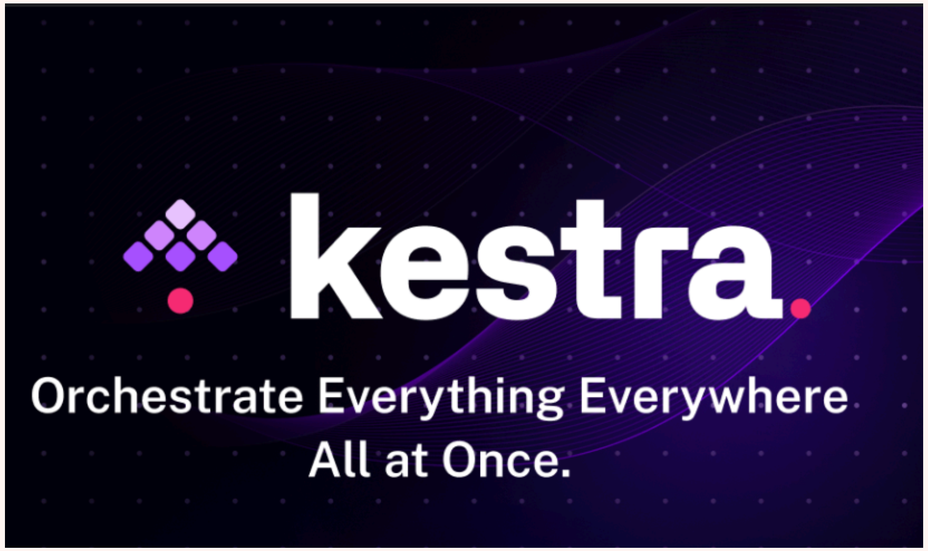
WK 2 DE ZOOMCAMP KESTRA

INTRO

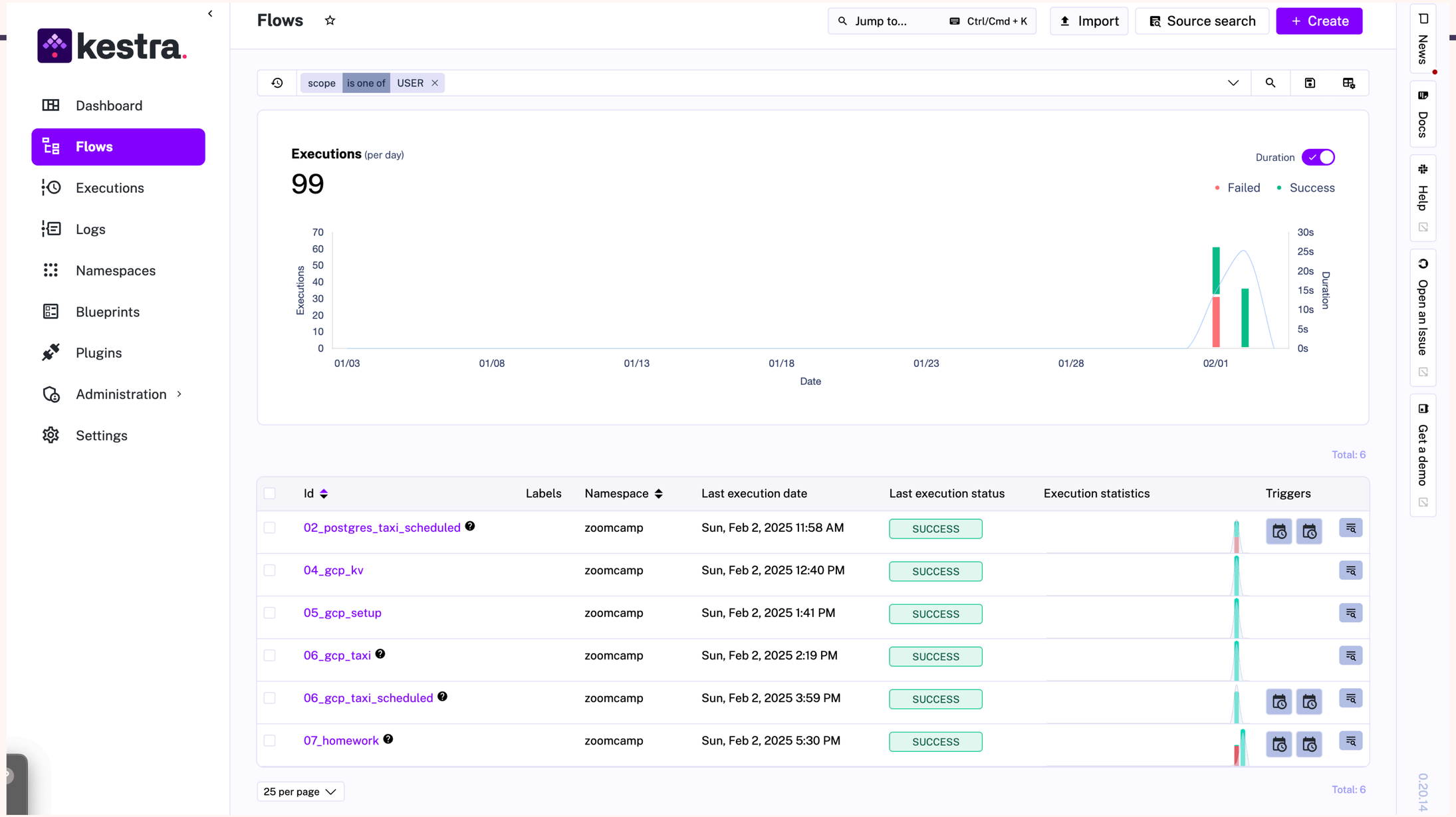
- **Kestra** is an open source data orchestration platform designed to automate complex workflows and pipelines
- It supports scheduling, dependency management, and backfill operations for seamless batch processing
- With native integrations to data sources like PostgreSQL, GCP BigQuery, and Cloud Storage, Kestra streamlines data movement and transformation
- Its extensible plugin architecture empowers users to customize and scale their data workflows effectively



