NETWORK AUTOMATION FORUM

WS-A3 - Network App Development: Design, Deploy, & Automate

Autocon3 – Deck 3

Mau – Deepinder - Emre

AUTOCON 3

THE NETWORK AUTOMATION CONFERENCE

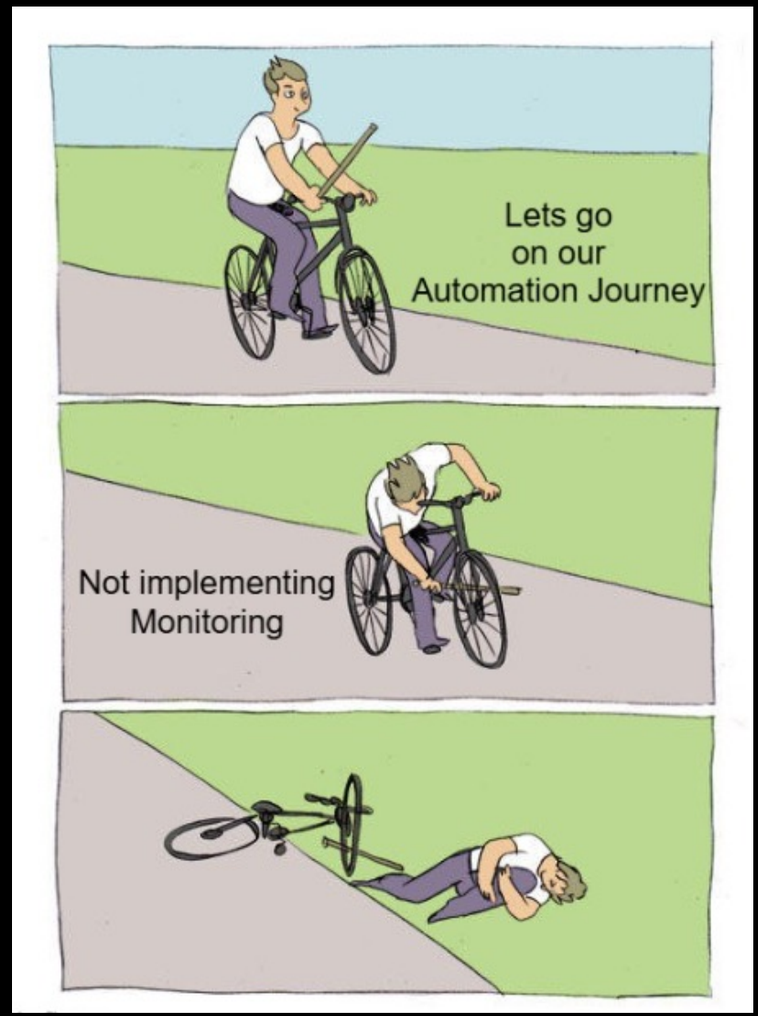
Introduction & Foundations

- **Technology Overview:** Monitoring for Network Automation (Deep)
- **Designing Apps for Customization:** Structuring configurations to avoid modifying core code (Mau)
- **GitHub Best Practices:** Managing branches, pull requests, merging, versioning, CI/CD with GitHub Actions. (Mau)


NAF
NETWORK AUTOMATION FORUM
Technology Overview

The Why?

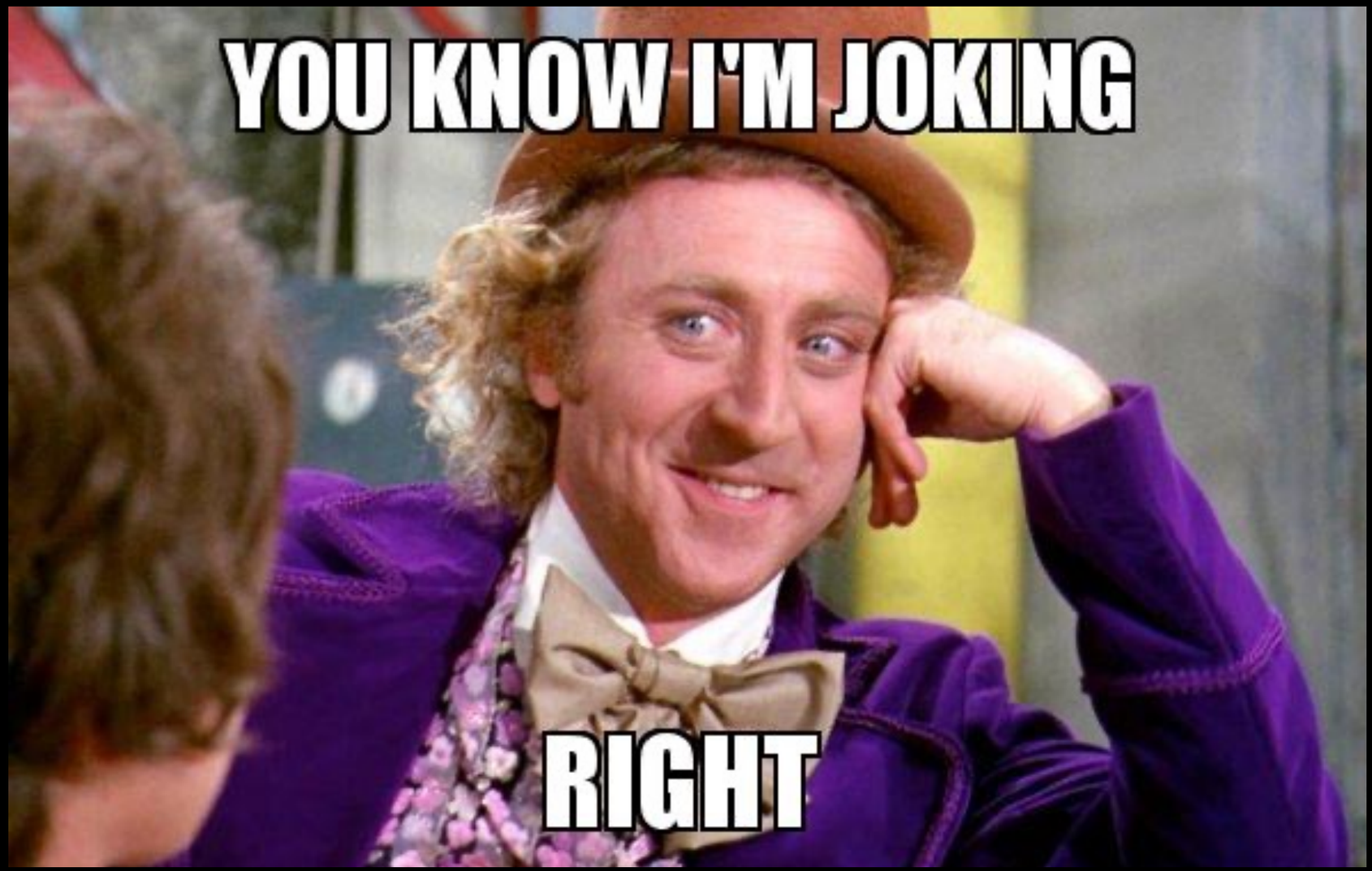
- Automation without monitoring is like flying blind
- Monitoring tells us
  - What is working?
  - What is broken?
  - Why?
- This will help to build with confidence



