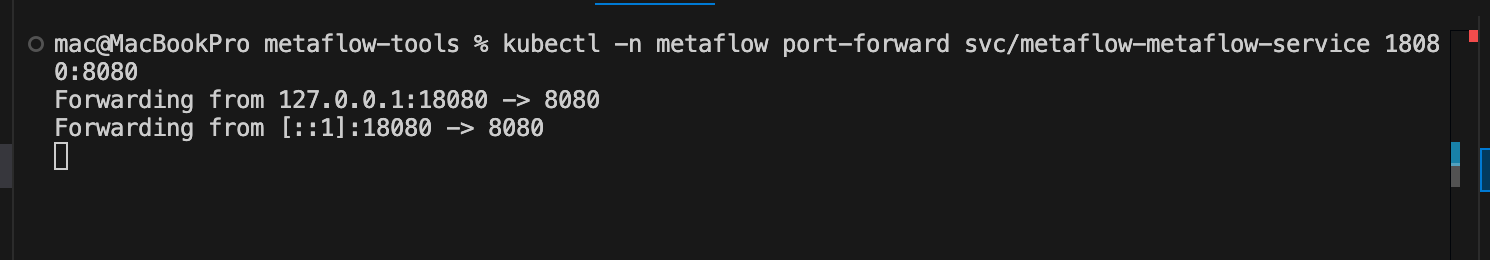
Metaflow service and UI pods

These pods run the Metaflow service, UI, and metadata components required to track workflow executions.

Metaflow service and UI pods  
kubectl -n metaflow port-forward svc/metaflow-metaflow-service 18080:8080  
kubectl -n metaflow port-forward svc/metaflow-metaflow-ui 8083:8083  
kubectl -n metaflow port-forward deployment/metaflow-metaflow-ui-static 3000:3000