Kestra

RUNNING KESTRA USING DOCKER COMPOSE

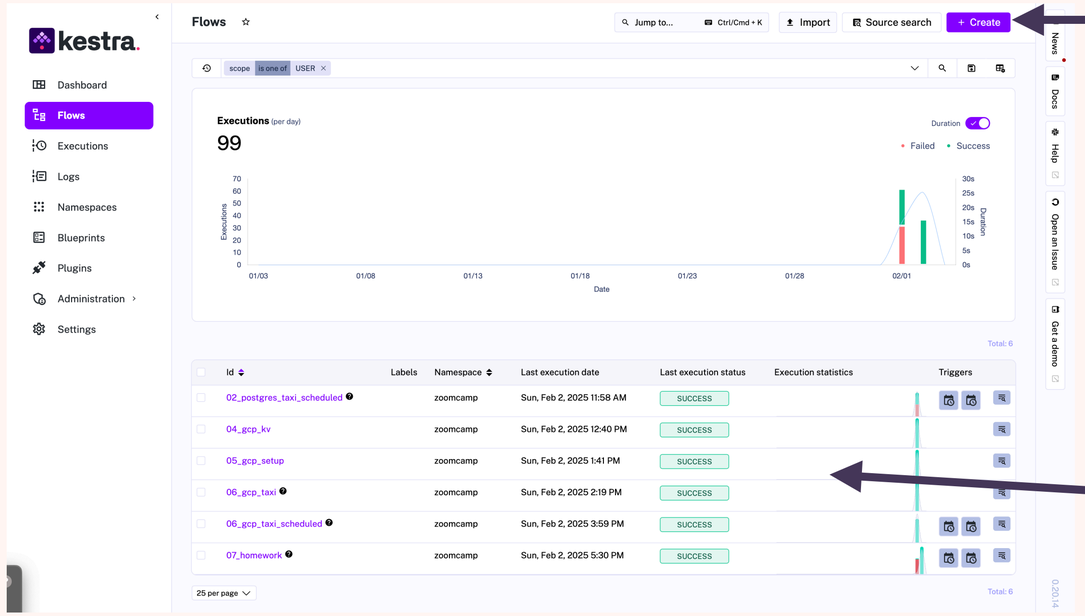
- Kestra can be run or installed in multiple ways (see https://kestra.io/docs/installation)
- You can download the docker-compose file to run Kestra from the following link: https://raw.githubusercontent.com/kestra-io/kestra/develop/docker-compose.yml
- Once the docker-compose file is downloaded, run ‘docker compose up’ from your terminal. If successful, the Kestra page should be accessible on port 8080