Introducing

S N M P

Simple Network Management Protocol


YOU KNOW I'M JOKING
RIGHT

NETWORK AUTOMATION FORUM

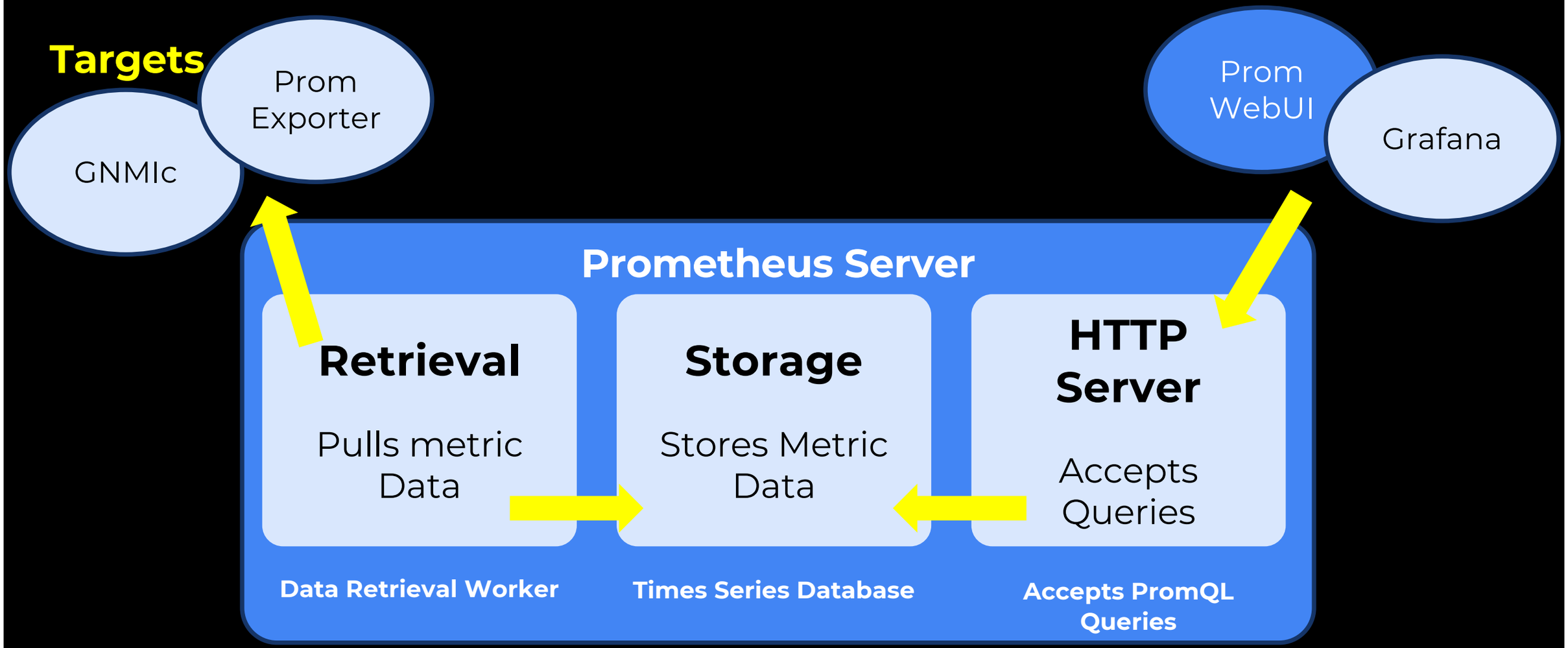
Technology Overview

Prometheus, Grafana and Kafka

What is Prometheus?

- Open-Source monitoring and alerting tool - Easy to Deploy
- Collects metrics by scraping HTTP targets – Powerful Querying
- Stores data in time-series database
- Designed for reliability – Pull based collection
- Part of Cloud Native Computation Foundation (same folks as Kubernetes) – Cloud Native Friendly

Prometheus Main Components



Targets and Metrics

What does Prometheus Monitor?

Linux/Windows Server

Apache Server

Single Application

Service, e.g. Database

Which units are monitored on those targets?

CPU Status

Memory/Disk Usage

Requests Count

Exceptions Count

Request Duration

Prometheus scrapes from http:/host/metrics

Prometheus Alertmanager – Overview

🔔 What is Alertmanager?

- A component of the Prometheus monitoring stack
- Handles alerts sent by Prometheus servers
- Manages silencing, deduplication, grouping, and routing

Example Flow:

Prometheus ➡️ Alert triggered ➡️ Sent to Alertmanager

What is Grafana

grafana

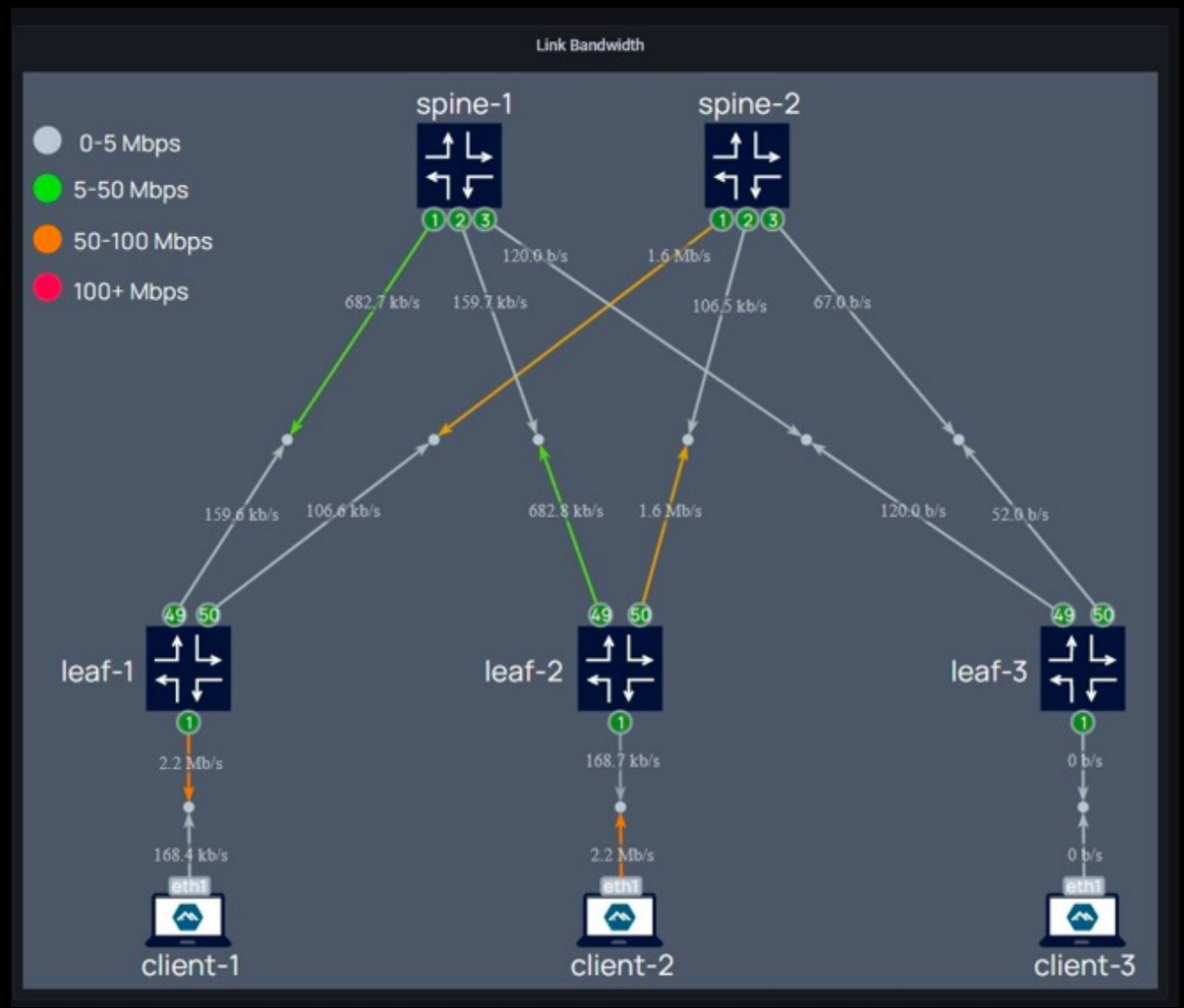
Watch 1.3k | Fork 11.6k | Star 60.5k

- Open-source analytics and monitoring dashboards
- Visualized data from Prometheus, Graphite and many more
- Supports complex dashboards and data transformation

