# **Lab: Prometheus + Grafana + Alertmanager Stack (Docker)**

**Goal:** Stand up a local observability stack and scrape Redpanda metrics.  
**Outcome:** See live **broker health** and **message flow** from one pane (Grafana).

# **Purpose of the Lab**

Stand up a complete, local **observability stack**—**Prometheus**, **Grafana**, and **Alertmanager**—alongside a single-node **Redpanda** cluster to **scrape**, **visualize**, and **alert on** broker health and streaming activity. The focus is on getting practical visibility into a developer environment (Docker) so learners can quickly verify message flow, spot issues (e.g., slow scrapes, broker down), and turn raw metrics into actionable insights.

# **Learning Outcomes**

By the end of this lab, learners will be able to:

* **Deploy** Prometheus, Grafana, Alertmanager, Redpanda, and Console with Docker Compose using correct networking and mounts.
* **Scrape Redpanda metrics** from the admin endpoint (/metrics on 9644) and validate target health in Prometheus.
* **Query metrics in Prometheus** using PromQL (e.g., up, rate()), including the **vectorized\_\*** metric family for produce/consume activity.
* **Provision Grafana automatically** (datasource + dashboard) and **visualize** broker health, scrape performance, and per-topic throughput.
* **Trigger and observe alerts** (e.g., RedpandaDown) and confirm resolution via Alertmanager.
* **Generate and interpret traffic signals** (produce/consume bursts) to confirm that dashboards and queries reflect real streaming behavior.
* **Troubleshoot common issues** (mount paths, DNS/targets, no-data panels, zsh URL quoting) using a repeatable checklist.
* **Adapt the stack** by adding new panels/queries, recording rules, and alert routes to fit team SLOs and real use cases.

## **Folder layout**

Create a clean folder (e.g., obs-lab/) and add these files:

|  |
| --- |
| obs-lab/ ├── alertmanager/ │ └── alertmanager.yml ├── grafana/ │ ├── dashboards/ │ │ └── redpanda-starter.json│ │ │ └── provisioning/ │ ├── dashboards/ │ │ └── dashboards.yml │ └── datasources/ │ └── datasource.yml ├── prometheus/ │ ├── prometheus.yml │ └── rules.yml └── docker-compose.yml |

## 

## **docker-compose.yml**

The docker-compose.yml file in this lab is the **orchestration blueprint** that defines and launches the complete observability stack. It declares the services—**Redpanda**, **Prometheus**, **Grafana**, and **Alertmanager**—and ensures they run together on a shared Docker network with the right ports exposed. Within the file, each service is configured with its own volumes for persistence (e.g., Redpanda data, Grafana dashboards), health checks to verify readiness, and environment variables to connect components seamlessly.

For example, Prometheus is wired to scrape Redpanda’s /metrics endpoint on port 9644, Grafana is provisioned with Prometheus as a datasource and preloaded dashboards, and Alertmanager is integrated with Prometheus to handle alert rules. In short, the file acts as the **single source of truth** for standing up a working end-to-end monitoring and visualization environment with one command (docker compose up -d).

Dual Kafka listeners so **containers** use redpanda:9092 and **host apps** use localhost:19092. We **don’t** set --http-addr; the admin API binds to 9644 by default.

|  |
| --- |
| version: "3.8"  services:  redpanda:  image: redpandadata/redpanda:latest  container\_name: redpanda  command:  - redpanda  - start  - --overprovisioned  - --smp=1  - --memory=1G  - --reserve-memory=0M  - --node-id=0  - --check=false  # Two Kafka listeners: internal (containers) + external (host)  - --kafka-addr=PLAINTEXT://0.0.0.0:9092,OUTSIDE://0.0.0.0:19092  - --advertise-kafka-addr=PLAINTEXT://redpanda:9092,OUTSIDE://localhost:19092  # RPC (node-to-node / client metadata)  - --rpc-addr=0.0.0.0:33145  - --advertise-rpc-addr=redpanda:33145  ports:  - "9092:9092" # containers use redpanda:9092  - "19092:19092" # host apps use localhost:19092  - "9644:9644" # admin/metrics endpoint  volumes:  - redpanda-data:/var/lib/redpanda/data  healthcheck:  test: ["CMD-SHELL", "curl -sf http://localhost:9644/v1/status/ready >/dev/null"]  interval: 5s  timeout: 3s  retries: 30  start\_period: 10s  networks: [redpanda-net]  restart: unless-stopped   console:  image: redpandadata/console:latest  container\_name: redpanda-console  depends\_on:  redpanda: { condition: service\_healthy }  environment:  KAFKA\_BROKERS: redpanda:9092  ports:  - "8080:8080"  networks: [redpanda-net]  restart: unless-stopped   prometheus:  image: prom/prometheus:latest  container\_name: prometheus  depends\_on:  redpanda: { condition: service\_healthy }  command:  - --config.file=/etc/prometheus/prometheus.yml  - --storage.tsdb.retention.time=3d  ports:  - "9090:9090"  volumes:  - ./prometheus:/etc/prometheus  networks: [redpanda-net]  restart: unless-stopped   alertmanager:  image: prom/alertmanager:latest  container\_name: alertmanager  ports:  - "9093:9093"  volumes:  - ./alertmanager/alertmanager.yml:/etc/alertmanager/config.yml:ro  networks: [redpanda-net]  restart: unless-stopped   grafana:  image: grafana/grafana:latest  container\_name: grafana  depends\_on:  prometheus:  condition: service\_started  environment:  GF\_SECURITY\_ADMIN\_USER: admin  GF\_SECURITY\_ADMIN\_PASSWORD: admin  GF\_AUTH\_ANONYMOUS\_ENABLED: "false"  ports:  - "3000:3000"  volumes:  - ./grafana/provisioning:/etc/grafana/provisioning  - ./grafana/dashboards:/var/lib/grafana/dashboards  networks: [redpanda-net]  restart: unless-stopped  volumes:  redpanda-data:  networks:  redpanda-net: |

