Config Validator – Technical Documentation

1. Project Overview

Config Validator is a modular and extensible validation framework designed to verify configuration files across multiple environments and storage backends.
It provides asynchronous validation, dynamic rule loading, real-time monitoring, and complete observability through Grafana dashboards.

The project follows modern Python design principles — using **Strategy** and **Decorator** patterns — and is built with scalability, maintainability, and developer efficiency in mind.

---

2. Architecture Overview

Core Components

Component	Description
CLI (cli.py)	Command-line entry point for executing validation workflows and custom commands.
Core	Main validation engine including asynchronous validation, orchestration, configuration loading, reporting, and filesystem watching.
Rules	Modular validation rule set loaded dynamically via a plugin system.
Storage	Implements the **Strategy Pattern** to support multiple storage backends (Local, S3, HDFS, etc.).
Watcher	File-system monitoring service enabling real-time validation and change tracking.

Key Features

- Asynchronous validation with configurable concurrency
- Plugin-based rule loading for easy extensibility
- Real-time file watching and automatic re-validation
- Report generation in both **JSON** and **NDJSON** streams
- Built-in observability via **Grafana, Loki, and Promtail**
- Type-hinted codebase for Python 3.11+
- Developer-friendly automation using **Makefile**

---

3. Project Structure

```
config-validator/
│
├── src/config_validator/
│   ├── cli.py                    # Command-line interface
│   ├── core/                     # Validation engine
│   ├── rules/                    # Modular validation rules
│   ├── storage/                  # Strategy pattern for backends
│   └── utils/                    # Logging and reusable helpers
│
├── tests/                        # Unit and integration tests
├── config/                       # Validation and storage configs
├── ops/observability/            # Monitoring stack (Grafana, Loki, Promtail)
├── reports/                      # Generated reports (NDJSON/JSON)
├── logs/                         # Application logs
├── docs/                         # Project documentation
├── Makefile                      # Build and run automation
└── pyproject.toml                # Dependencies and project settings
```

---

4. Components Overview

1. Easy Rule Extension and Maintenance

Using the **Decorator Pattern**, adding or updating validation rules is simple and flexible. Each rule can be attached to any validation target by adding a decorator above it, ensuring modularity and maintainability.

2. High-Volume Data Handling

Optimized for large-scale data processing.
Since the system is IO-bound, it leverages **async/await** and **Semaphore** for efficient concurrency control.

3. File Monitoring and History Tracking

Powered by **Watchdog**, the system supports **real-time monitoring** and **historical tracking** of configuration file changes.

4. Dynamic Log Reporting

Provides dynamic reporting and filtering capabilities across multiple fields in the logs, enabling flexible analytics.

5. Simplified Execution via Makefile

All project tasks (build, test, run, lint, etc.) can be easily executed via **Makefile**, simplifying developer workflows.

6. Function-Level Logging with Decorators

Custom decorators allow developers to easily enable **automatic logging** on any function. They record the **start and end** of each validation or process step, improving traceability and debugging visibility.

7. Extensible Storage Backends (Strategy Pattern)

Implements the **Strategy Pattern** for seamless integration of different storage types (Local, S3, HDFS, etc.), providing scalability and idempotent behavior.

8. Comprehensive and Extensible Testing Framework

Each module and rule includes dedicated test cases.
The testing system allows **easy writing, extending, and maintaining** of tests, ensuring continuous validation of logic and preventing regressions.