kestra.
Dashboard
Flows
Executions
Logs
Namespaces
Blueprints
Plugins
Administration
Settings
Flows
Jump to...
Ctrl/Cmd + K
Import
Source search
+ Create
scope
is one of
USER
Executions (per day)
99
Duration
Failed
Success
Executions
Duration
Date
01/03
01/08
01/13
01/18
01/23
01/28
02/01
Total: 6
Id
Labels
Namespace
Last execution date
Last execution status
Execution statistics
Triggers
02_postgres_taxi_scheduled
zoomcamp
Sun, Feb 2, 2025 11:58 AM
SUCCESS
04_gcp_kv
zoomcamp
Sun, Feb 2, 2025 12:40 PM
SUCCESS
05_gcp_setup
zoomcamp
Sun, Feb 2, 2025 1:41 PM
SUCCESS
06_gcp_taxi
zoomcamp
Sun, Feb 2, 2025 2:19 PM
SUCCESS
06_gcp_taxi_scheduled
zoomcamp
Sun, Feb 2, 2025 3:59 PM
SUCCESS
07_homework
zoomcamp
Sun, Feb 2, 2025 5:30 PM
SUCCESS
25 per page
Total: 6
News
Docs
Help
Open an Issue
Get a demo
0:20:14

Kestra UI

FLOWS IN KESTRA UI

- Flows in Kestra orchestrate the sequence of tasks within a data workflow, defining clear steps for execution



Create a flow

Created flows

Flows

DATA INGESTION INTO POSTGRES USING KESTRA

DATA INGESTION INTO POSTGRES USING KESTRA

- Data Ingestion with Kestra: Ingest NYC Taxi data into the Postgres database using Kestra
- Container Deployment: Run the Postgres container using Docker or Docker Compose
- Docker Compose Configuration: Since Kestra's Docker Compose already includes a Postgres instance for metadata, using a separate Compose file for external Postgres with PgAdmin can cause conflicts. I modified the Kestra Compose file to expose its Postgres on port 5433 and the external Postgres on port 5432, and created a network called "pg-network2" to connect Kestra, its metadata Postgres, and the external Postgres. (Note: I attempted to use "docker.host.internal" to avoid creating a new network, but it did not work in Codespaces.)