## **Prometheus config**

**prometheus/prometheus.yml**

The **prometheus.yml** file defines how Prometheus scrapes metrics from Redpanda and integrates with Alertmanager. It contains **global settings** (such as scrape interval and evaluation interval), and one or more **scrape\_configs** that tell Prometheus *where* to pull metrics from. In this lab, the key job is redpanda, which targets the Redpanda admin API at redpanda:9644/metrics inside the Docker network. This ensures that Prometheus continuously collects broker, topic, and consumer-related metrics that Redpanda exposes.

|  |
| --- |
| global:  scrape\_interval: 15s  evaluation\_interval: 15s  rule\_files:  #- /etc/prometheus/rules.yml  scrape\_configs:  - job\_name: redpanda  metrics\_path: /metrics  static\_configs:  - targets: ["redpanda:9644"]  # optional relabels here if you want to add env/cluster labels   - job\_name: prometheus  static\_configs:  - targets: ["prometheus:9090"]   - job\_name: alertmanager  static\_configs:  - targets: ["alertmanager:9093"] |

**prometheus/rules.yml**

The config also references the rules.yml file for custom alerting rules (e.g., broker down, high error rate) and points to Alertmanager (alertmanager:9093) so firing alerts can be routed to notifications. In short, this YAML acts as Prometheus’ **playbook**, controlling what data is collected, how often, and where alerts should be sent.

|  |
| --- |
| groups: - name: redpanda.rules  rules:  - alert: RedpandaDown  expr: up{job="redpanda"} == 0  for: 1m  labels: { severity: critical }  annotations:  summary: "Redpanda broker is down"  description: "The 'up' metric for job=redpanda is 0 for 1 minute."   - alert: RedpandaScrapeSlow  expr: scrape\_duration\_seconds{job="redpanda"} > 1  for: 5m  labels: { severity: warning }  annotations:  summary: "Redpanda scrape is slow (>1s)"  description: "scrape\_duration\_seconds > 1s for 5 minutes."   - alert: RedpandaMetricsMissing  expr: absent(up{job="redpanda"})  for: 5m  labels: { severity: warning }  annotations:  summary: "No Redpanda metrics found"  description: "Prometheus cannot find 'up{job=\"redpanda\"}'." |

## **Alertmanager config**

**alertmanager/alertmanager.yml**

The **alertmanager.yml** file defines how **Alertmanager** handles alerts sent from Prometheus. It doesn’t generate alerts itself (that happens via rules.yml in Prometheus), but it specifies the **routing, grouping, and notification policies** once an alert fires.

|  |
| --- |
| route:  receiver: "dev-null"  group\_by: ["alertname"]  group\_wait: 10s  group\_interval: 2m  repeat\_interval: 2h  receivers:  - name: "dev-null" |

## **Grafana provisioning**

**grafana/provisioning/datasources/datasource.yml**

The **datasource.yml** file is part of Grafana’s **provisioning system**, which allows you to preconfigure connections to data sources automatically when Grafana starts. Instead of adding Prometheus manually through the Grafana UI, this YAML ensures the datasource is created on first boot and persists across restarts.

|  |
| --- |
| apiVersion: 1 datasources:  - name: Prometheus  type: prometheus  access: proxy  url: http://prometheus:9090  isDefault: true  editable: true |

**grafana/provisioning/dashboards/dashboards.yml**

The **dashboards.yml** file is part of Grafana’s provisioning mechanism that tells Grafana **where to find JSON dashboards on disk** and how to load them automatically when Grafana starts. Instead of manually importing dashboards one by one via the UI, this file points Grafana at a directory (e.g., /var/lib/grafana/dashboards) where JSON files like redpanda-starter.json live.

|  |
| --- |
| apiVersion: 1 providers:  - name: 'Starter'  orgId: 1  folder: 'Starter'  type: file  disableDeletion: false  updateIntervalSeconds: 10  allowUiUpdates: true  options:  path: /var/lib/grafana/dashboards  foldersFromFilesStructure: false |

**grafana/dashboards/redpanda-starter.json** (minimal starter dashboard)

The **redpanda-starter.json** file is a **prebuilt Grafana dashboard** expressed in JSON format. It provides a ready-to-use visualization of Redpanda’s most important metrics, so learners don’t need to build panels from scratch. When provisioned via dashboards.yml, this JSON file is automatically loaded into Grafana, appearing under the **Starter** folder.

|  |
| --- |
| {  "schemaVersion": 39,  "title": "Redpanda Starter (Prometheus)",  "time": { "from": "now-30m", "to": "now" },  "panels": [  {  "type": "stat",  "title": "Broker Up",  "gridPos": { "h": 6, "w": 6, "x": 0, "y": 0 },  "targets": [{ "expr": "sum(up{job=\"redpanda\"})" }]  },  {  "type": "timeseries",  "title": "Scrape Duration (s)",  "gridPos": { "h": 6, "w": 12, "x": 6, "y": 0 },  "targets": [{ "expr": "max(scrape\_duration\_seconds{job=\"redpanda\"})" }]  },  {  "type": "timeseries",  "title": "Produce Rate by Topic (5m)",  "gridPos": { "h": 8, "w": 18, "x": 0, "y": 6 },  "targets": [{  "expr": "sum by (topic) (rate(vectorized\_cluster\_partition\_batches\_produced{namespace=\"kafka\"}[5m]))"  }]  }  ] } |

The third panel is a **discovery helper**. In Grafana **Explore**, type redpanda\_ and use auto-complete to find the concrete metrics your build exposes (then swap into panels). This keeps the lab compatible across versions.