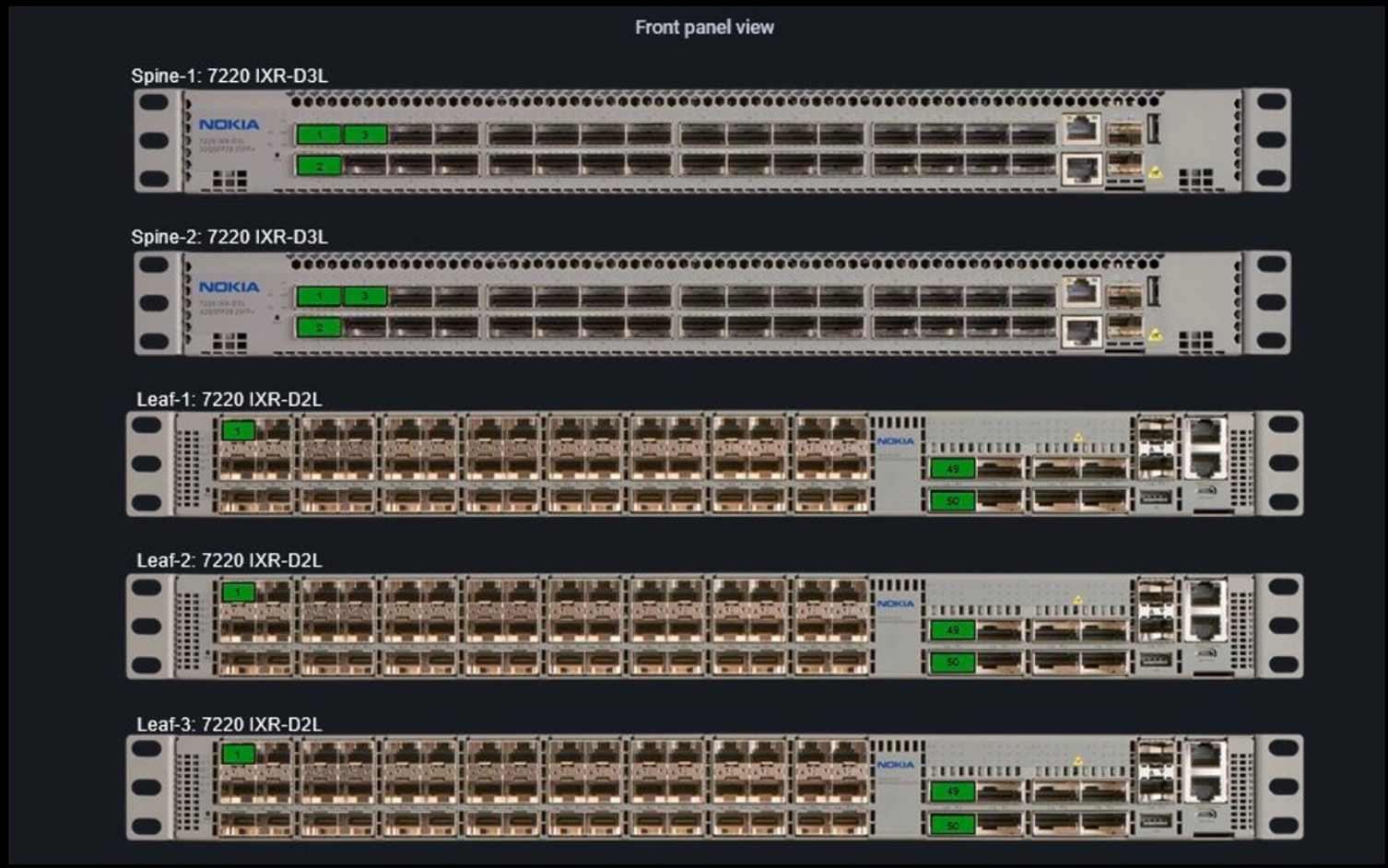
📊 Why Grafana (vs Other Visualization Tools)

- ✅ **Multi-source support** — works with Prometheus, Loki, InfluxDB, Elasticsearch, and more
- ✅ **Real-time dashboards** — dynamic, interactive visualizations
- ✅ **Alerting built-in** — flexible, rule-based alert system
- ✅ **Open source & extensible** — rich plugin ecosystem
- ❌ Other tools — often tied to one data source or commercial licenses

Why do we love Grafana?



Why do we love Grafana?



What is Kafka

- Instead of things, think of Events.
- Distributed event streaming platform
- Used for building real-time data pipelines and streaming applications
- Publishes, subscribes to stores and processes streams of records in real-time

What is an Event

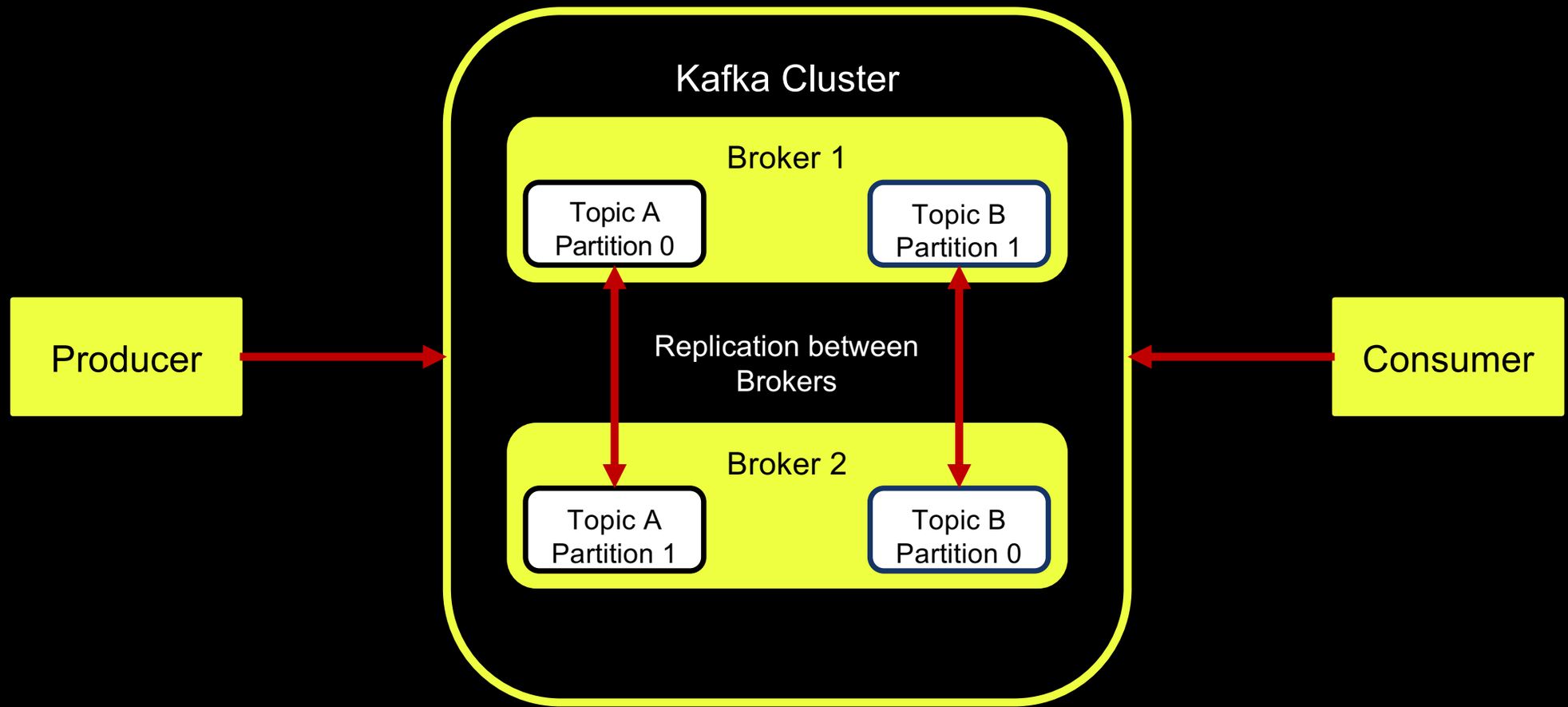
An **event** in Kafka is a data structure that represents **something that happened** at a point in time.