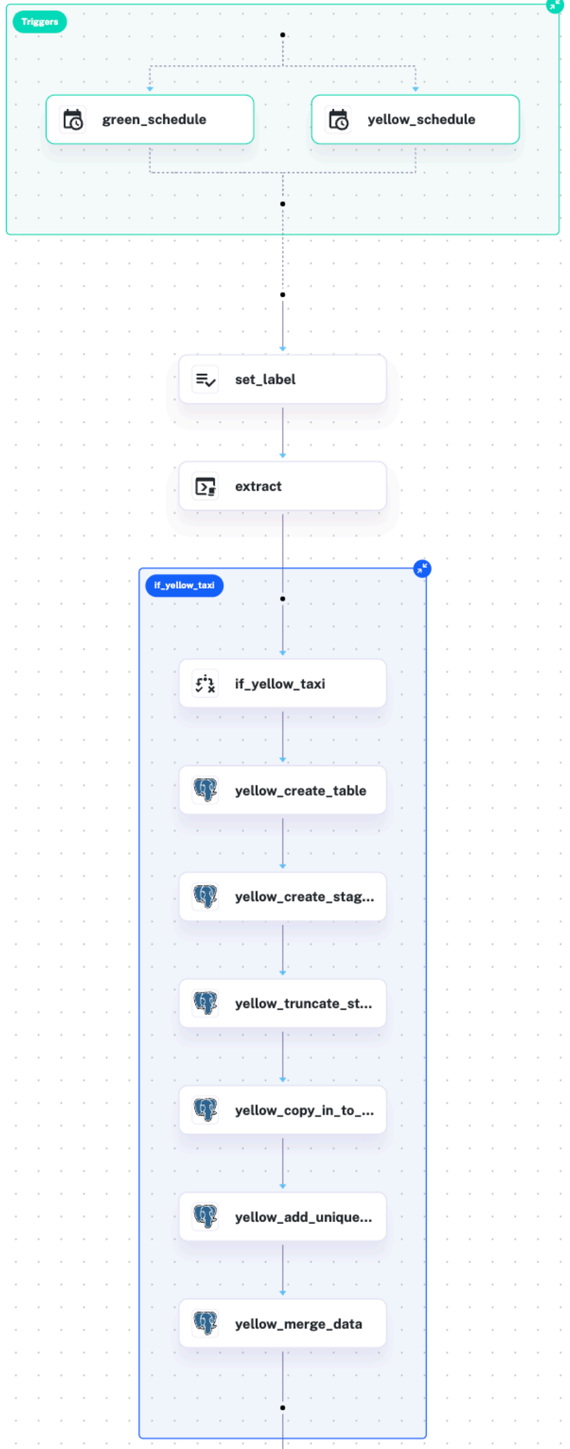
```yaml
networks:
  pg-network2:
    external: true

services:
  postgres:
    image: postgres
    container_name: postgres-db
    environment:
      - POSTGRES_USER=kestra
      - POSTGRES_PASSWORD=k3str4
      - POSTGRES_DB=postgres-zoomcamp
    volumes:
      - postgres-data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U kestra -d postgres-zoomcamp"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - pg-network2

  pgadmin:
    image: dpage/pgadmin4
    container_name: pg_admin
    environment:
      - PGADMIN_DEFAULT_EMAIL=admin@admin.com
      - PGADMIN_DEFAULT_PASSWORD=root
    ports:
      - "8090:80"
    depends_on:
      postgres:
        condition: service_healthy
    restart: unless-stopped
    networks:
      - pg-network2

volumes:
  postgres-data:
```