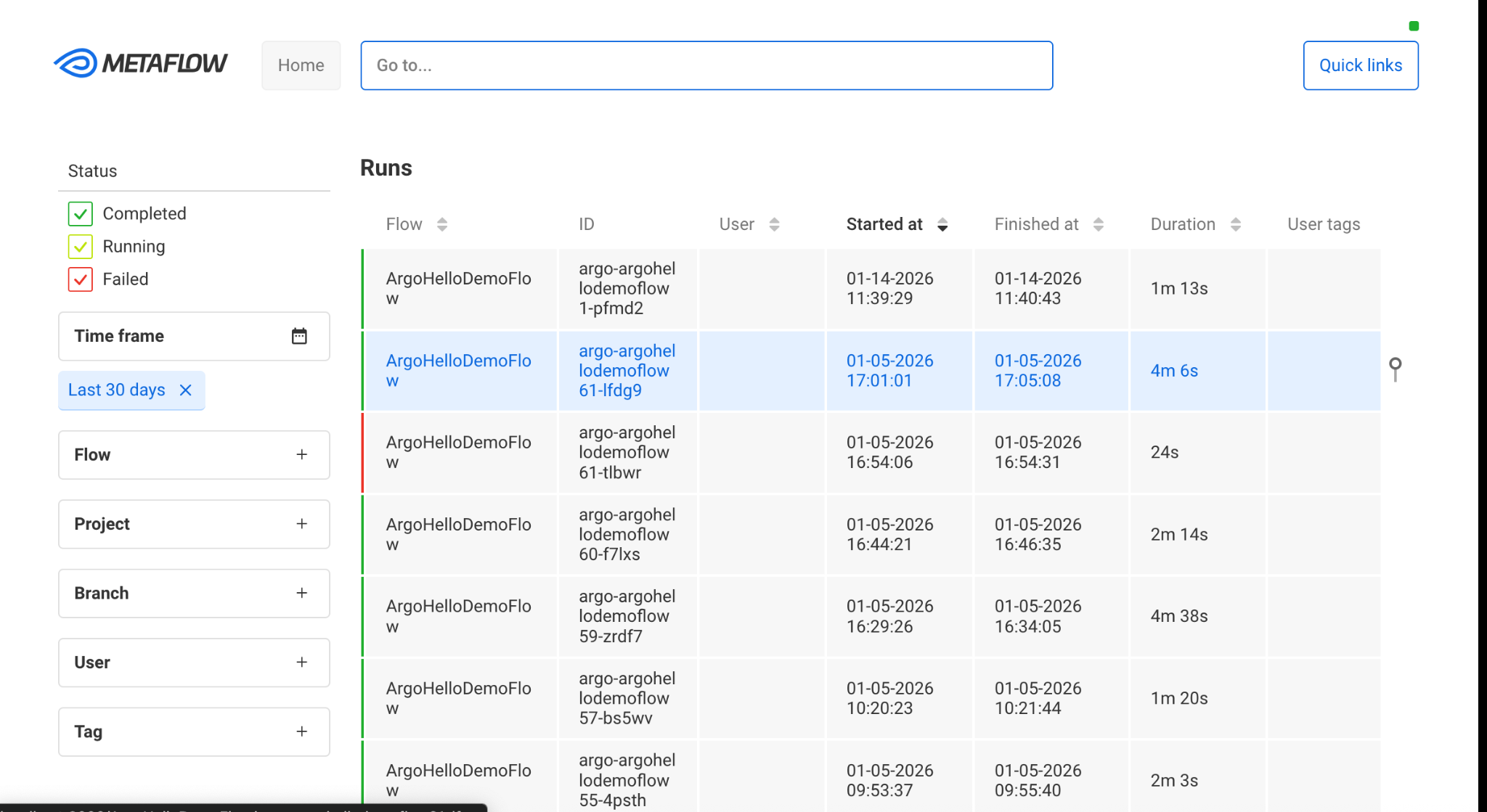






Metaflow UI: <http://localhost:3000>

**Screenshot: Metaflow UI home page**

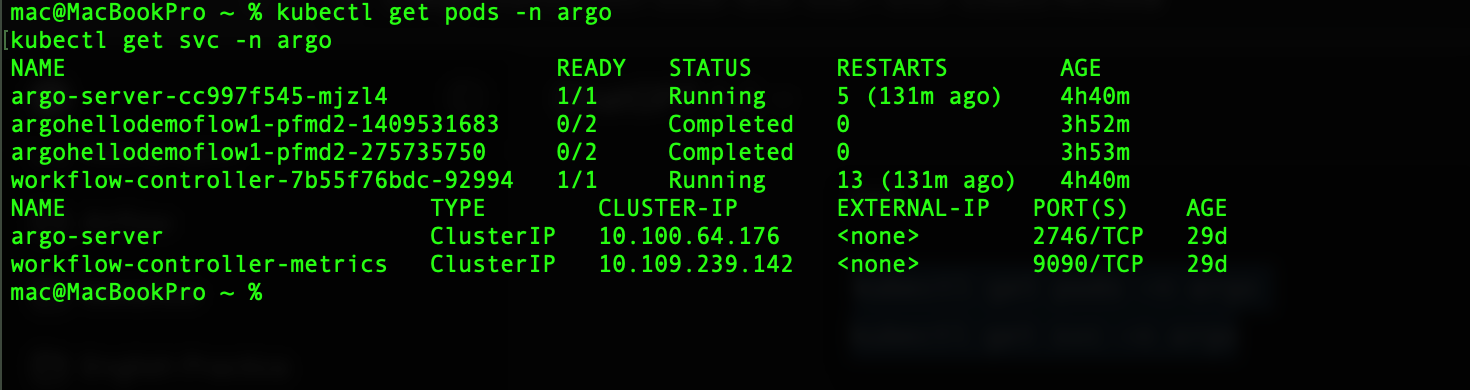


*Metaflow UI showing flows and recent runs.*

## B4. Verify Argo Workflows and set up Argo UI port-forward

Run:

kubectl get pods -n argo  
kubectl get svc -n argo



Argo Workflows controller and UI

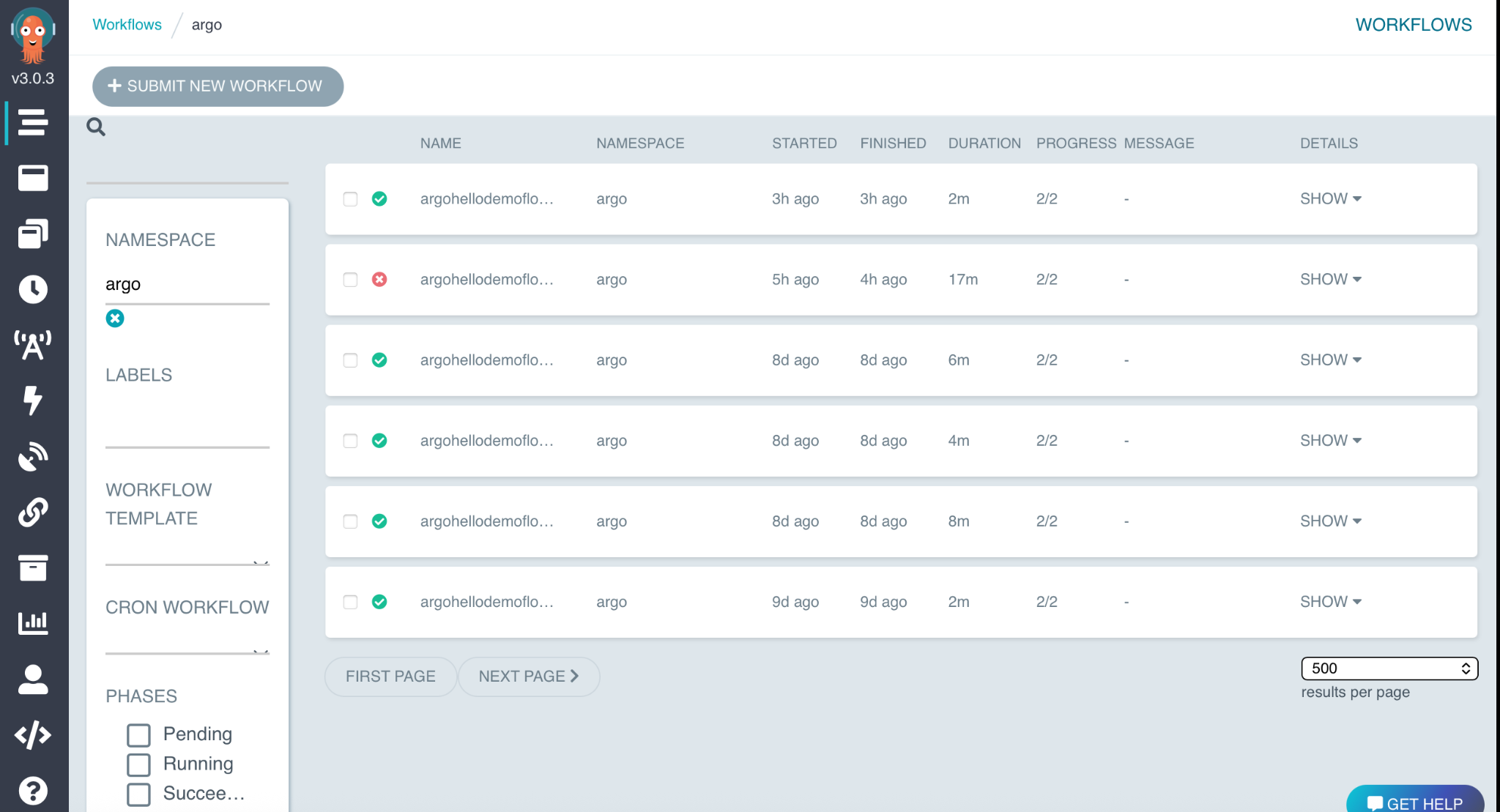
This shows the Argo Workflows controller responsible for executing workflows and the Argo UI for monitoring runs