Each event typically contains:

- **Key (optional):** Used for partitioning or identifying the event.
- **Value:** The actual data payload of the event.
- **Timestamp:** The time the event occurred or was logged.
- **Metadata:** Includes information like topic name, partition number, and offset.

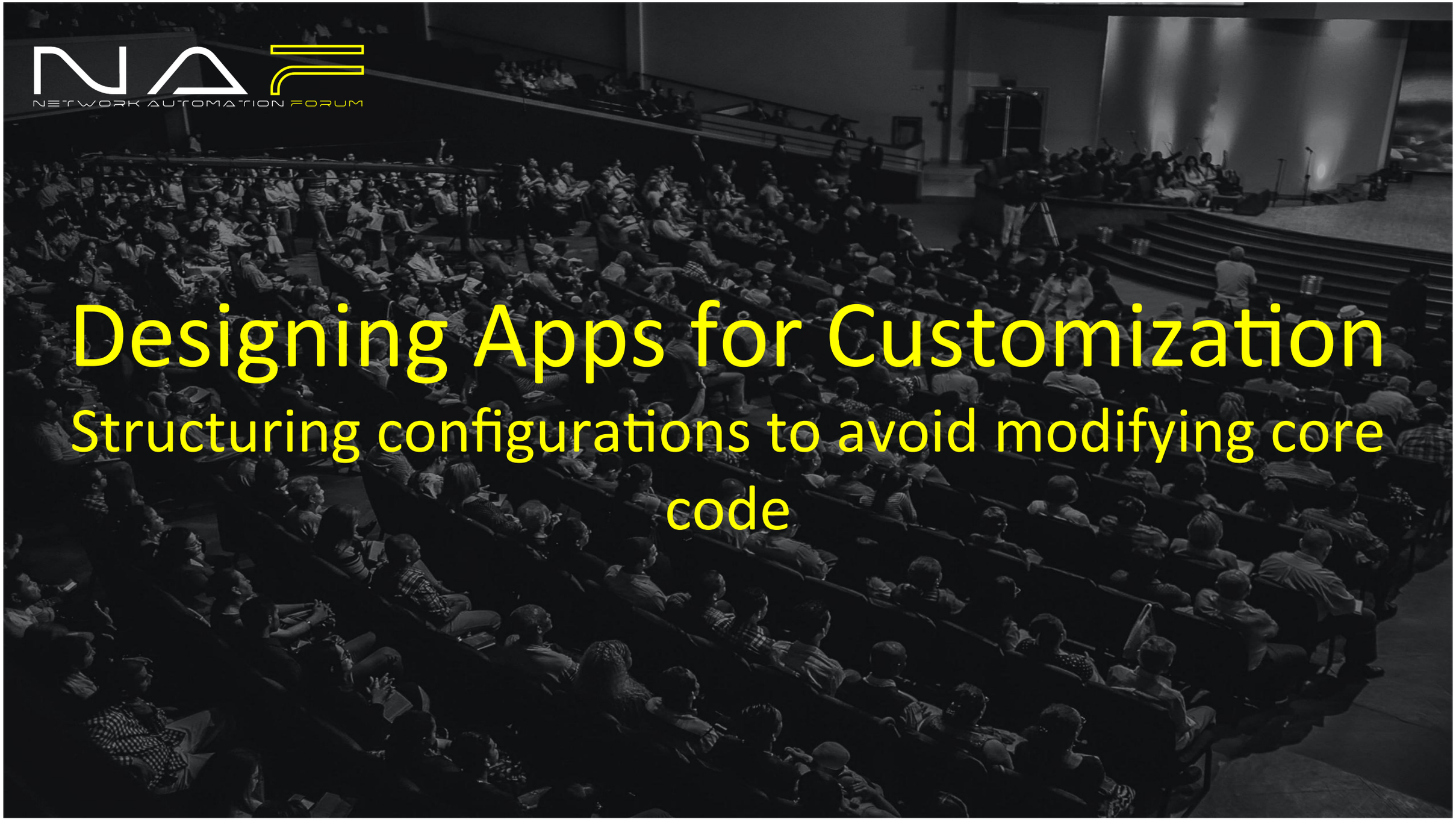
```
{
  "key": "user123",
  "value": {
    "action": "login",
    "timestamp": "2025-05-24T10:45:00Z"
  }
}
```