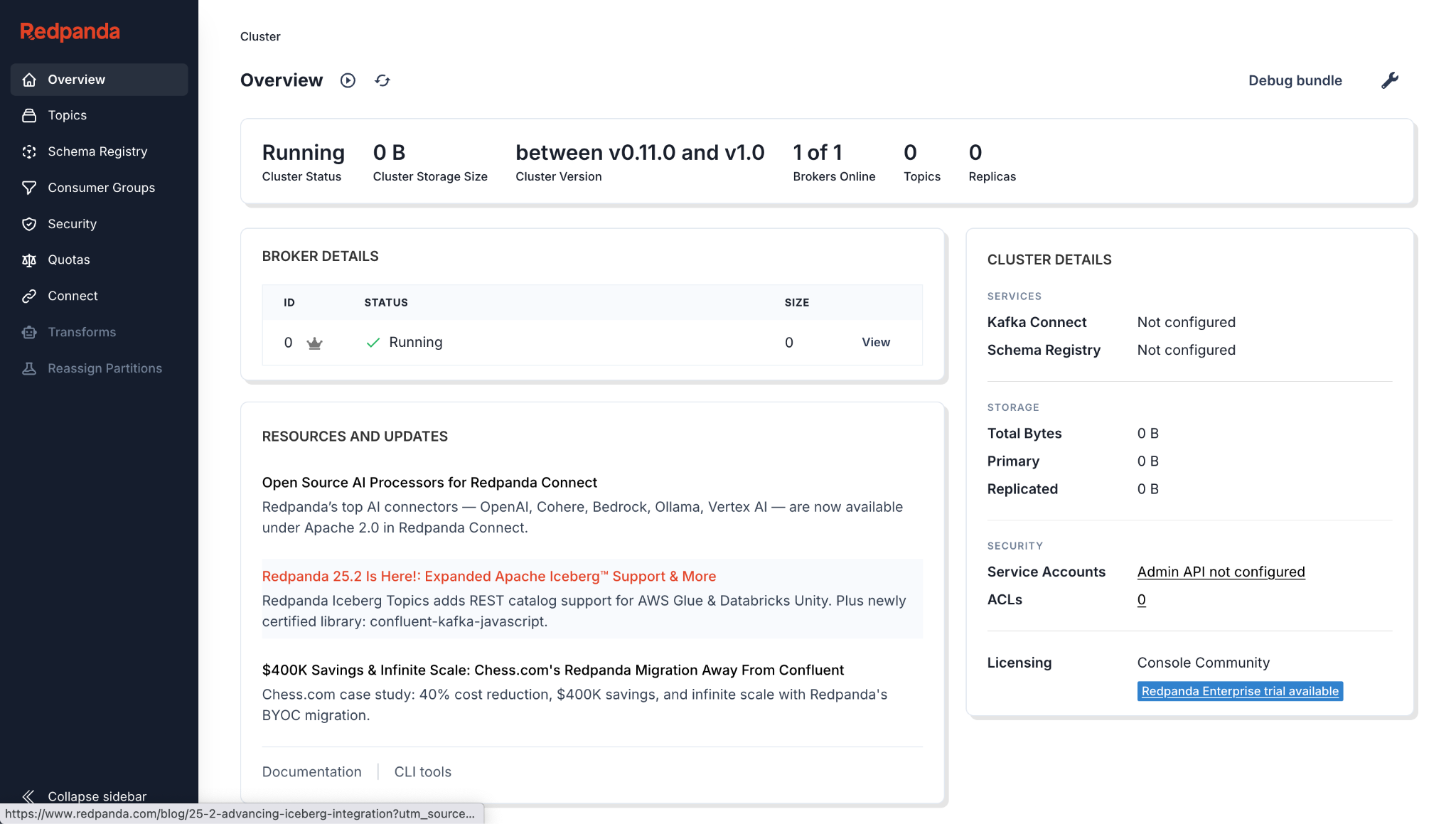
## **Bring the stack up**

From obs-lab/:

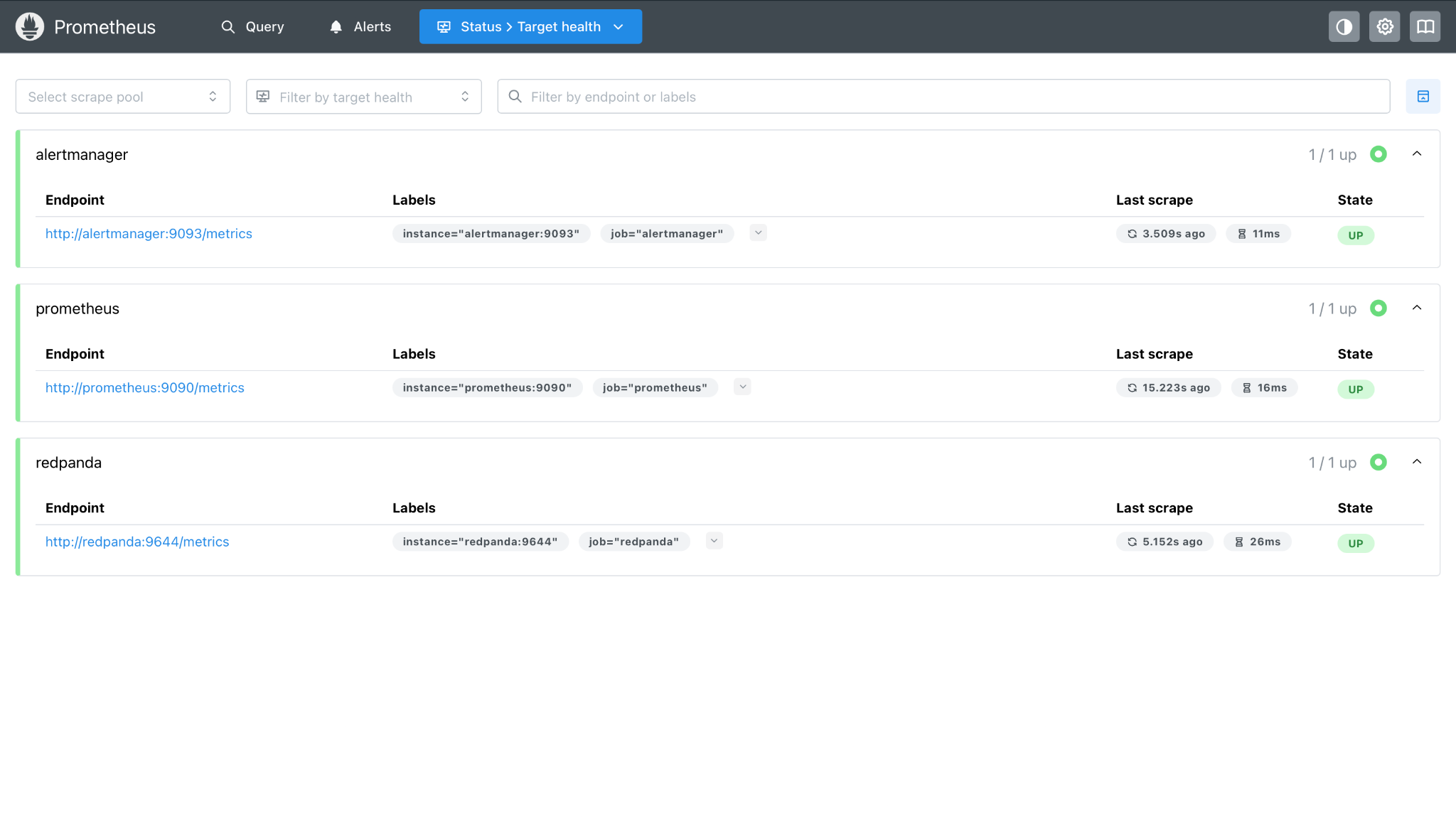
|  |
| --- |
| docker compose up -d docker compose ps |

**Open UIs**

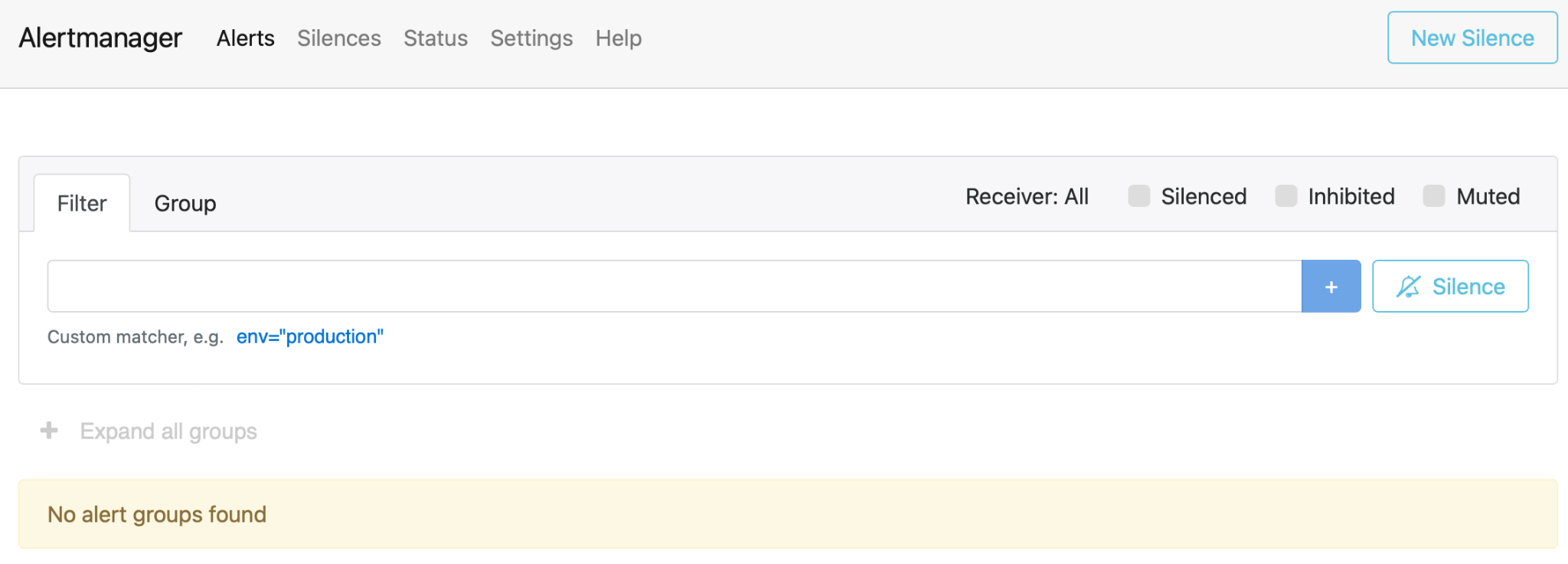
* **Console:** [http://localhost:8080](http://localhost:8080/)