External postgres and Pgadmin

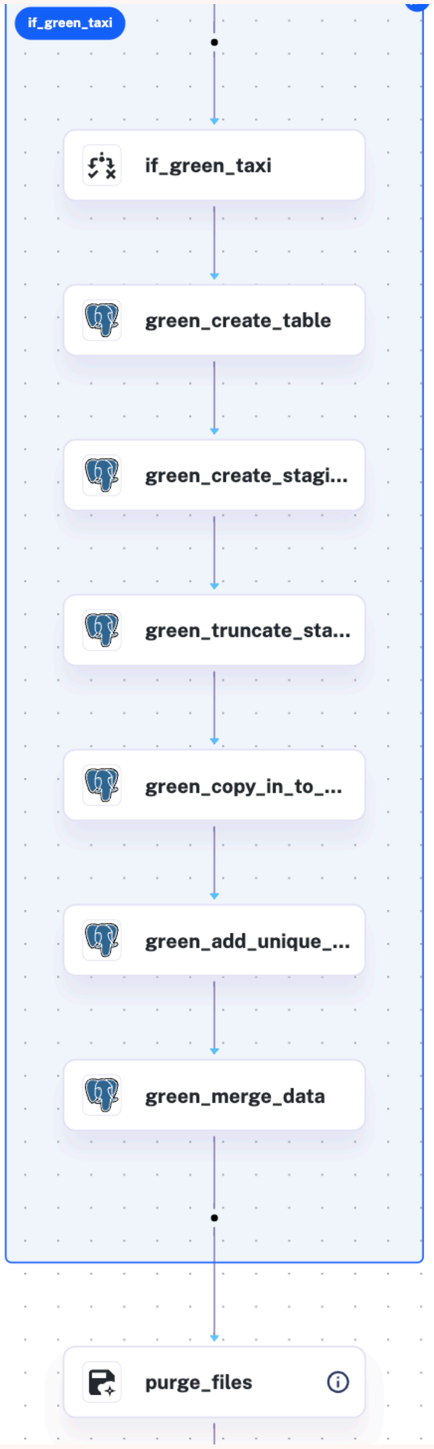
```yaml
volumes:
  postgres-data:
    driver: local
  kestra-data:
    driver: local

networks:
  pg-network2:
    external: true

services:
  postgres:
    image: postgres
    volumes:
      - postgres-data:/var/lib/postgresql/data
    ports:
      - "5433:5432"  # Running on a different port to avoid conflicts
    environment:
      POSTGRES_DB: kestra
      POSTGRES_USER: kestra
      POSTGRES_PASSWORD: k3str4
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -d ${POSTGRES_DB} -U ${POSTGRES_USER}"]
      interval: 30s
      timeout: 10s
      retries: 10
    networks:
      - pg-network2
```

My modified Kestra's docker compose

With this configuration, we can connect to the external Postgres from Kestra using `jdbc:postgresql://postgres-db:5432/postgres-zoomcamp`


Triggers
green_schedule
yellow_schedule
set_label
extract
if_yellow_taxi
if_yellow_taxi
yellow_create_table
yellow_create_stag...
yellow_truncate_st...
yellow_copy_in_to_...
yellow_add_unique...
yellow_merge_data

+


if_green_taxi
if_green_taxi
green_create_table
green_create_stagi...
green_truncate_sta...
green_copy_in_to_...
green_add_unique_...
green_merge_data
purge_files

WORKFLOW IN SLIDE 7

- We set a trigger for green_taxi and yellow_taxi, the yellow_taxi is to run at 9:00AM on the first day of every month,and the green_taxi scheduler is to run at 10:00AM on the first day of every month
- We created 5 tasks:
  - set_label
  - Extract
  - If_yellow_taxi
  - If_green_taxi
  - Purge

FLOW INPUTS AND VARIABLES

- We have one input, taxi of type select which allows users to select an option between ‘yellow” and “green” in the Kestra UI
- We set 4 variables: file stores the expected filename. Staging_table and table store the expected staging_table name and table name respectively. Data is used to reference the output file from “extract” task.
- Inputs can be accessed using inputs.taxi
- Variables can be accessed using vars.file, vars.table and so on

```
inputs:
  - id: taxi
    type: SELECT
    displayName: Select taxi type
    values: [yellow, green]
    defaults: yellow

variables:
  file: "{{inputs.taxi}}_tripdata_{{trigger.date | date('yyyy-MM')}}.csv"
  staging_table: "public.{{inputs.taxi}}_tripdata_staging"
  table: "public.{{inputs.taxi}}_tripdata"
  data: "{{outputs.extract.outputFiles[inputs.taxi ~ '_tripdata_' ~ (trigger.date
```