Main Components of Kafka



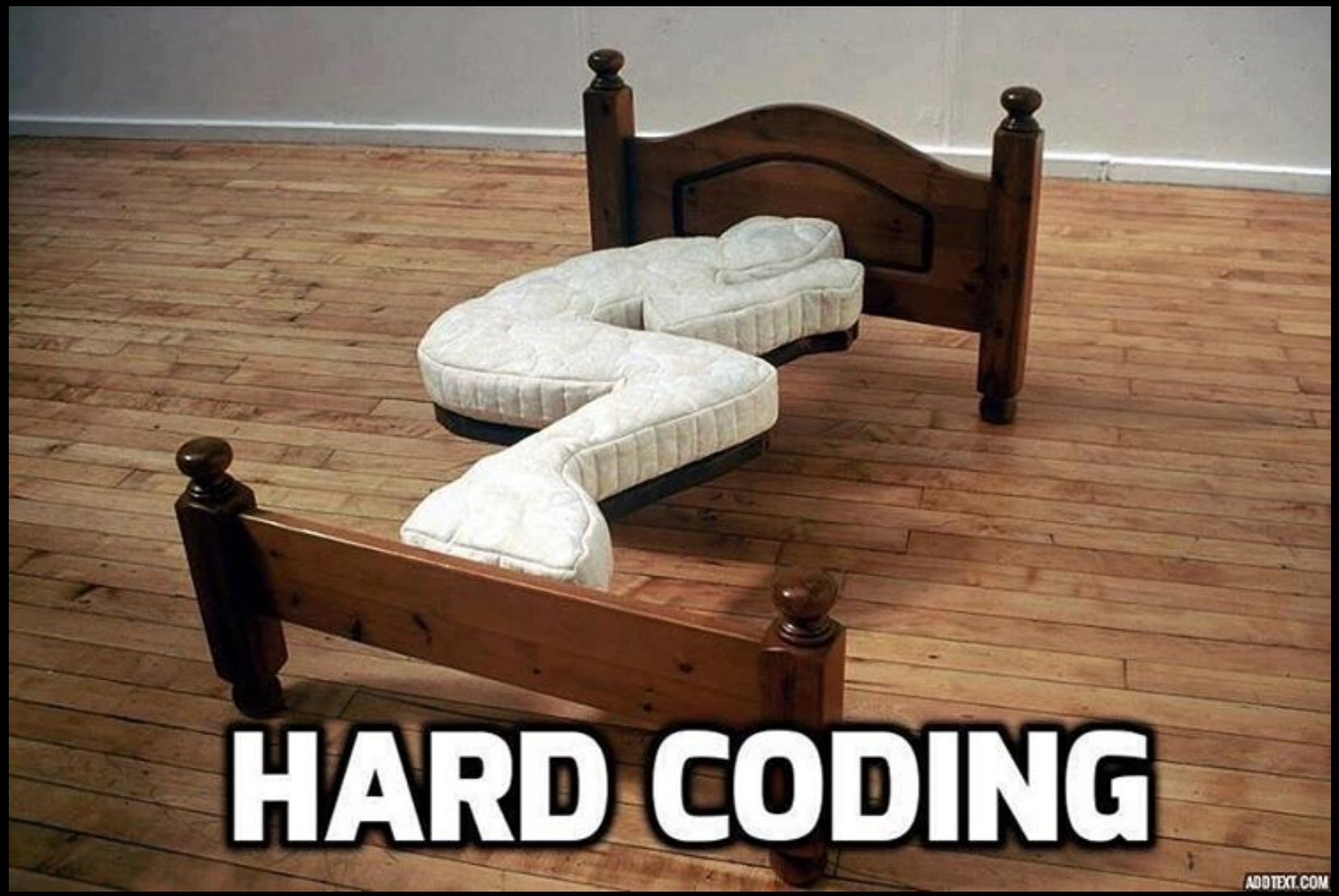
Why Kafka Stands Out

- ✅ High throughput — handles millions of events per second
- ✅ Durability — stores data on disk, replayable by consumers
- ✅ Scalability — horizontally scales with brokers and partitions
- ✅ Decouples producers & consumers — ideal for event-driven systems
- ✅ Closed Loop Automation – Consumption within different services
- ❌ Traditional queues (e.g., RabbitMQ) — focus on delivery, not storage



Designing Apps for Customization

Structuring configurations to avoid modifying core code


HARD CODING
ADDTEXT.COM

Why it matters

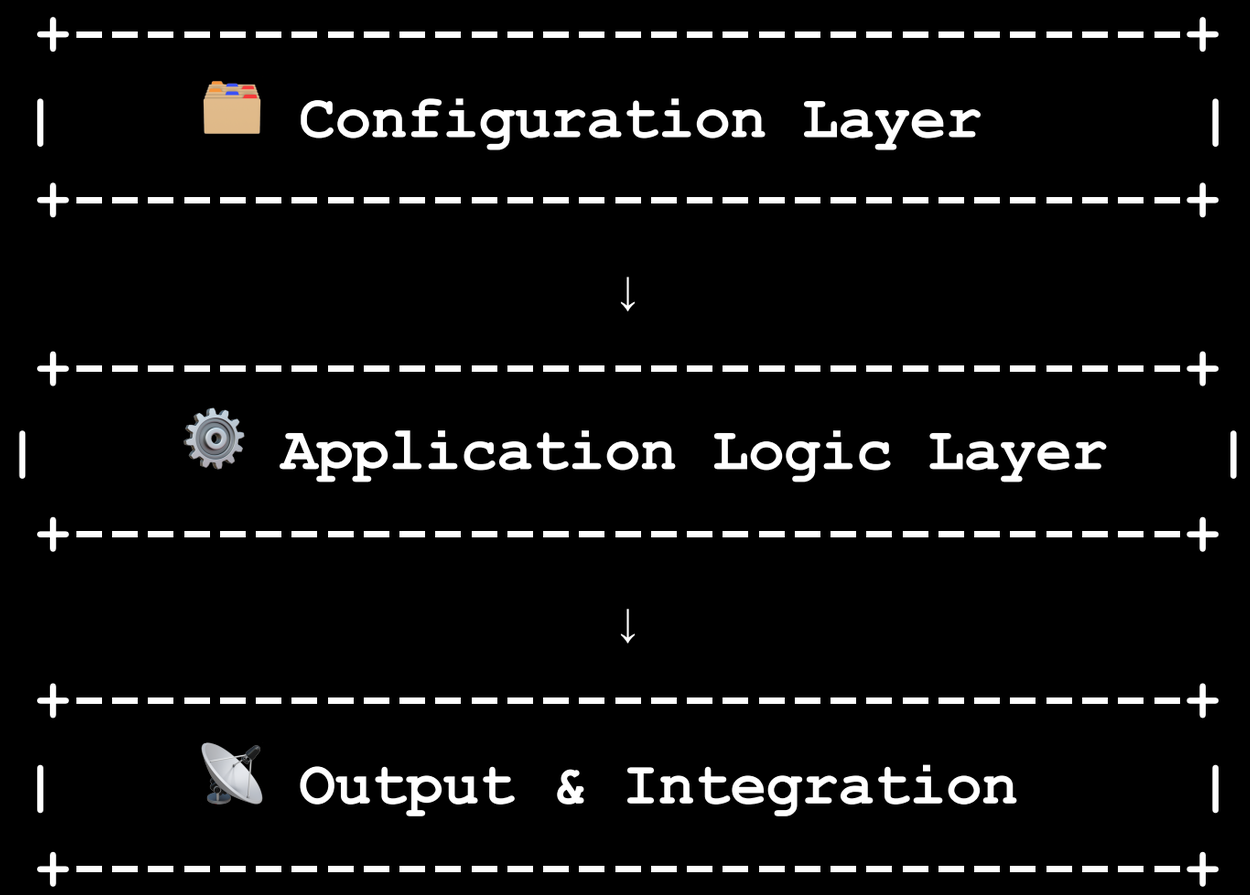
✅ Modular & Reusable

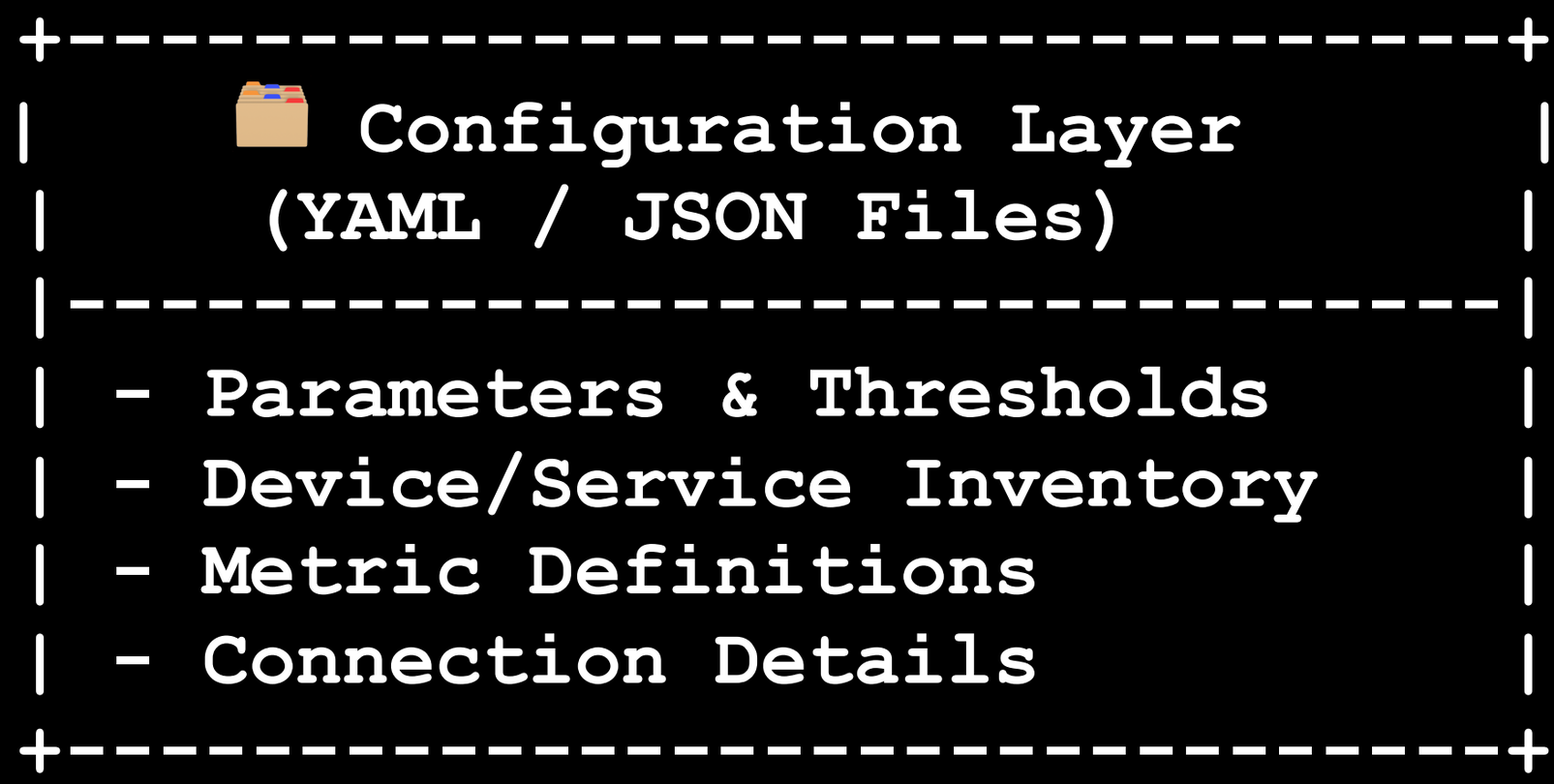
Separate logic from data so your code works across teams and projects.