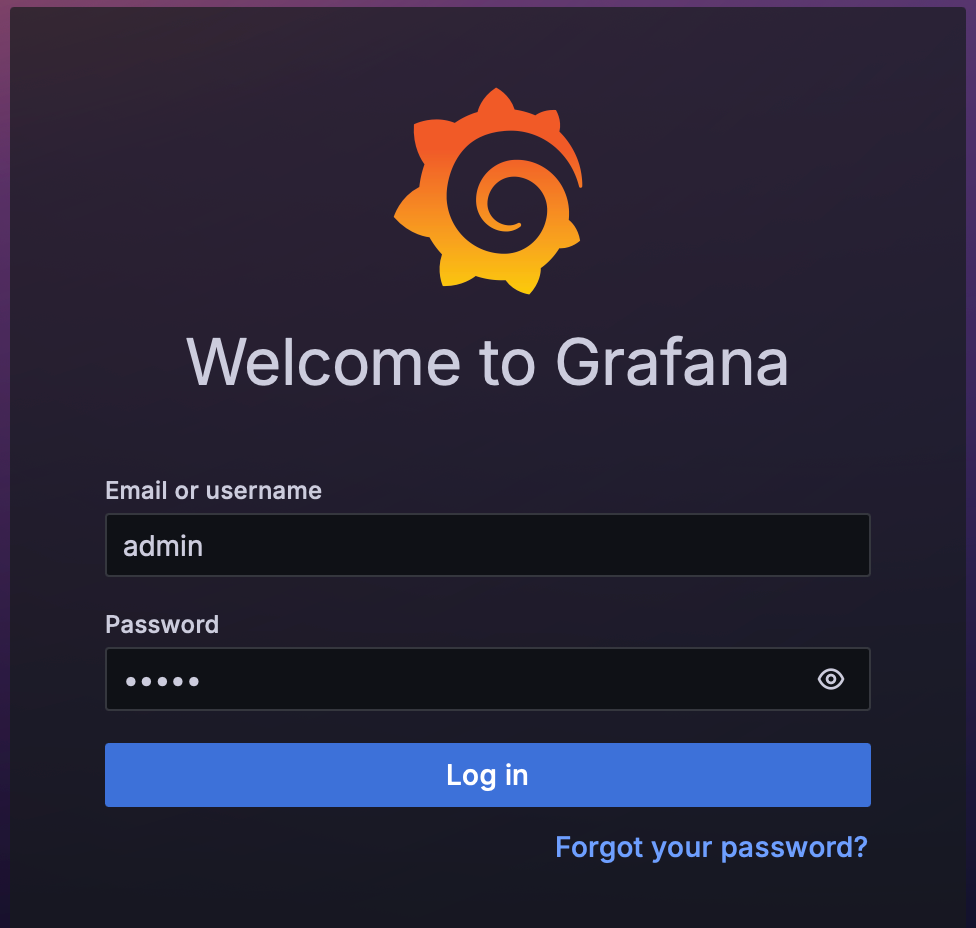
* **Prometheus:** [http://localhost:9090](http://localhost:9090/) (Status → Targets should show **redpanda** UP)