Inputs and Variables

TASK 1: SET LABEL

- Assigns labels to the execution for easy tracking in the UI
- Uses render(vars.file) to dynamically insert the selected file name
- Adds a label for the taxi type (yellow or green).

```yaml
tasks:
  - id: set_label
    type: io.kestra.plugin.core.execution.Labels
    labels:
      file: "{{render(vars.file)}}"
      taxi: "{{inputs.taxi}}"
```

set_label task

TASK 2: EXTRACT

- User selects a taxi type, year, and month in the Kestra UI
- The workflow constructs dynamic filenames and table names based on user selections.
- It downloads the selected dataset from GitHub and extracts and decompresses it as a CSV file using shell command, wget, and gunzip

```yaml
id: extract
type: io.kestra.plugin.scripts.shell.Commands
outputFiles:
  - "*.csv"
taskRunner:
  type: io.kestra.plugin.core.runner.Process
commands:
  - wget -qO- https://github.com/DataTalksClub/nyc-
  file)}}.gz | gunzip > {{render(vars.file)}}
```

Extract

TASK 3: IF_YELLOW

- Checks if the input taxi type is ‘yellow'
- Executes tasks related to processing yellow taxi data
- Creates main and staging tables if not exist
- Delete rows in staging_table to ensure that only the current execution data is processed
- Loads data into staging table and updates with unique IDs
- Merges data into the main yellow taxi table

TASK 4: IF_GREEN

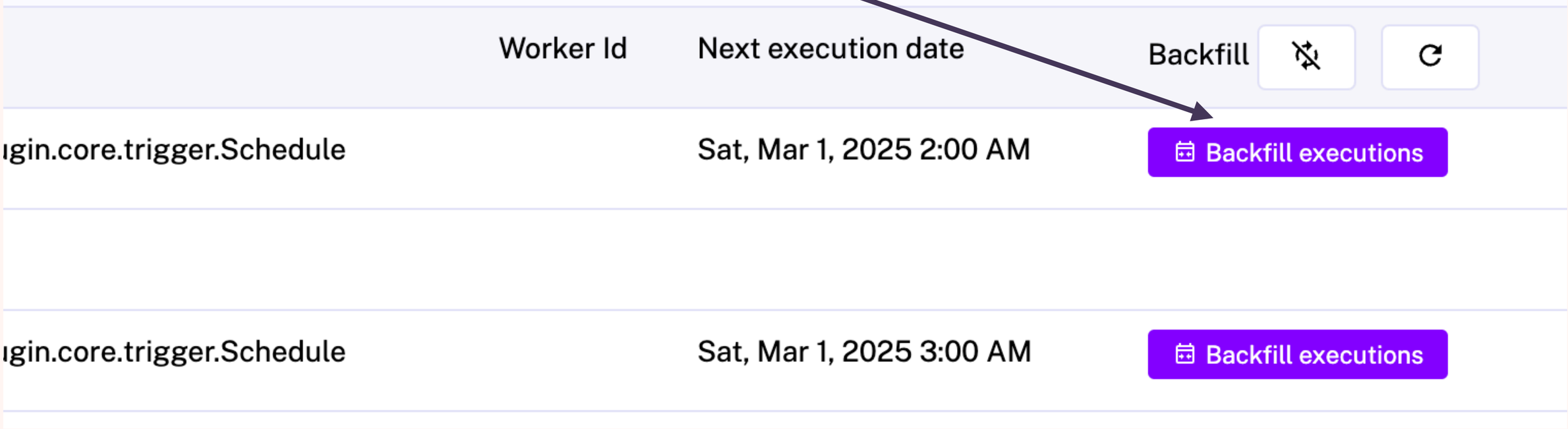
- Checks if the input taxi type is ‘green'
- Executes tasks related to processing green taxi data
- Creates main and staging tables if not exist
- Delete rows in staging_table to ensure that only the current execution data is processed
- Loads data into staging table and updates with unique IDs
- Merges data into the main green taxi table

TASK 5: PURGE_FILES

- This removes downloaded files from the Kestra storage

EXECUTION

- To process data from 2019–2021, use backfill executions in the trigger tab (as it's now 2025)