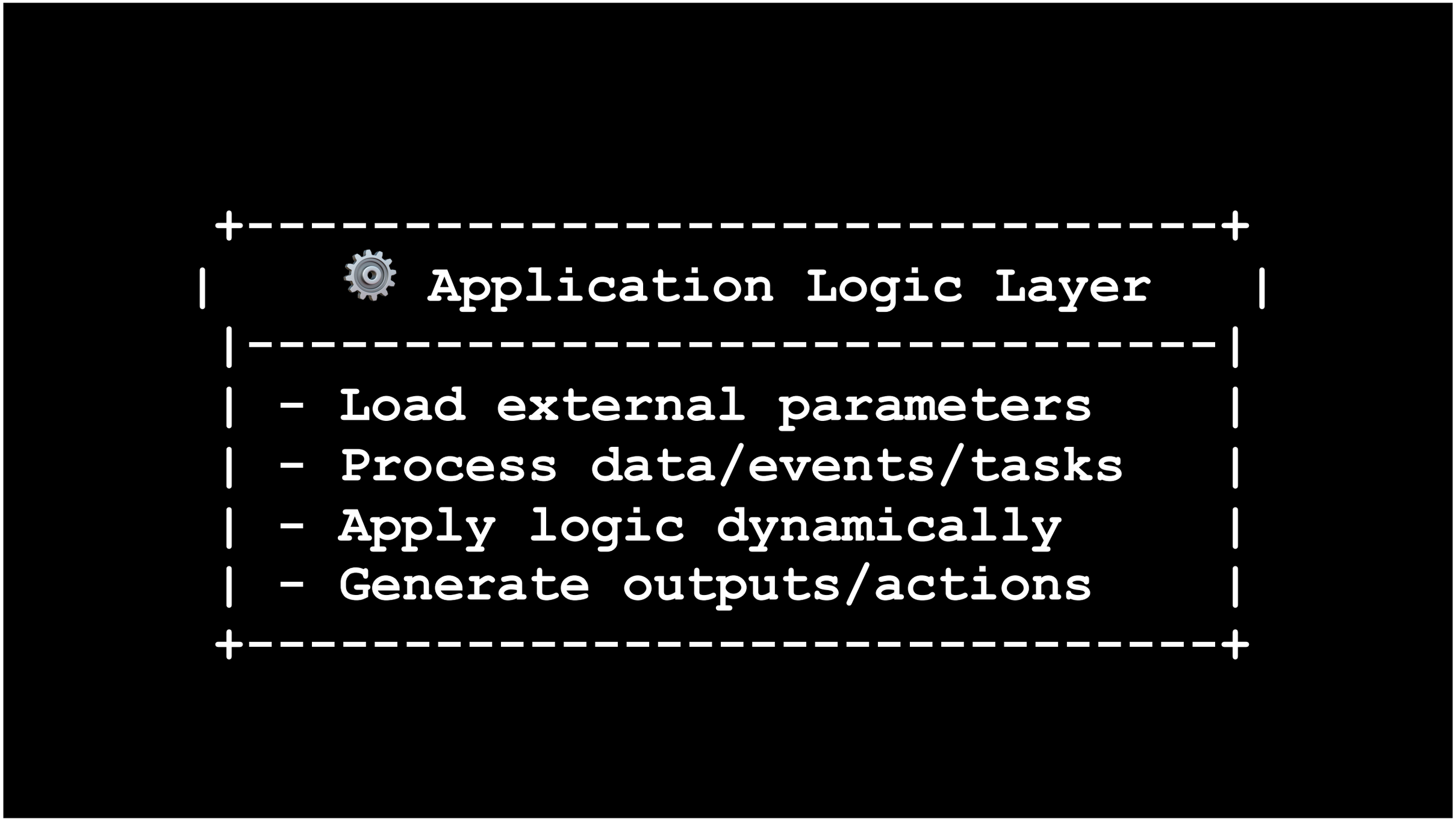
🔁 Adaptable to Any Use Case

Whether you're automating alarms, configs, or analytics — just update the config.

Workflow


Configuration Layer
Application Logic Layer
Output & Integration


Configuration Layer
(YAML / JSON Files)
- Parameters & Thresholds
- Device/Service Inventory
- Metric Definitions
- Connection Details


⚙ Application Logic Layer
- Load external parameters
- Process data/events/tasks
- Apply logic dynamically
- Generate outputs/actions

Output & Integration

- Prometheus / Grafana
- REST APIs / gNMI / Kafka
- CLI / NetConf / Webhooks
- Ansible / CI/CD / Reports

Configuration Layer

```
alarms:
  - topic: 'rt-anomalies’
    partition: 0
    alertType: 'baseline’
    counters:
      anomaly_detected:
        name: 'anomaly_detected’
        description: 'Anomaly detected in real-time telemetry’
        labels: ['neId', 'neName', 'counter']
```

⚙️ Application Logic Layer

```
parser.add_argument('--config', required=True, type=str,
       help='YAML file with alert topics and metrics’)
```

Application Logic Layer

```
with open(args.config, 'r') as file:
        config = yaml.safe_load(file)
```

Application Logic Layer

```
topic_partitions = [TopicPartition(alarm['topic’],
        alarm['partition']) for alarm in config['alarms’]]
consumer.assign(topic_partitions)
```

📡 Output & Integration

```
scrape_configs:
  - job_name: 'nsp-kafka’
    scrape_interval: 30s
    static_configs:
      - targets: ['kafka-alert-exporter:8001']
```

Kafka exporter: Key messages

🔧 **No hardcoding** of metric names, labels, topics, or thresholds.

📁 **External configuration** using YAML for alerts & metrics definition.

🔁 **Easier updates** — change behavior without modifying the source code.