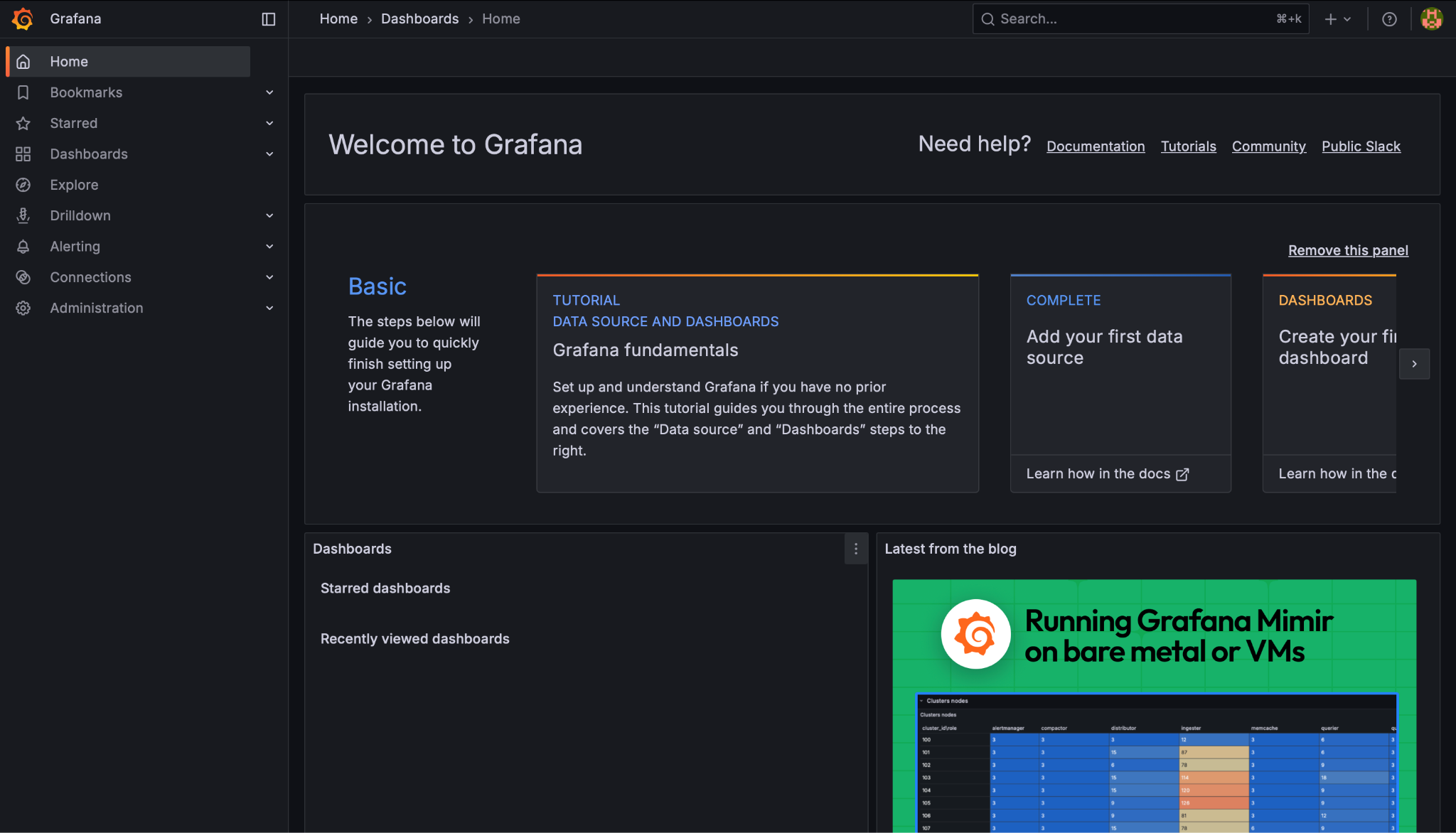


* **Alertmanager:** [http://localhost:9093](http://localhost:9093/)



* **Grafana:** [http://localhost:3000](http://localhost:3000/) (login **admin/admin**)





## **Generate a little traffic (optional but recommended)**

Create a topic and produce a few messages (inside the Redpanda container):

|  |
| --- |
| docker exec -it redpanda rpk topic create metrics\_demo -p 3 -r 1 --brokers redpanda:9092 printf "a\nb\nc\n" | docker exec -i redpanda rpk topic produce metrics\_demo --brokers redpanda:9092 docker exec -it redpanda rpk topic consume metrics\_demo --num 3 --brokers redpanda:9092 |

This helps ensure the broker is doing work while you watch metrics.

## **Verify metrics & queries**

This step focuses on validating that Prometheus is successfully scraping Redpanda metrics and that learners can interactively explore them.

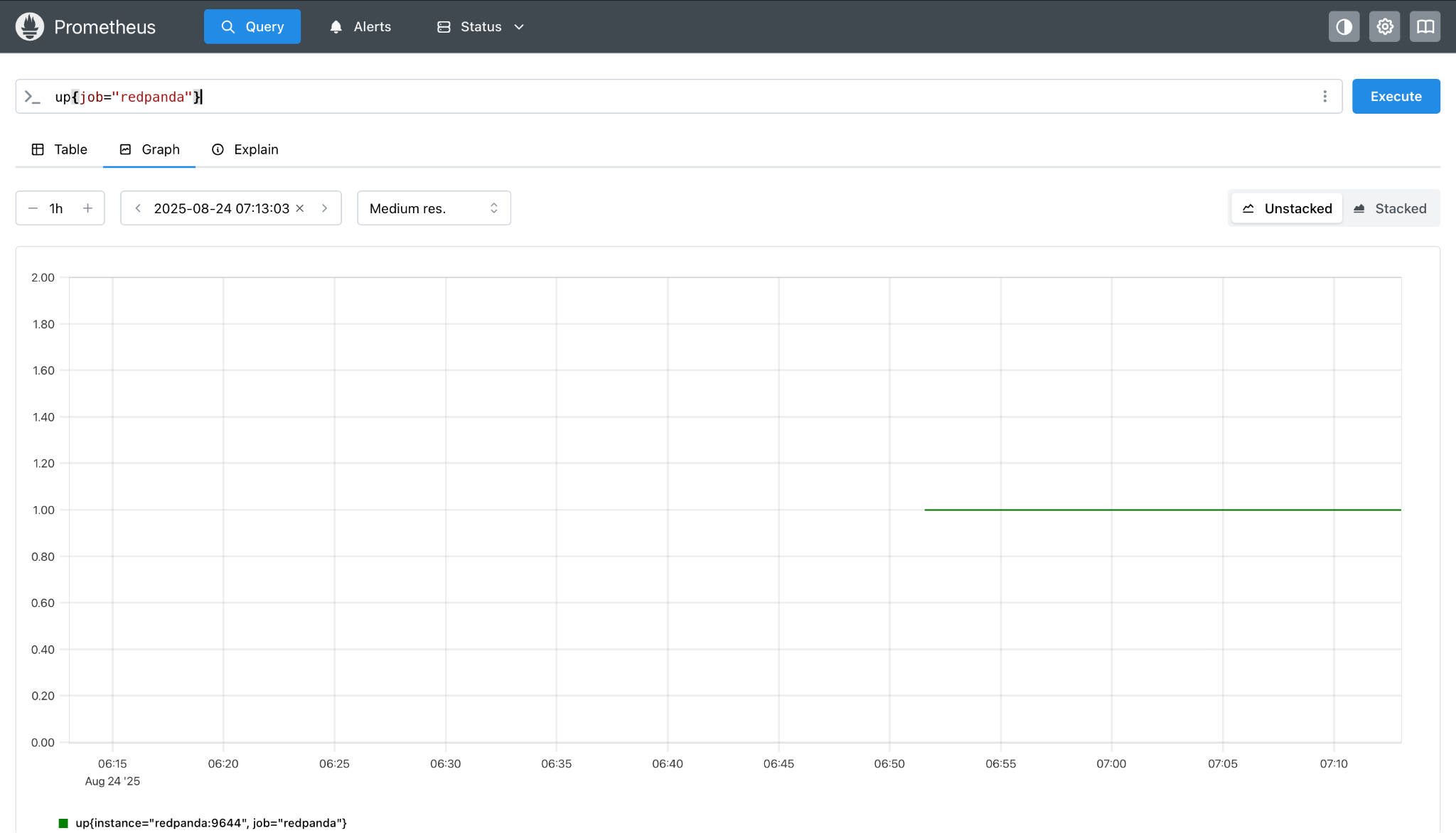
By running simple PromQL expressions such as up{job="redpanda"}, learners confirm broker health; with discovery queries like count by(\_\_name\_\_)({job="redpanda"}), they see the full set of available metric names.