kubectl -n argo port-forward svc/argo-server 2746:2746



Argo UI: http://localhost:2746

**Screenshot: Argo UI home page**



*Argo UI showing WorkflowTemplates and workflow runs.*

## B5. Configure local CLI environment variables

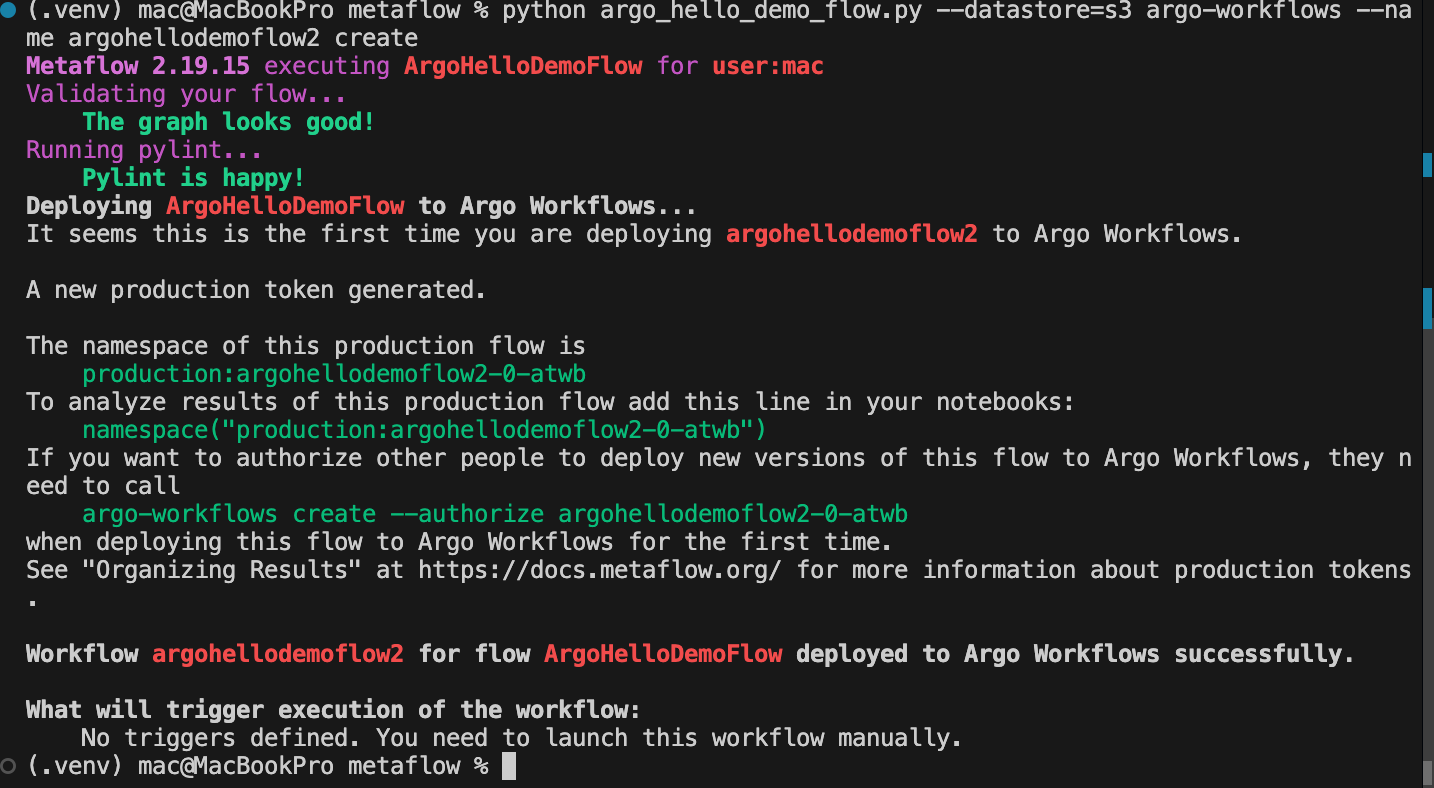
Set environment variables for local CLI (port-forwarded endpoints).

export METAFLOW\_S3\_ENDPOINT\_URL=http://127.0.0.1:19000  
export METAFLOW\_SERVICE\_URL=http://127.0.0.1:18080  
export AWS\_ACCESS\_KEY\_ID=minio  
export AWS\_SECRET\_ACCESS\_KEY=minio123  
export METAFLOW\_DEFAULT\_DATASTORE=s3  
export METAFLOW\_DATASTORE\_SYSROOT\_S3=s3://metaflow  
export METAFLOW\_ARGO\_WORKFLOWS\_NAMESPACE=argo  
export METAFLOW\_KUBERNETES\_NAMESPACE=argo  
export METAFLOW\_ARGO\_WORKFLOWS\_ENV\_VARS\_TO\_SKIP=METAFLOW\_S3\_ENDPOINT\_URL  
export METAFLOW\_SERVICE\_INTERNAL\_URL=http://metaflow-metaflow-service.metaflow.svc.cluster.local:8080