Kafka exporter: Key messages (cont)

 **Scalable architecture** — add new topics/alerts via config files.

 **Secure and adaptable** — supports SSL and plaintext modes via parameters.

NETWORK AUTOMATION FORUM

GitHub Best Practices for Network Automation Projects

Branching Strategy

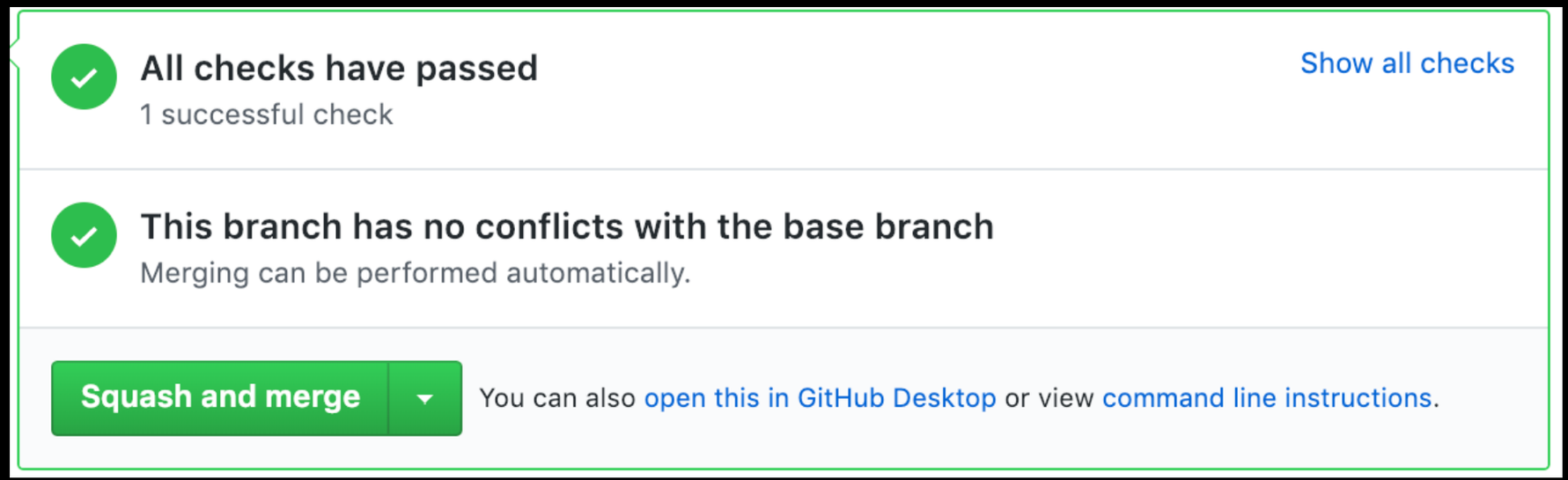
- main or release is always deployable
- Use feature branches (feature/xyz), bugfix (fix/xyz)

  git checkout -b feature/telemetry-parser
- Isolate work, test before merging
- Protect important branches via branch rules

Pull Requests & Code Review

- Open Pull Requests (PR) for merging
- Use clear titles + descriptions
- Tag reviewers
- Use review comments and suggestions
- Squash commits before merging (clean history)

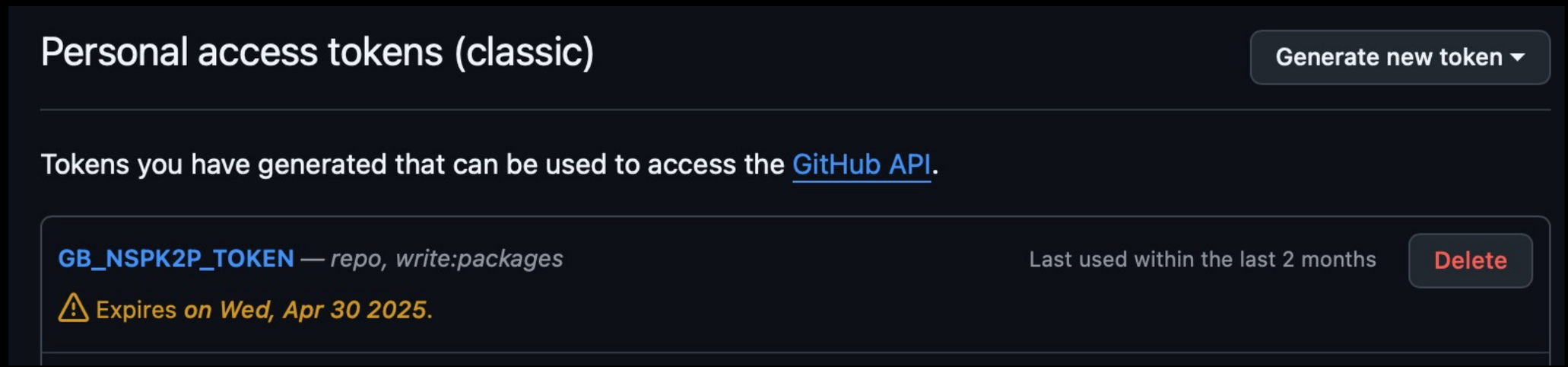
Pull Requests & Code Review



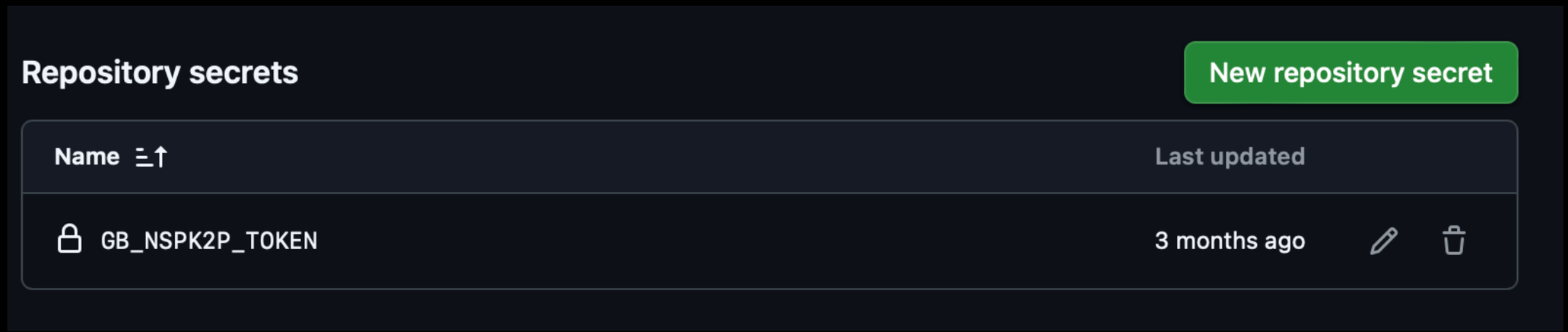
GitHub Actions (CI/CD?)

