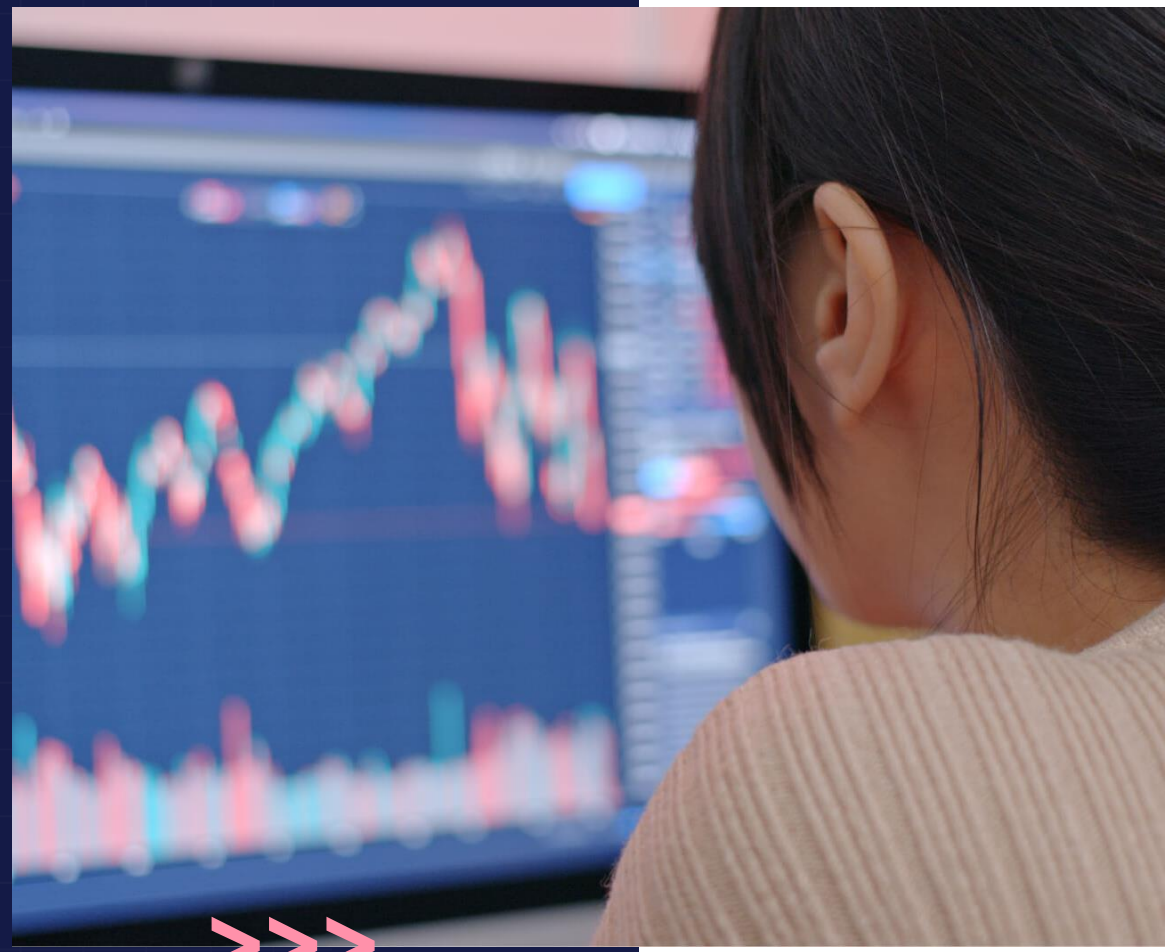
{mthree}

Introduction to Prometheus and Grafana

Site Reliability Engineering



Objectives

In this module, we will look at a monitoring system called Prometheus, the dashboard called Grafana, and how they interact.

Learning Objectives

- Explain what Prometheus is and does
- Examine how Prometheus gets data
- Configure Prometheus
- Query Prometheus with PromQL
- Using Grafana