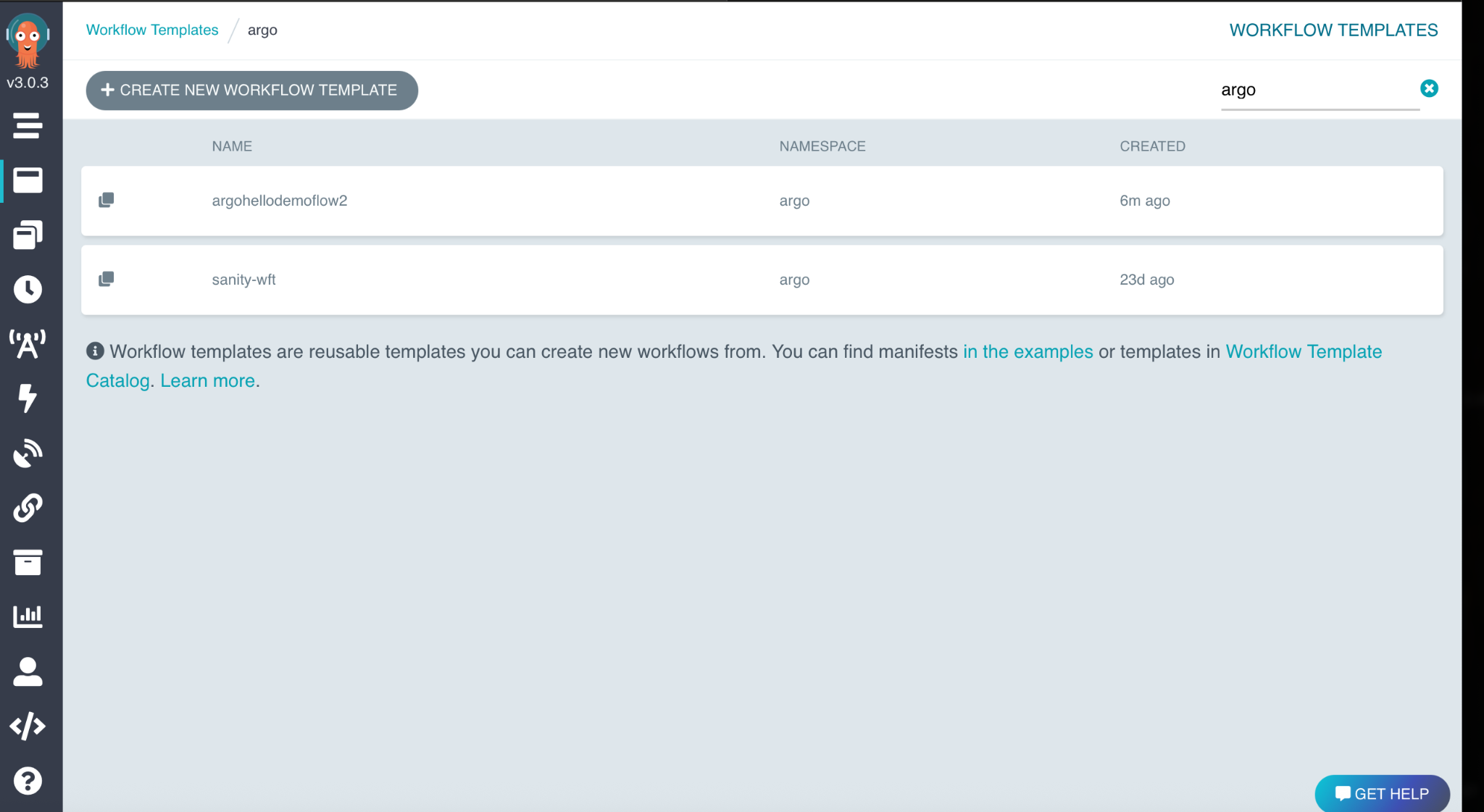
## B6. Deploy the WorkflowTemplate (Metaflow → Argo)

From your repo directory, run:

python argo\_hello\_demo\_flow.py --datastore=s3 argo-workflows --name <workflow-name> create

