GARNET | Ghanaian Academic and Research Network

DAY 1 · SESSION 1

Introduction to Monitoring & Architecture

Why we monitor, what we measure, and how Prometheus fits in

THE PROBLEM

Why do we monitor?

Reactive operations

Users tell you the network is down before your team notices.

No root cause

Something is slow — but is it the link, the router, the upstream, or the application?

No baseline

Without history, you cannot say what 'normal' looks like, so you cannot define 'broken'.

No accountability

SLA reports to clients become guesswork. Was uptime 99.5% or 98%? You don't really know.

THE LANDSCAPE

Three pillars of observability

Metrics

Numbers over time

CPU %, bytes/sec, request count. Cheap to store, fast to query, perfect for dashboards and alerts.

OUR FOCUS

Logs

Discrete events

'User X logged in at 14:32.' Detailed but verbose. Great for after-the-fact investigation.

Traces

Request journeys

Follow one request through every service it touches. Essential for distributed systems.

A KEY DESIGN CHOICE

Pull vs Push monitoring

PULL

Server scrapes targets

Prometheus, Nagios

- Server controls the schedule
- Easy to detect a target that is down (no response)
- Targets stay simple — just expose metrics
- Harder across NATs and firewalls

PUSH

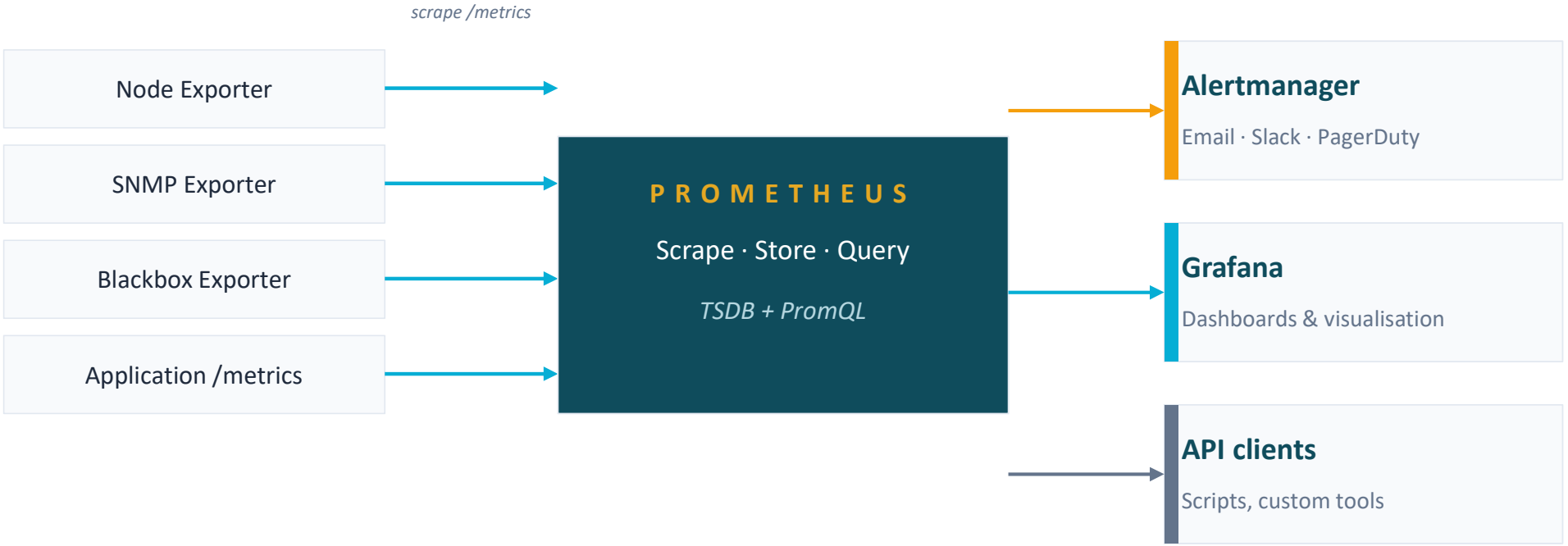
Targets send to server

StatsD, InfluxDB (often), Graphite

- Targets choose when to send
- Friendly to short-lived jobs and batch tasks
- Works through NATs without inbound rules
- If a target is silent, is it dead or healthy?

ARCHITECTURE

The Prometheus ecosystem



Three loosely-coupled pieces. You can swap any of them.

DATA MODEL

Everything is a time series

```
ifHCInOctets{instance="core-rt-1", ifName="GigE0/1"}
```

metric name + a unique set of labels = one time series

That series accumulates samples over time:

Timestamp	Value (bytes)
14:00:00	8,201,344,512
14:00:15	8,203,891,002
14:00:30	8,206,447,891

WHY THIS TOOL

What makes Prometheus different

Pull model

Server initiates scrapes. Down-detection comes for free.

Dimensional data

Labels turn one metric into many slice-able series.

PromQL

A query language built specifically for time-series math.

Single binary

No external database. Local disk, no Kafka, no Cassandra.

Service discovery

Targets discovered automatically from Consul, DNS, files, Kubernetes.

Battle-tested

CNCF graduated, used at SoundCloud, GitLab, DigitalOcean, GitHub.

HONEST LIMITS

Where it fits — and where it doesn't

Excellent fit

- Network device monitoring (SNMP)
- Server and host metrics
- Service availability (ping / HTTP)
- Application performance metrics
- Capacity planning baselines
- Real-time alerting

Wrong tool

- Log search and analysis (use Loki / ELK)
- Distributed tracing (use Tempo / Jaeger)
- Billing-grade accuracy (sampling loses precision)
- Years of high-resolution history
- Per-event auditing or compliance

BEFORE WE TOUCH THE LAB

Why Docker, not 'real' installs?