TODO

- Modify the if_yellow and if_green conditions to process only months with existing data, or use ExecutionStatus/ExecutionOutputs conditions with the trigger scheduler

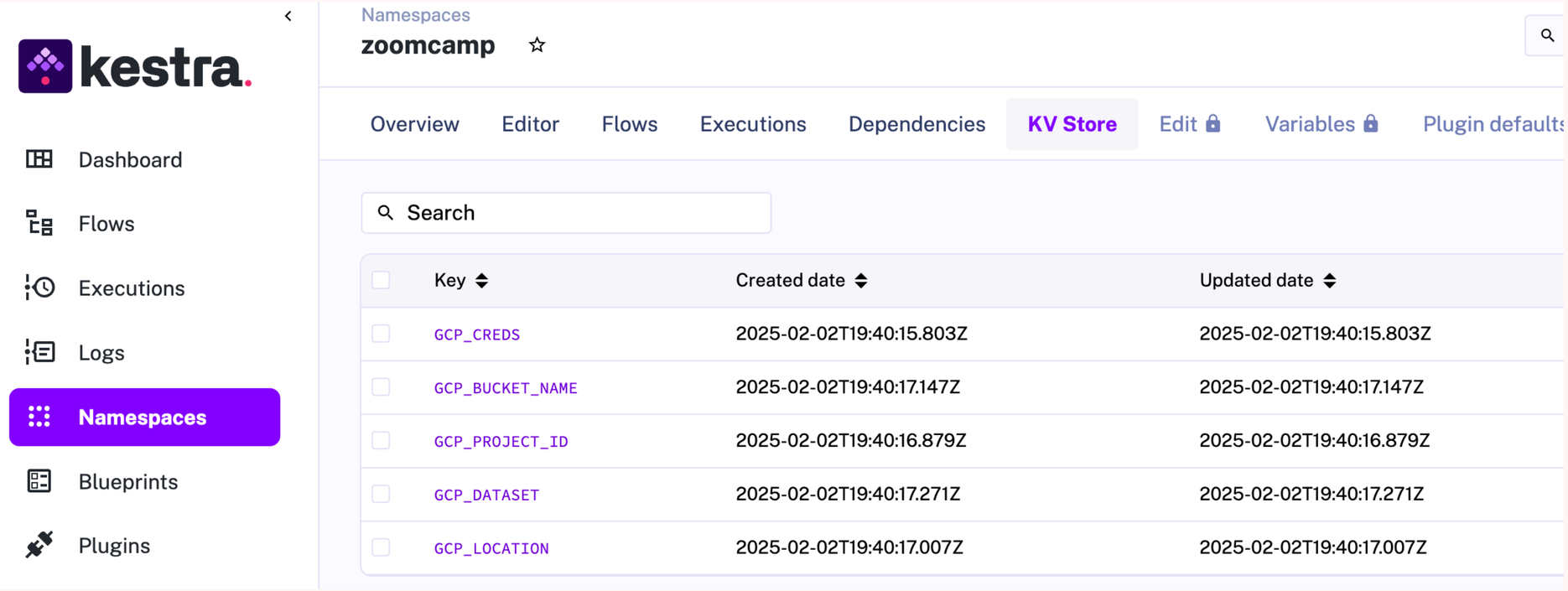
DATA INGESTION INTO GOOGLE CLOUD

WORKFLOW

- The Google Cloud workflow mirrors the Postgres workflow, but each month has a dynamically created staging_table, eliminating the need to delete rows. However, Google Cloud variables must be set

SETTING GOOGLE CLOUD VARIABLES

- The Google Cloud variables can be set manually from the KV Store in the namespace tab



KV Store

SETTING GOOGLE CLOUD VARIABLES

- The Google Cloud variables can also be set by creating a KV Set-based flow

```
- id: gcp_project_id
  type: io.kestra.plugin.core.kv.Set
  key: GCP_PROJECT_ID
  kvType: STRING
  value: de-zoomcamp-449719 #TODO
```