From there, rate queries (e.g., rate(vectorized\_cluster\_partition\_batches\_produced[5m])) demonstrate throughput by topic, while fetch-related metrics show consumer activity. The exercise builds confidence in reading raw metrics directly in Prometheus before visualizing them in Grafana, and ensures learners understand how to move from health checks to performance insights.

In **Prometheus → Graph**:

Basic health:

|  |
| --- |
| up{job="redpanda"} |



## **Make sure the file exists on the host in the right path**

Run these from your **obs-lab** folder (same place as docker-compose.yml):

|  |
| --- |
| mkdir -p grafana/dashboards grafana/provisioning/datasources grafana/provisioning/dashboards  # dashboards.yml (puts dashboards into a visible "Starter" folder) cat > grafana/provisioning/dashboards/dashboards.yml <<'YAML' apiVersion: 1 providers:  - name: 'Starter'  orgId: 1  folder: 'Starter'  type: file  disableDeletion: false  updateIntervalSeconds: 10  allowUiUpdates: true  options:  path: /var/lib/grafana/dashboards  foldersFromFilesStructure: false YAML  # datasource.yml (points Grafana at Prometheus in the compose network) cat > grafana/provisioning/datasources/datasource.yml <<'YAML' apiVersion: 1 datasources:  - name: Prometheus  type: prometheus  access: proxy  url: http://prometheus:9090  isDefault: true  editable: true YAML  # minimal starter dashboard JSON cat > grafana/dashboards/redpanda-starter.json <<'JSON' {  "schemaVersion": 39,  "title": "Redpanda Starter (Prometheus)",  "time": { "from": "now-30m", "to": "now" },  "panels": [  {  "type": "stat",  "title": "Broker Up",  "gridPos": { "h": 6, "w": 6, "x": 0, "y": 0 },  "targets": [{ "expr": "sum(up{job=\"redpanda\"})" }]  },  {  "type": "timeseries",  "title": "Scrape Duration (s)",  "gridPos": { "h": 6, "w": 12, "x": 6, "y": 0 },  "targets": [{ "expr": "max(scrape\_duration\_seconds{job=\"redpanda\"})" }]  },  {  "type": "timeseries",  "title": "Produce Rate by Topic (5m)",  "gridPos": { "h": 8, "w": 18, "x": 0, "y": 6 },  "targets": [{  "expr": "sum by (topic) (rate(vectorized\_cluster\_partition\_batches\_produced{namespace=\"kafka\"}[5m]))"  }]  }  ] } JSON |