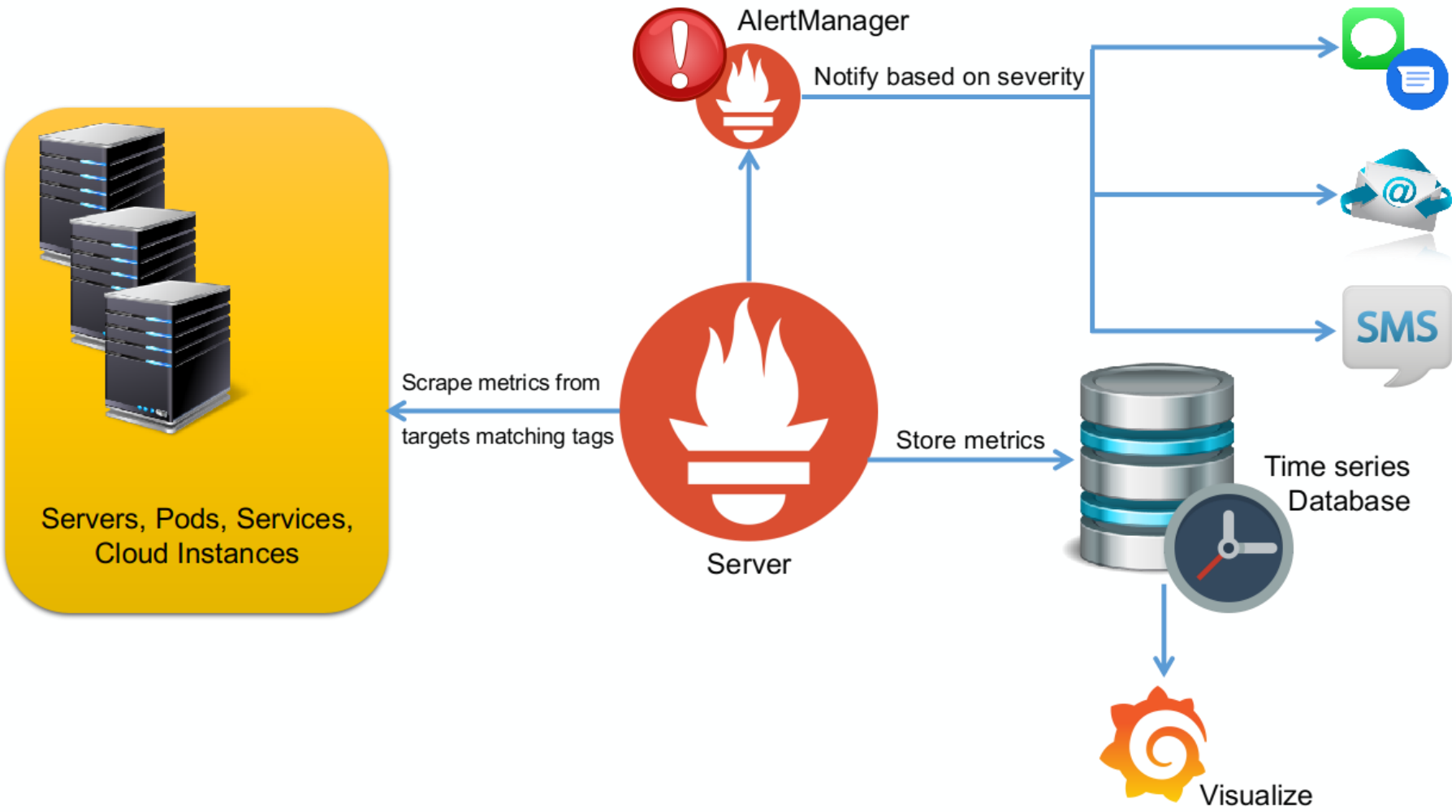
{mthree}

What is Prometheus?

- Monitoring tool
- From SoundCloud
  - >>> Created to monitor highly dynamic environments
    - ~ Very useful for monitoring Kubernetes, Docker swarm
  - >>> Can be used in traditional non-container environments
- Has become the go-to monitoring tool in the container and microservice world
  - >>> Modern DevOps complex needs more and more automation
  - >>> Challenging and complex infrastructures must be managed

{mthree}

How Prometheus Works



{mthree}

How Does Prometheus Work?

- Prometheus server
  - >>> Time series database
    - ~ Stores metrics data for current CPU usage
  - >>> Data retrieval worker
    - ~ Responsible for pulling metrics
  - >>> Web server
    - ~ Accepts queries and displays data in UI or dashboard visualization such as Grafana
- Monitors targets
  - >>> Linux server, Windows server, web servers, application servers, databases
- Measures units such as
  - >>> CPU status, request counts and durations, exception counts, memory/disk usage – Metrics

{mthree}

Monitoring Applications

- Collects relevant metrics such as:
  - >>> How many requests
  - >>> How many exceptions
  - >>> How many server resources
- Client libraries expose endpoints via HTTP URL called /metrics
  - >>> Requires using the library within your code
  - >>> https://prometheus.io/docs/instrumenting/clientlibs/
- Metrics can be
  - >>> Counter
  - >>> Guage
  - >>> Histogram