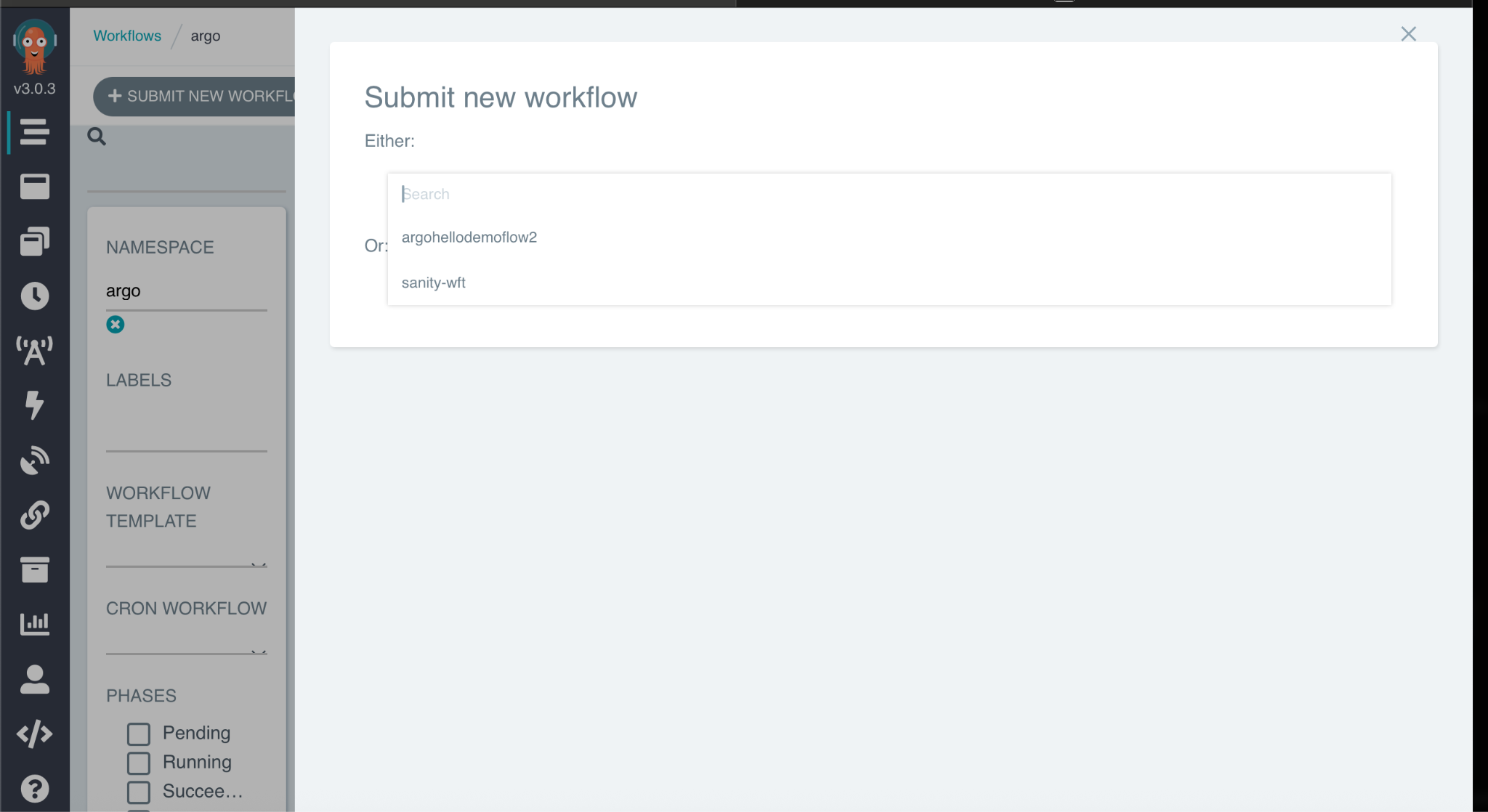


**Screenshot: Argo UI – WorkflowTemplates list**



*Metaflow-generated WorkflowTemplate deployed to Argo.*

The workflow can be triggered either from the Metaflow CLI or by submitting the WorkflowTemplate directly from the Argo UI.



## B7. Trigger the workflow run

Run:

python argo\_hello\_demo\_flow.py --datastore=s3 argo-workflows --name <workflow-name> trigger



**Screenshot: Argo UI – Workflow runs list**



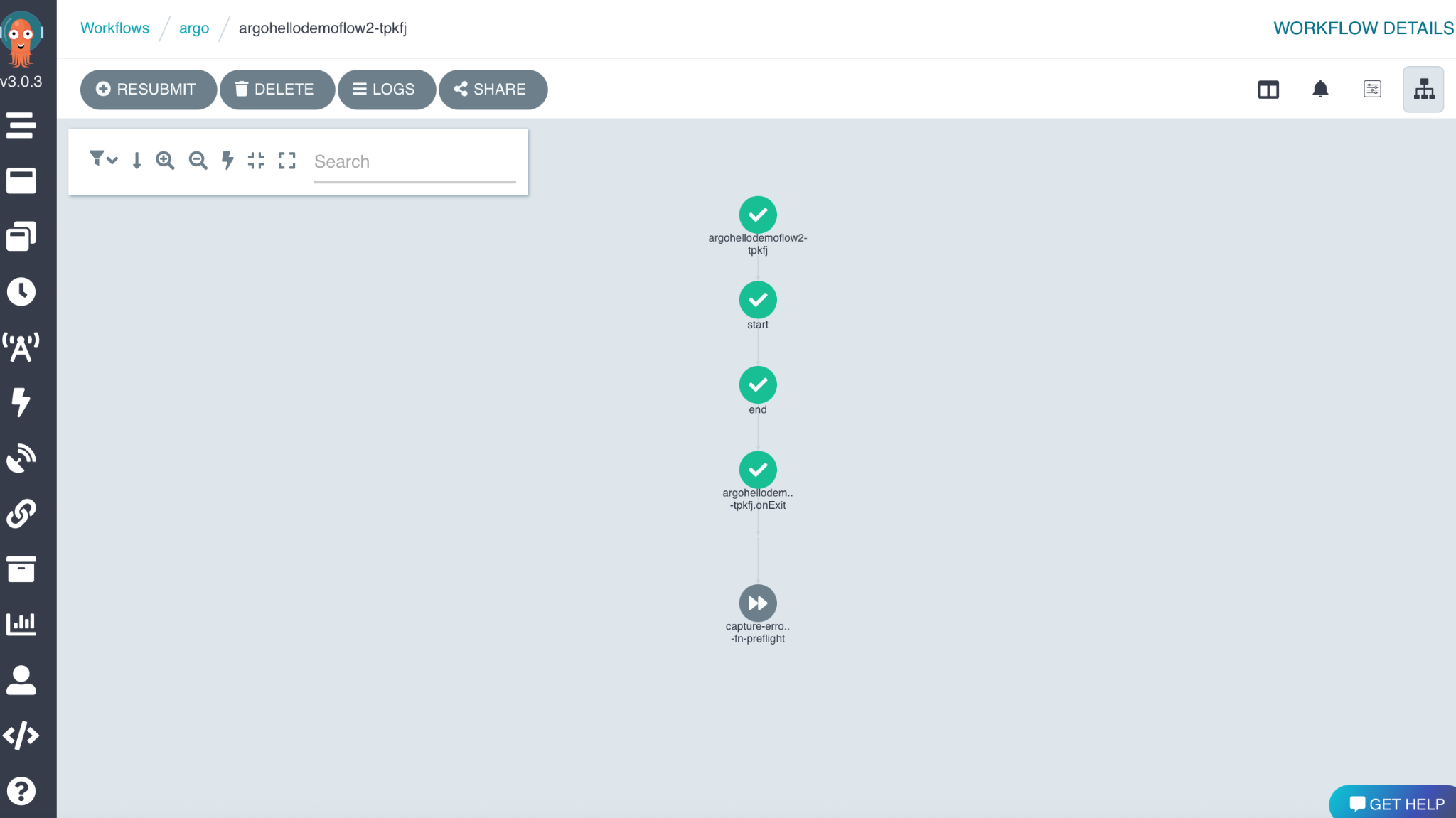
*Workflow run triggered from Metaflow CLI and visible in Argo UI.*

## B8. Validate execution (DAG + logs + kubectl)

In Argo UI, open the workflow run and capture:

* DAG view (step dependencies and status)
* Step logs (click a node → Logs)