Same setup for everyone

Identical software, identical versions, identical behaviour on every participant's lab. No "works on my machine".

vs. each laptop fighting its own OS, package manager, version conflicts

Minutes, not hours

`docker compose up` brings up eight services in well under a minute.

vs. installing Prometheus, Grafana, Alertmanager, three exporters, configuring users, opening ports, debugging — half a day

Throwaway and reproducible

Break it during a practical? `docker compose down -v` and you're back to clean state.

vs. uninstalling, cleaning config files, hoping you got every leftover

Same artefact everywhere

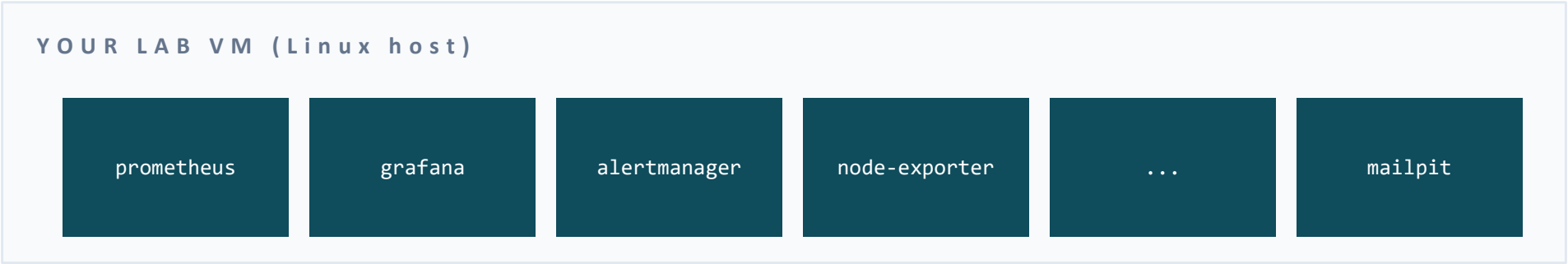
The exact docker-compose.yml from this workshop runs on your laptop, on a server at your institution, on a cloud VM.

vs. learning a different deployment story for every environment

Containers won because they made the unit of deployment the same as the unit of development.

BEFORE WE TOUCH THE LAB

Our setup runs on Docker



Each service is a container — packaged code + dependencies, isolated from the host. We run them all together with Docker Compose.

Three commands you'll use a lot

`docker compose up -d`	Start every service in the background
`docker compose ps`	Show what's running and its health
`docker compose logs <name>`	See what one service is saying

READING THE YAML

What's in a service definition?

Here's the prometheus block from our docker-compose.yml. Six concepts you'll meet on every service.

```yaml
1 prometheus:
2   image: prom/prometheus:v3.11.2
3   container_name: prometheus
4   ports:
      - "9090:9090"
5   volumes:
      - ./prometheus.yml:/etc/Prometheus/conf:ro
      - prometheus-data:/prometheus
6   networks:
      - garnet
```

1. **Service name**
   What you call this service inside compose. Used by `docker compose logs prometheus`.
2. **Image**
   Pre-built software package + version tag. Pulled from Docker Hub on first run.
3. **Container name**
   Friendly name in `docker ps`. Optional — defaults to a generated one.
4. **Ports**
   Map host:container. Visiting localhost:9090 reaches the container's :9090.
5. **Volumes**
   Mount files (`./path:/...`) or named volumes (`prometheus-data:/...`) for persistent data.
6. **Networks**
   Virtual bridge so services can find each other by name (`http://prometheus:9090`).

WHAT'S IN THE STACK

The cast of characters

CORE

Prometheus

`:9090`

Scrapes metrics, stores, evaluates rules

Grafana

`:3000`

Dashboards on top of the metrics

Alertmanager

`:9093`

Routes alerts to channels

COLLECTORS

Node Exporter

`:9100`

Host metrics: CPU, mem, disk, net

SNMP Exporter

`:9116`

Translates SNMP to Prometheus format

Blackbox Exporter

`:9115`

External probes: ping, HTTP, TCP

PRACTICAL HELPERS

Mailpit

`:8025`

Local SMTP + web inbox for alerts

Webhook Receiver

`:9000`

Catches Alertmanager webhooks

Eight services, one network, one config repo. Bookmark this slide — you'll come back to it.

THE ARC

Where this workshop is going

DAY 1	**Foundations & first pipeline** Concepts, environment, Prometheus basics, PromQL, Grafana intro.
DAY 2	**Network monitoring & service discovery** SNMP, blackbox, file SD, relabeling — every device monitored.
DAY 3	**Alerting & on-call discipline** Alert rules, Alertmanager, channels, silences, recording rules.
DAY 4	**Dashboards & client deliverable** Design principles, templating, build the final client dashboard.

CHEAT SHEET

Vocabulary you will hear all week

Target	An endpoint Prometheus scrapes (a host, a router via exporter, a service).
Exporter	A small program that translates something else into Prometheus format.
Scrape	One fetch of /metrics from one target. Default every 15s.
Time series	Metric name + label set + stream of timestamped values.
Label	A key=value pair attached to a metric — a dimension you can group by.
PromQL	Prometheus Query Language. What you write in dashboards and alerts.
Instance	host:port of a single target. A label that's always present.
Job	A logical group of similar targets, set in prometheus.yml.

READY FOR THE PIPELINE

Three ideas to carry forward

Pull beats push for stable infrastructure · Labels make metrics dimensional · Prometheus is one piece of a larger stack