- Define workflows in .github/workflows/*.yml
- Run tests on PRs???
- Lint Python/YAML to catch format errors
- Auto-build & deploy on merge to main

GitHub Actions: Settings / Developer Settings



Github: Repo Settings / Secrets / Actions



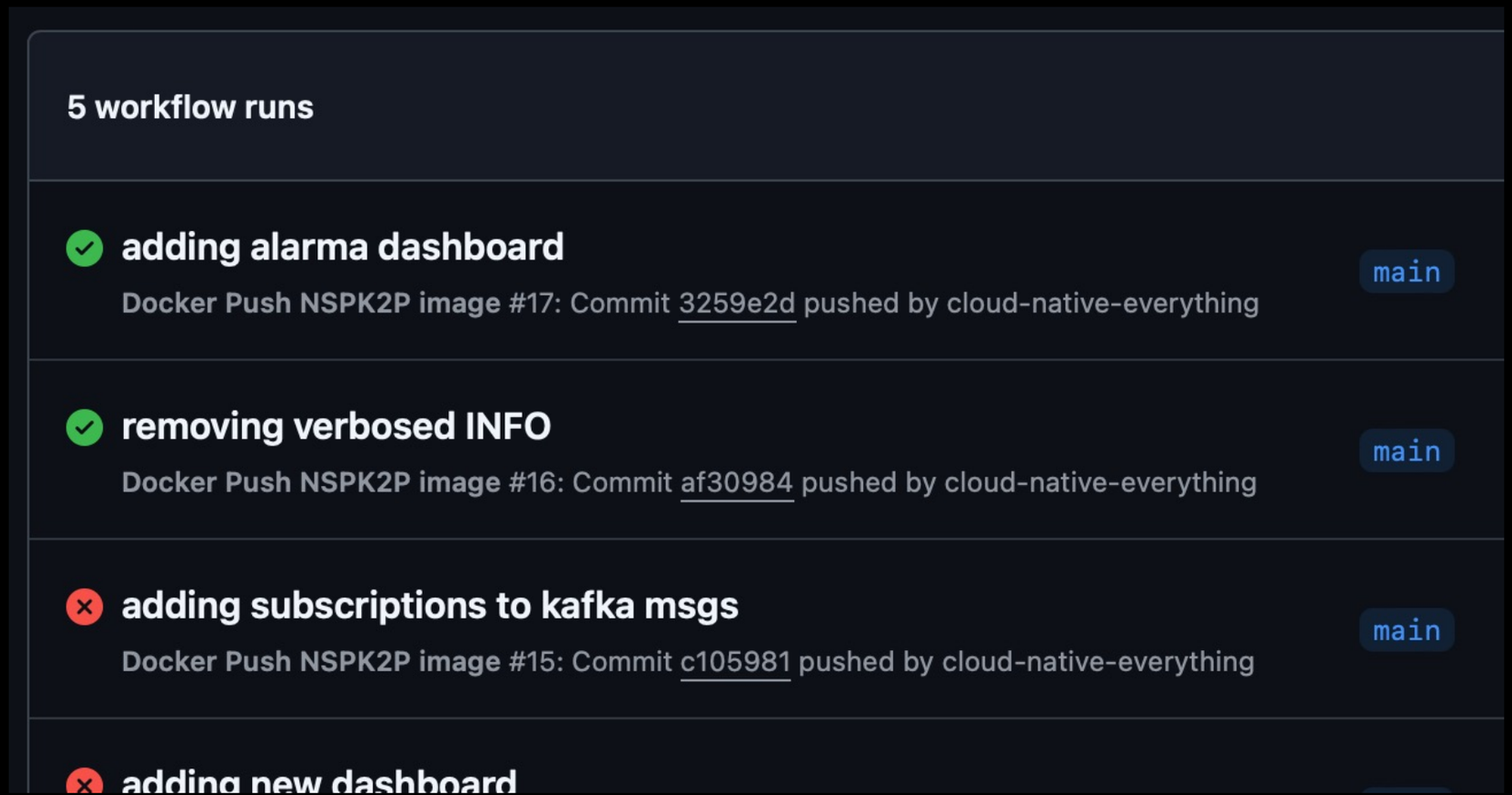
.github/workflows/docker-publish.yml

```yaml
name: Docker Push NSPK2P image
on:
  push:
env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}:v2501
jobs:
  build_and_publish:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3
```

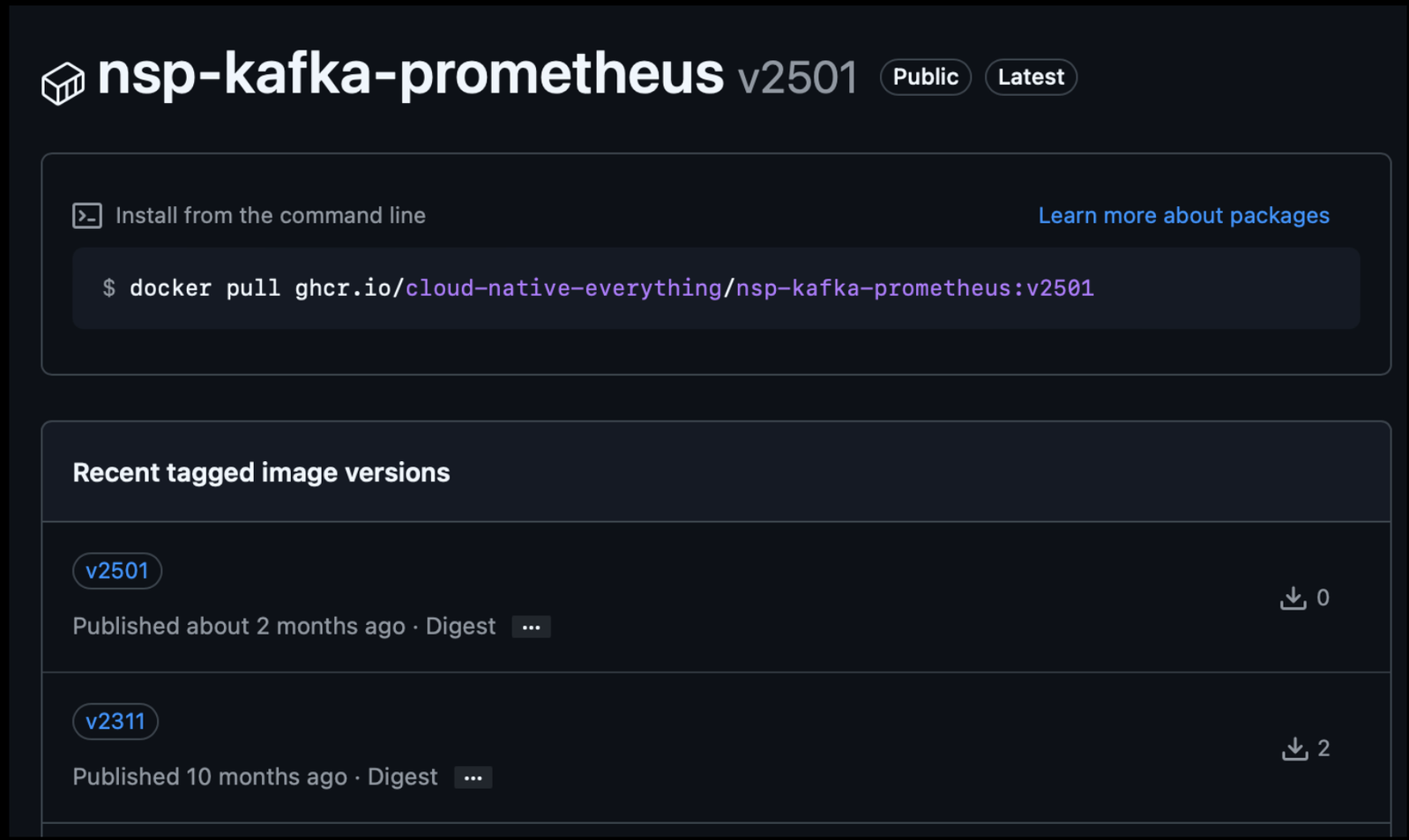
.github/workflows/docker-publish.yml (cont)

```
steps :
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Docker Login
        run: echo "${{ secrets.GB_NSPK2P_TOKEN }}" | docker login ${{
   env.REGISTRY }} --username cloud-native-everything --password-stdin

      - name: Docker Build
        run: docker build -t ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
   .

      - name: Docker Push
        run: docker push  ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
```

Github Repo Actions



Github Repo Packages




NAF
NETWORK AUTOMATION FORUM
Questions?