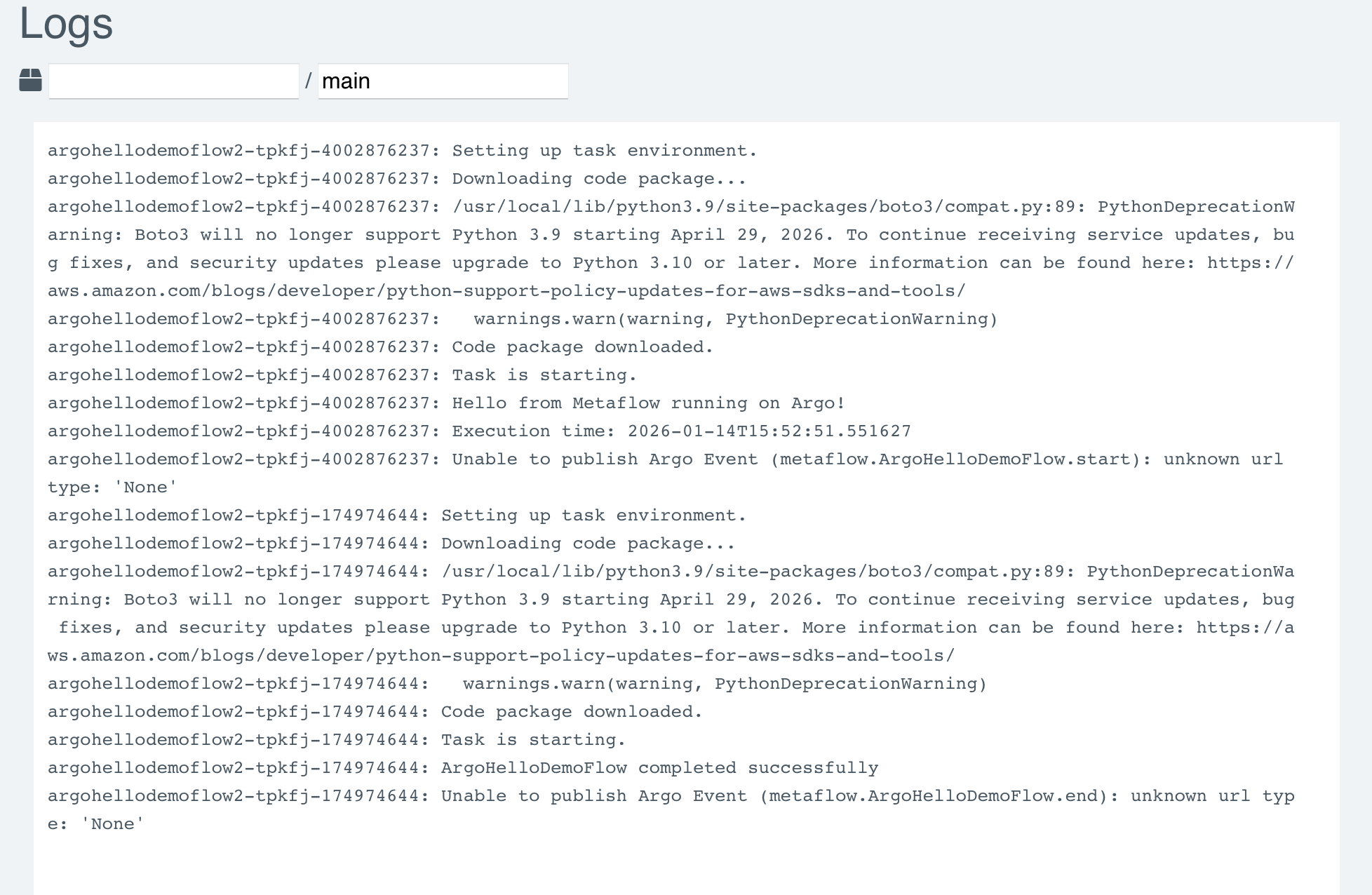
**Screenshot: Argo UI – DAG view**



*Workflow DAG view in Argo UI*

*This view shows the dependency graph of workflow steps and their execution status.*

**Screenshot: Argo UI – Step logs**



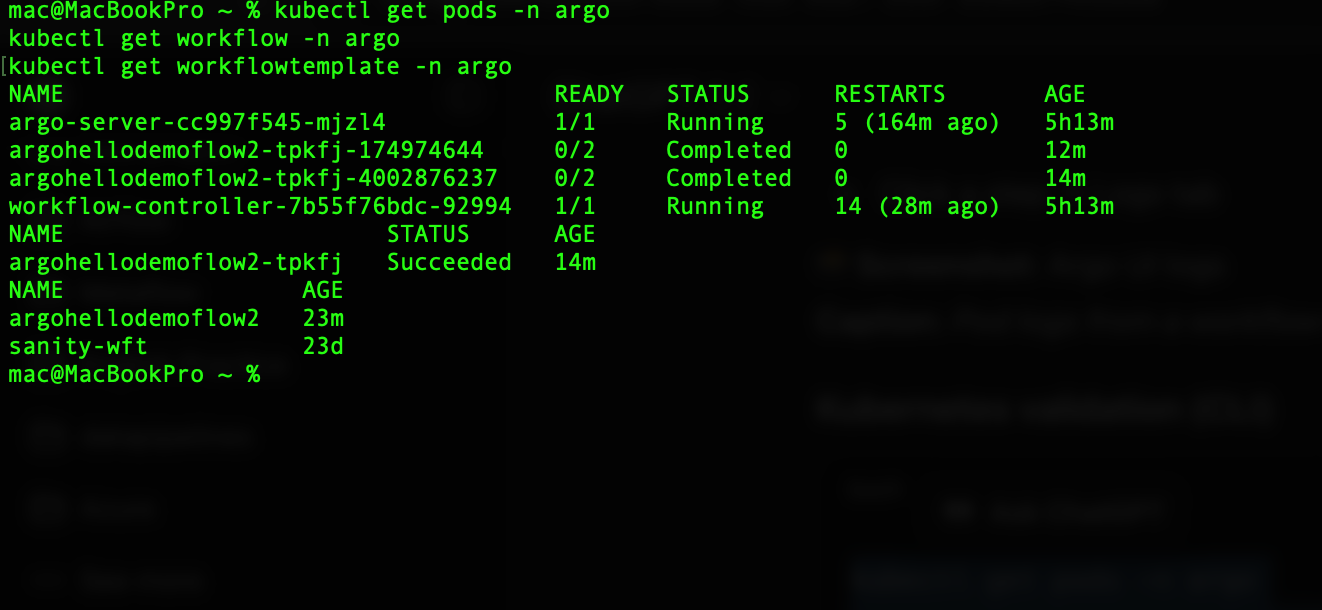
*Logs for an individual workflow step*

*This shows logs generated by a specific workflow step, useful for debugging and monitoring execution.*

