Containerized Scraper Development Guide

IODA Team

July 30, 2025

Contents

1 Modern Architecture Overview

1.1 Container-First Design

All scrapers run in Docker containers orchestrated by Dagu. Each provider has:

- **One container image** with both scraper and processor
- **One DAG file** defining the workflow
- **Volume-mounted data persistence** at `/data`
- **Independent execution** with retry logic and monitoring

1.2 Standardized Structure

```
src/scrapers/{country}/{provider}/
          scrape.py                 # Raw data collection
          post_process.py           # Data processing
          requirements.txt          # Dependencies
          .gitignore                # Ignore patterns
          Dockerfile                # Generated by publish.sh

dagu_config/dags/
          {country}_{provider}.yaml   # Workflow definition

data/{country}/{provider}/
          raw/{year}/{month}/         # Raw scraped data
          processed/{year}/{month}/   # Structured JSON output
```

1.3 Separation of Concerns

Key Change: Scrapers and processors are separate, independent steps:

```python
# scrape.py - ONLY fetches raw data
class BSESRajdhani:
    def scrape(self):
        # Fetch data from website
        # Save raw HTML/JSON/Excel file
        # Exit (no processing)

# post_process.py - ONLY processes data
class BSESRajdhaniProcessor:
    def process(self):
        # Find raw files
        # Parse and structure data
        # Save JSON output
```

Naming Convention

Always name your class after the actual provider, such as `TataPower`, `BSESRajdhani`, or `TNPDCL`. This helps improve code clarity and logging traceability.

2 Scraper Types and Templates

3 Standard Class Structure

```python
class BSESRajdhani:
    def __init__(self):
        self.provider = "bses_rajdhani"
        self.country = "india"
        self.base_path = "/data"
        self.today_iso = datetime.today().strftime("%Y-%m-%d")
        self.year = str(datetime.now().year)
        self.month = str(datetime.now().month).zfill(2)
        self.url = 'https://www.bsesdelhi.com/...'
```

4 Error Handling Patterns

4.1 404 Error Text Files

When a dropdown option is not available, or the page has no data for the selected day, create a `404_YYYY-MM-DD.txt` file in the `raw` folder. Example:

```python
with open(os.path.join(raw_folder, f"404_{self.today_iso}.txt"), "w", encoding=
    "utf-8") as f:
    f.write(f"No dropdown entry for {self.today_indian}: {type(e).__name__} - {
    str(e)}\n")
```

Example content:

```
No dropdown entry for 28-07-2025: NoSuchElementException - Message: Cannot locate option with
```

4.2 No Data Log (Post-Processor)

If the scraper creates a 404 file, the processor should skip parsing and log the absence:

```python
log_path = os.path.join(self.create_folder("processed"), f"no_data_found.{self.
    today_iso}.log")
with open(log_path, "w") as f:
    f.write(f"No outage schedule found for {self.today_iso}. See 404_{self.
    today_iso}.txt in raw folder.")
```

Log file example: `no_data_found.2025-07-28.log`
Contents:

```
No outage schedule found for 2025-07-28. See 404_2025-07-28.txt in raw folder.
```

5 File Output Summary

File Name	Location	Trigger
power_outages.IND.provider.raw.YYYY-MM-DD.html	/raw/YYYY/MM/	Successful scrape
404_YYYY-MM-DD.txt	/raw/YYYY/MM/	Dropdown/date m
no_data_found.YYYY-MM-DD.log	/processed/YYYY/MM/	Processor found nc
power_outages.IND.provider.processed.YYYY-MM-DD.json	/processed/YYYY/MM/	Successful parse

6 Testing Requirements

- When scraping a known missing date:
  - `404_YYYY-MM-DD.txt` is written to `/raw`
  - `no_data_found.YYYY-MM-DD.log` is created in `/processed`
  - Processor exits cleanly with a log message