---

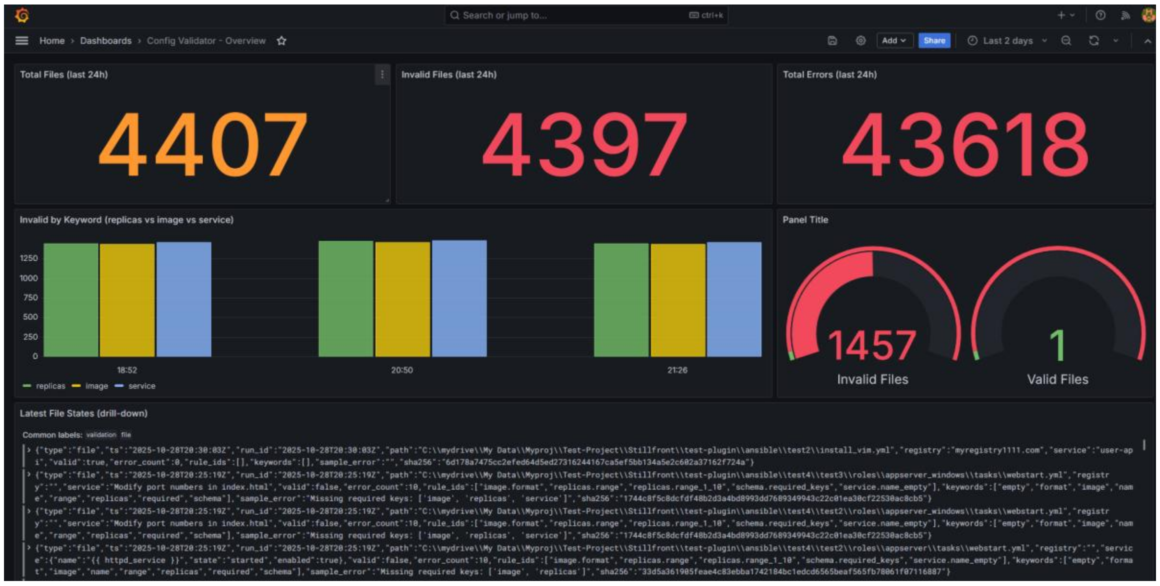
5. Reports and Dashboards

Config Validator – Overview Dashboard

Purpose:
Provides a high-level summary of validation results, errors, and data trends.

Panels:

- **Total Files (last 24h)** – Total number of processed files in the last 24 hours.
- **Invalid Files (last 24h)** – Number of invalid configuration files detected.
- **Total Errors (last 24h)** – Total number of validation issues found.
- **Invalid by Keyword (replicas / image / service)** – Comparison of invalid files by key terms.
- **Gauge – Valid vs Invalid Files** – Visual ratio of valid to invalid files.
- **Latest File States (drill-down)** – Real-time Loki logs for live validation insight.