CLI validation commands:

kubectl get pods -n argo  
kubectl get workflow -n argo  
kubectl get workflowtemplate -n argo

**Screenshot: Terminal output (kubectl get pods/workflow/workflowtemplate)**

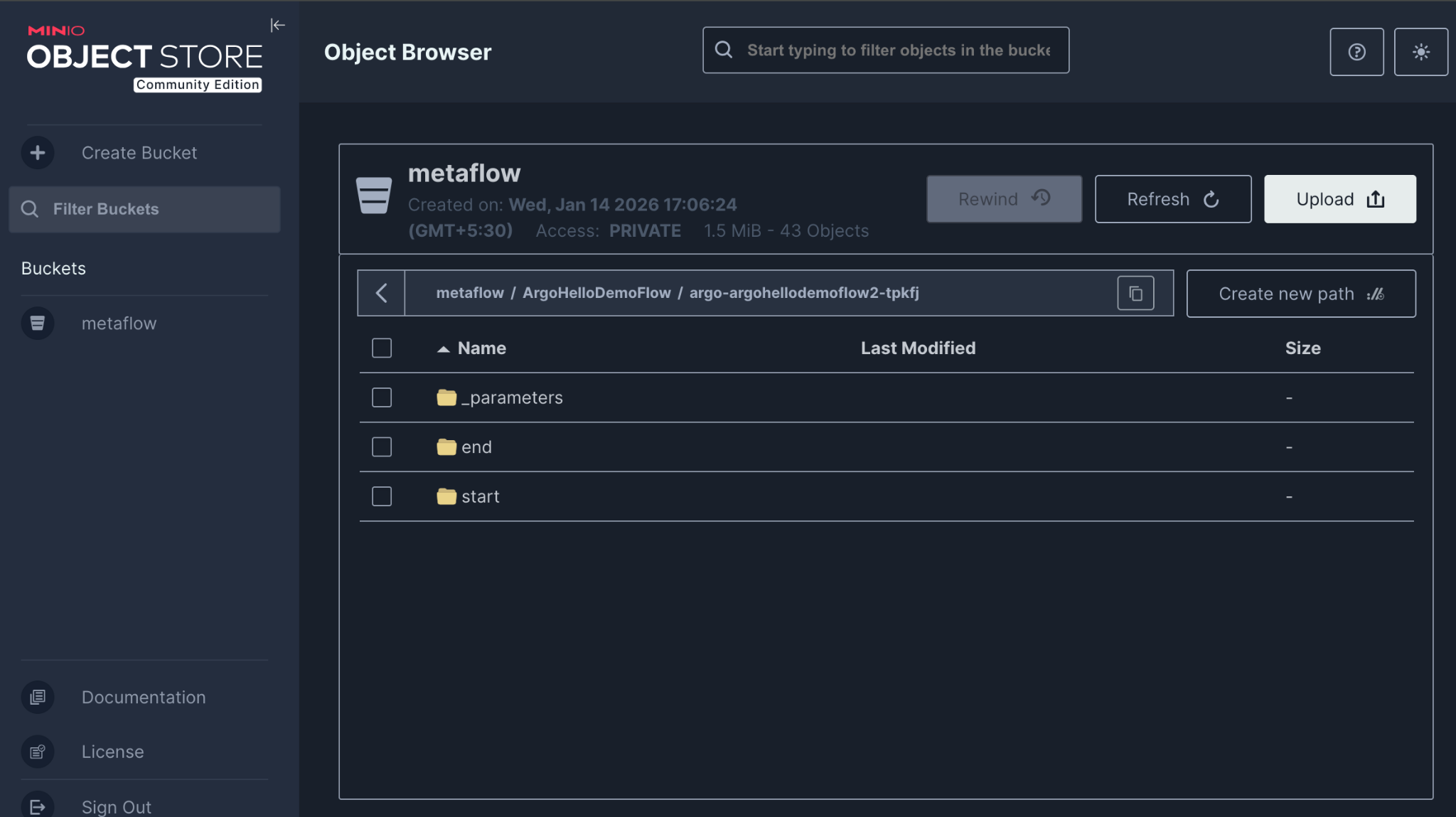


*Kubernetes confirmation of workflow resources and pods.*

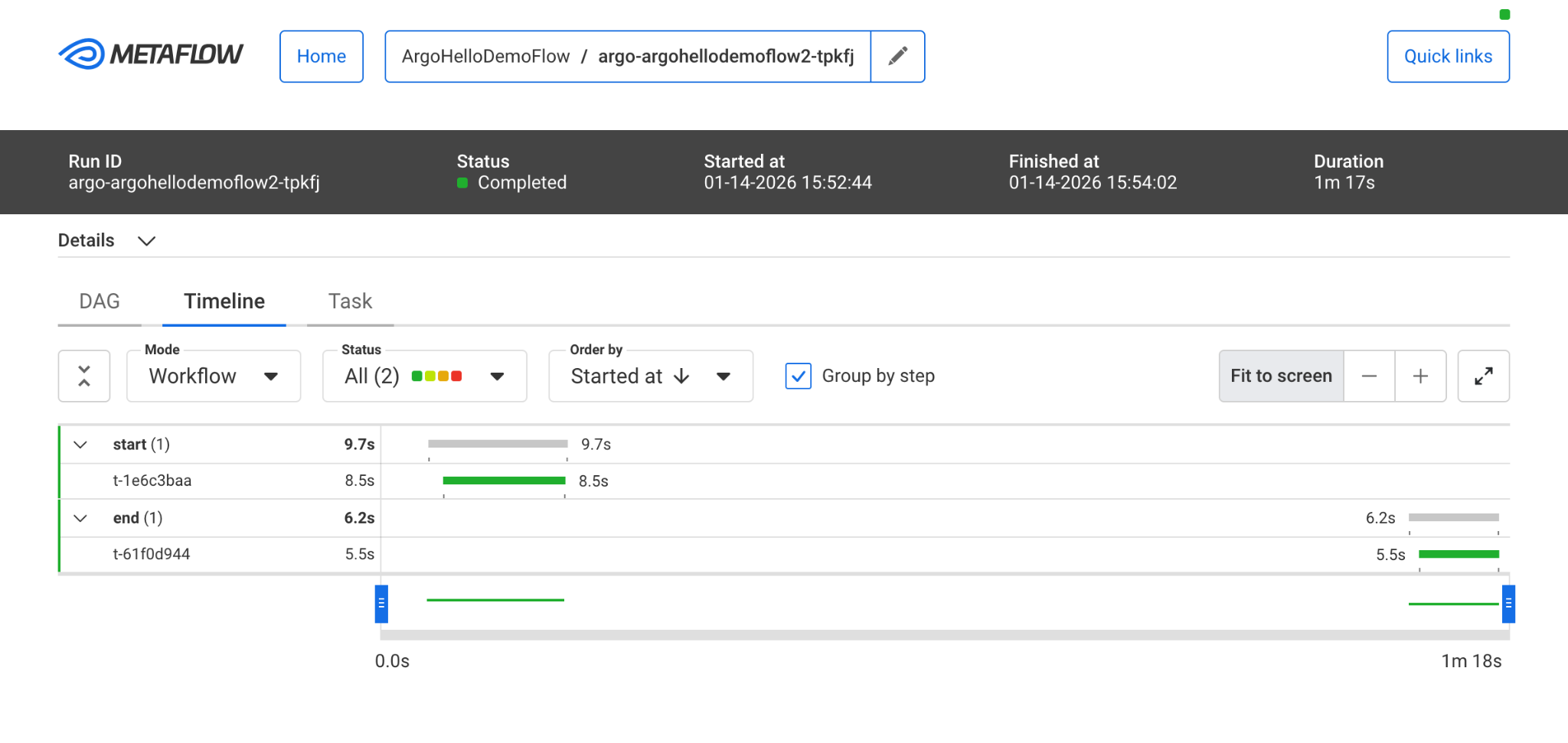
## B9. Validate artifacts in MinIO and metadata in Metaflow UI

MinIO validation: open MinIO UI and confirm objects created under the artifacts bucket.

**Screenshot: MinIO UI – objects created by run**

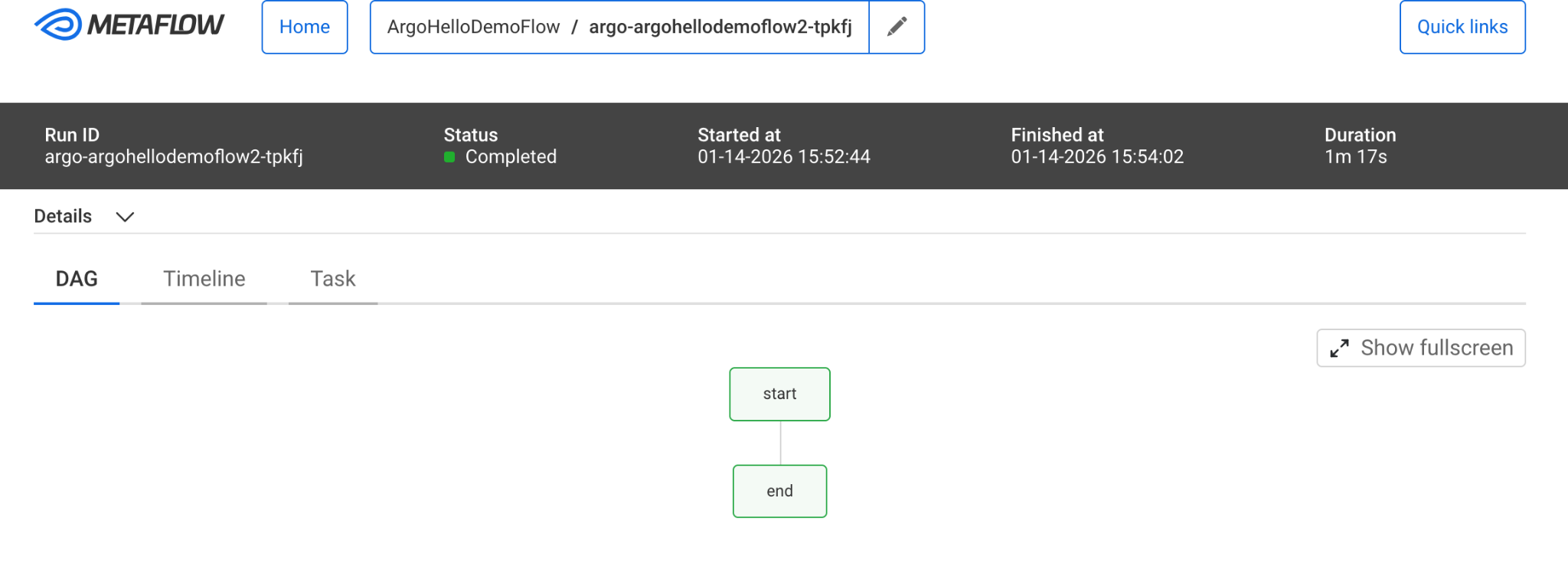


*Artifacts created by workflow steps stored in MinIO.*



Metaflow validation: open Metaflow UI and confirm the run is visible with step details.

**Screenshot: Metaflow UI – run details**



*Metaflow run details view (steps + status).*

*This view shows execution details, step status, and metadata recorded by the Metaflow service.*

## B10. Repo references (files to mention in the document)

Key files in the repository:

* argo\_hello\_demo\_flow.py – example Metaflow pipeline
* secret.yaml – MinIO credentials and endpoint for pods
* configmap.yaml – Metaflow configuration for Argo execution
* metapolis-flow-sa-rbac.yaml – ServiceAccount and namespace RBAC
* metaflow-argo-cluster-rbac.yaml – ClusterRole and ClusterRoleBinding