## **Recreate Grafana so it re-reads the mounts**

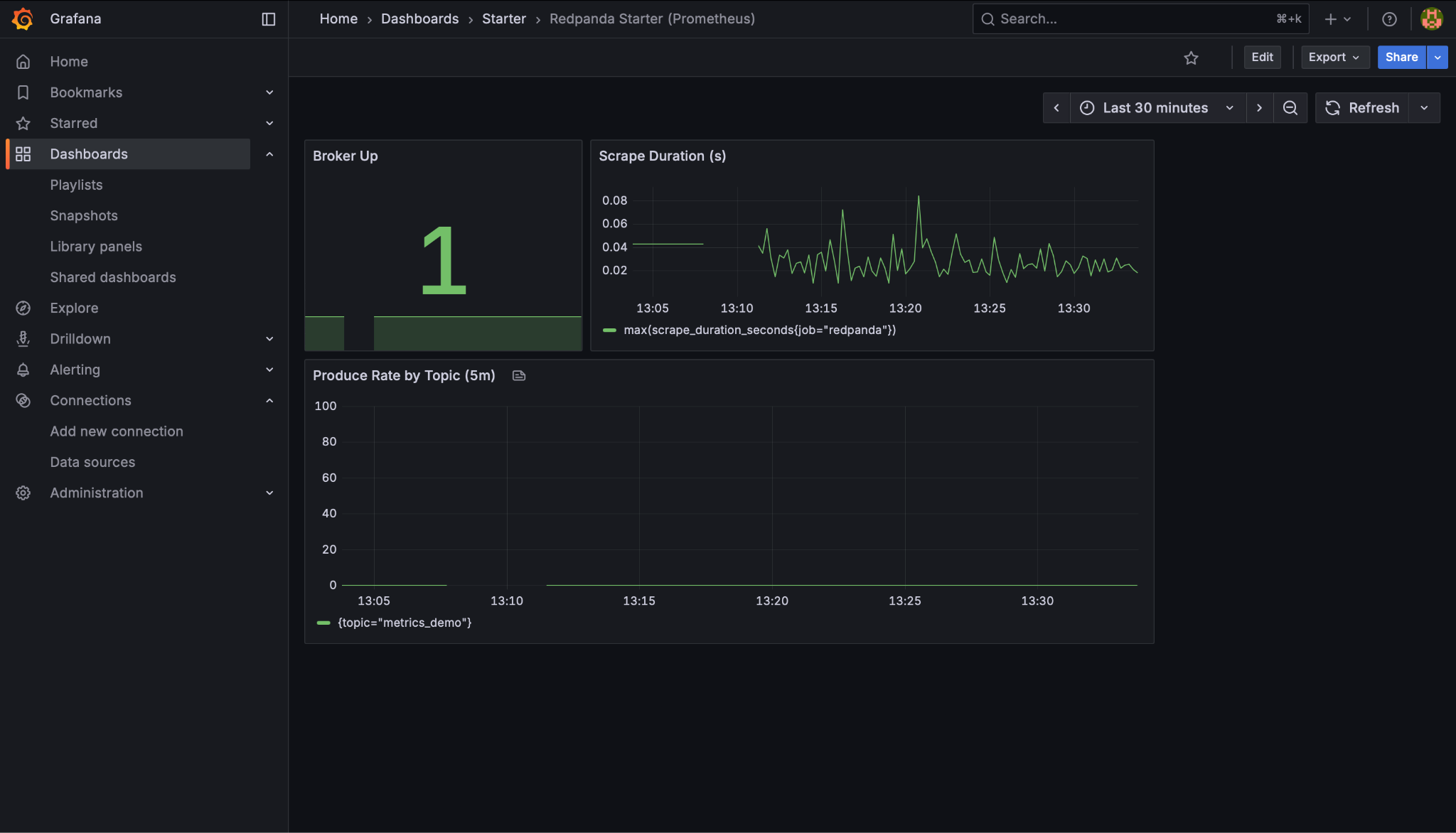
|  |
| --- |
| docker compose stop grafana docker compose rm -f grafana docker compose up -d grafana |

## **Verify the mount & provisioning, then open the dashboard**

|  |
| --- |
| # File should now exist INSIDE the container: docker exec -it grafana bash -lc 'ls -l /var/lib/grafana/dashboards'  # Check provisioning logs: docker logs -n 200 grafana | egrep -i "provision|dashboard|datasource" |

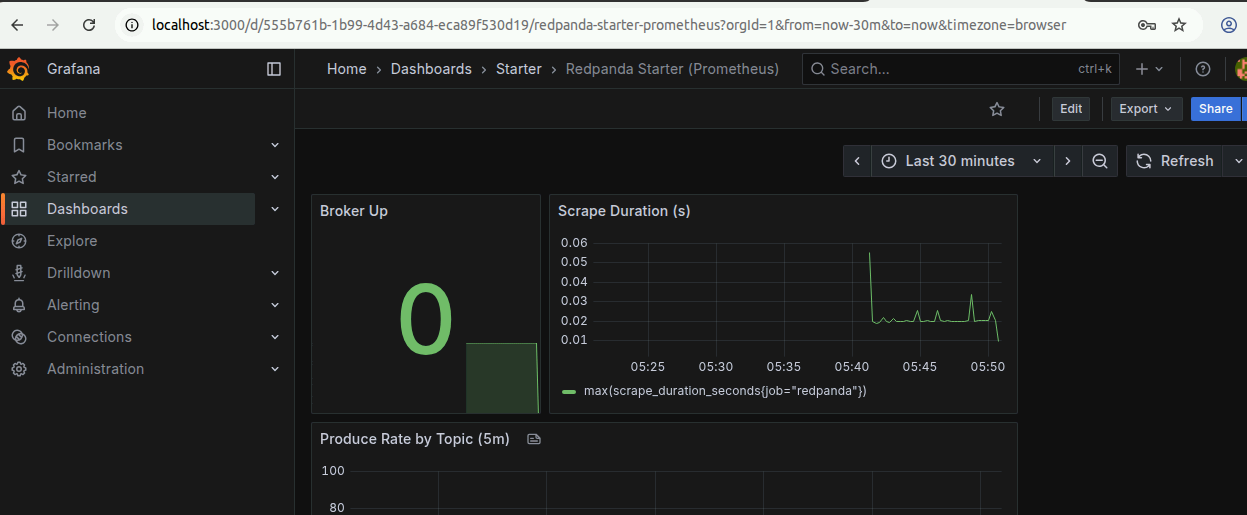
In Grafana (http://localhost:3000 → admin/admin):

* **Dashboards → Browse → Starter** → open **Redpanda Starter (Prometheus)**.



Stop Redpanda:

|  |
| --- |
| docker stop redpanda |



Start Redpanda:

|  |
| --- |
| docker start redpanda |

## A screenshot of a computer AI-generated content may be incorrect. **Cleanup**

|  |
| --- |
| docker compose down # or nuke data too: docker compose down -v |

## **Success checklist**

* redpanda target **UP** in Prometheus.
* Grafana shows **Broker Up** and activity panels.
* You can run PromQL using **vectorized\_** metrics (produce/read).
* Stopping/starting Redpanda triggers and resolves **RedpandaDown**.

That’s your updated, working observability stack for Redpanda.