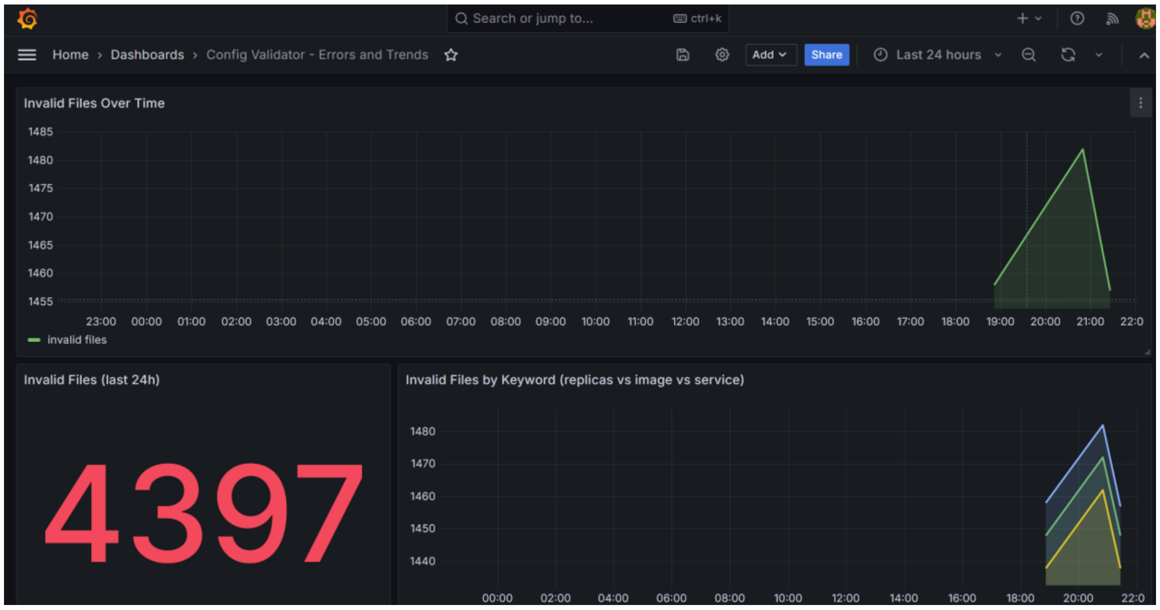
```
> {"type":"file","ts":"2025-10-28T20:30:03Z","run_id":"2025-10-28T20:30:03Z","path":"C:\\mydrive\\My Data\\Myproj\\Test-Project\\Stillfront\\test-plugin\\ansible\\test2\\install_vim.yml","registry":"myregistry1111.com","service":"user-ap
  i","valid":true,"error_count":0,"rule_ids":[],"keywords":[],"sample_error":"","sha256":"6d178a7475cc2efed64d5ed27316244167ca5ef5bb134a5e2c602a37162f724a"}
> {"type":"file","ts":"2025-10-28T20:25:19Z","run_id":"2025-10-28T20:25:19Z","path":"C:\\mydrive\\My Data\\Myproj\\Test-Project\\Stillfront\\test-plugin\\ansible\\test4\\test3\\roles\\appserver_windows\\tasks\\webstart.yml","registr
  y":"","service":"Modify port numbers in index.html","valid":false,"error_count":10,"rule_ids":["image.format","replicas.range","replicas.range_1_10","schema.required_keys","service.name_empty"],"keywords":["empty","format","image","nam
  e","range","replicas","required","schema"],"sample_error":"Missing required keys: ['image', 'replicas', 'service']","sha256":"1744c8f5c8dcfdf48b2d3a4bd8993dd7689349943c22c01ea30cf22530ac8cb5"}
> {"type":"file","ts":"2025-10-28T20:25:19Z","run_id":"2025-10-28T20:25:19Z","path":"C:\\mydrive\\My Data\\Myproj\\Test-Project\\Stillfront\\test-plugin\\ansible\\test4\\test2\\roles\\appserver_windows\\tasks\\webstart.yml","registr
  y":"","service":"Modify port numbers in index.html","valid":false,"error_count":10,"rule_ids":["image.format","replicas.range","replicas.range_1_10","schema.required_keys","service.name_empty"],"keywords":["empty","format","image","nam
  e","range","replicas","required","schema"],"sample_error":"Missing required keys: ['image', 'replicas', 'service']","sha256":"1744c8f5c8dcfdf48b2d3a4bd8993dd7689349943c22c01ea30cf22530ac8cb5"}
> {"type":"file","ts":"2025-10-28T20:25:19Z","run_id":"2025-10-28T20:25:19Z","path":"C:\\mydrive\\My Data\\Myproj\\Test-Project\\Stillfront\\test-plugin\\ansible\\test4\\test2\\roles\\appserver\\tasks\\webstart.yml","registry":"","servic
  e":{"name":"{{ httpd_service }}","state":"started","enabled":true},"valid":false,"error_count":10,"rule_ids":["image.format","replicas.range","replicas.range_1_10","schema.required_keys","service.name_empty"],"keywords":["empty","forma
  t","image","name","range","replicas","required","schema"],"sample_error":"Missing required keys: ['image', 'replicas']","sha256":"33d5a361905feae4c83ebba1742184bc1edcd6565beaf565fb78061f07116887"}
```

Config Validator – Errors and Trends Dashboard

Purpose:
Focuses on **error patterns and validation trends** over time.

Panels:

- **Invalid Files Over Time** – Time-series of invalid files to visualize error trends.
- **Invalid Files (last 24h)** – Current count of invalid files for quick monitoring.
- **Invalid Files by Keyword (replicas / image / service)** – Comparative trend showing which validation category fails most often.



---

6. Observability Stack

Located in ops/observability/, the monitoring stack includes:

- **Loki** – Log aggregation system.
- **Promtail** – Log collector that ships validation logs to Loki.
- **Grafana** – Visualization and dashboarding tool.
- **docker-compose.yml** – Spins up the full observability environment for local or staging use.

---

7. Development and Maintenance

- **Run Locally:**
- make run
- **Execute Tests:**
- make test
- **Start Observability Stack:**
- cd ops/observability && docker compose up -d

---

8. Summary

The **Config Validator** project offers a scalable, testable, and fully observable solution for validating configuration files across distributed environments.
Its modular design, asynchronous engine, and strong observability layer make it suitable for integration into complex DevOps ecosystems.