{mthree}

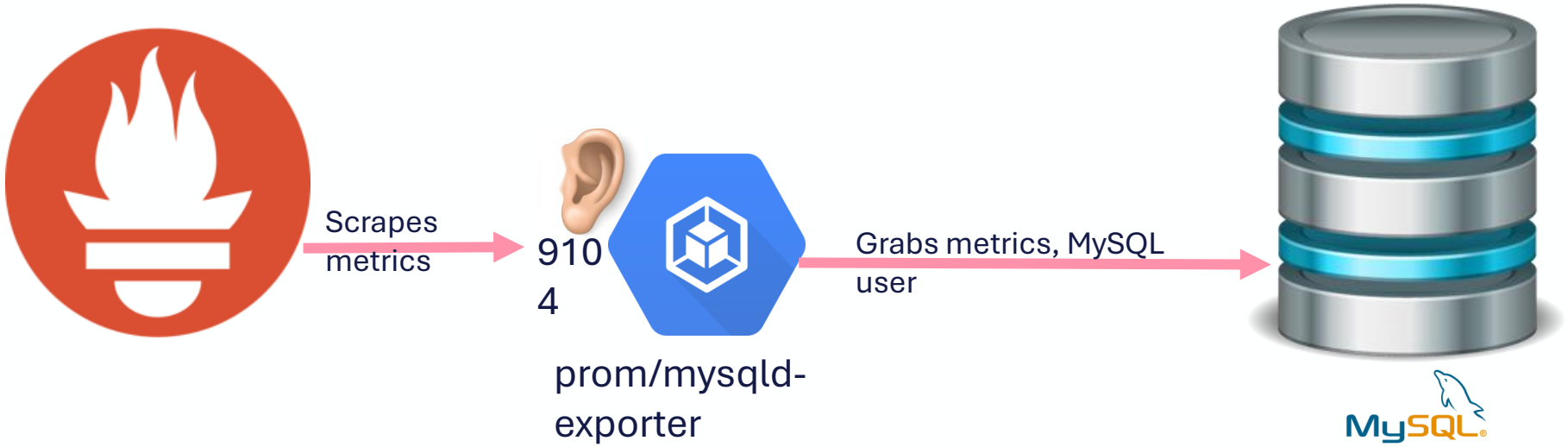
Our Lab Configuration

- Managed via Kubernetes manifests
  - >>> Within your **sre-course-infra** environment/namespace directory
- Requires
  - >>> podMonitor.yaml or serviceMonitor.yaml or probe.yaml (if exists)
    - ~ To tell Kubernetes about your monitoring ports to scrape
  - >>> Containers
    - ~ Either a separate monitoring container, e.g. mysqld-exporter
    - ~ Or a port in your application container
    - ~ Exposes Prometheus metrics

{mthree}

Example podMonitor

- Monitoring MySQL server remotely
- Reduce load on the database server
- Dedicated pod to monitor



{mthree}

Grafana

- Grafana allows you to:
  - >>> Query
  - >>> Visualize
  - >>> Alert on
  - >>> Understand metrics
- Central visualization of all your Prometheus monitoring
  - >>> Graphs, meters and more
- Dashboards to provide ease of understanding and speed
  - >>> Useful and colourful representation
  - >>> Thresholds and colours to determine issues or perfect conditions

{mthree}

The Loki Query Syntax

- Similar to Prometheus
- Key = value statement contained in { }
- Exact match of value use =
- RegEx match of value use =~
  - >>> Note if you wish to "contain" then use .* either side of your text
    - ~ e.g. {namespace=~".*example.*"}
- Multiple key searches are comma separated
  - >>> {namespace="sre-example-dev", app="orderbookapi"}
- Values must be in double quotes
- Looking for the word error in the log line
  - >>> {namespace="sre-example-dev", app="orderbookapi"} |= "error"

{mthree}

Summary Q & A

{mthree}

References and Further Learning

- How Prometheus monitoring works
  >>> https://www.youtube.com/watch?v=h4Sl21AKiDg
- How To Setup A Grafana Dashboard Step By Step
  >>> https://www.youtube.com/watch?v=4qpI4T6_bUw
- How to create Grafana Dashboards: The Easy way
  >>> https://www.youtube.com/watch?v=Mqt_bBsejKQ
- Grafanalib: Dashboards as Code
  >>> https://www.youtube.com/watch?v=OOyEGG98B7w
  >>> https://www.weave.works/blog/grafana-dashboards-as-code/

{mthree}

Useful Prometheus Selectors

probe_http_*

- HTTP request information for the application
- Provided you've added a probe to your namespace

default_jenkins_*

- default_jenkins_builds_duration_milliseconds_summary_sum
- default_jenkins_builds_health_score
- default_jenkins_builds_failed_build_count
- default_jenkins_builds_success_build_count
- default_jenkins_builds_last_build_result

- container_*
- kube_pod_*
- kube_deployment_*
- kube_endpoing_*
- kube_namepsace_*
- mysql_*
- nginx_ingress_controller_*
- probe_http_*
- probe_success

And also https://cXXXteam??dev.computerlab.online/metrics

{mthree}

Useful Loki LogQL

- {app=”kustomize-controller”}
  - >>> To see if your deployment is there, failing, or changed
  - >>> add |= “namespace”
    - ~ To view only your namespace
- {app=”ingress-nginx”} | json | host=”orderbookdev.computerlab.online”
  - >>> To view request/response details to a particular applications
    - ~ orderbookdev.computerlab.online is the application
- {app=”ingress-nginx”} | json | host=~”orderbookdev.*” | request_time > 0.005
  - >>> Checking if request time is > 5ms
- sum(count_over_time({container=”orderbookac”} |= ”Steve” |= ”buy”)[1h]))
  - >>> The number of buys by Steve over the last 1-hour period
- sum(rate({container=”orderbookac”} |= ”Steve” |= ”sell”)[1m]))
  - >>> The per second rate of all sell’s within the last minute
- {namespace=”orderbook-dev”}
  - >>> Get logs from containers running in your namespace

{mthree}