AMIS

Assuring business value

Monitoring

with Prometheus and Grafana

Lucas Jellema, CTO of AMIS
Fall 2018

Lucas Jellema
Architect / Developer

1994 started in IT at Oracle
2002 joined AMIS
Currently CTO & Solution Architect

Overview

- Why monitoring?
- How monitoring?
- Metrics
  - gathering | processing | analyzing | presenting | alerting
- Prometheus
  - History
  - Metrics
  - Exporters
  - Instrumentation of Applications
- Grafana
- Dinner
- Handson Workshop

Monitoring

- Observe [non-functional] behavior of business functions/applications in pnear] real-time
  - Availability and health
  - Performance
  - Access – as it should be for whom it should be

- Teams up with:
  - Profiling & Debugging – maximum context for activity spikes
  - Tracing – track paths through application [and platform & infra] stack
  - Logging – per application or platform component output for off-line processing
    - although log agents such as Elastic Stack Beats and Elasticsearch

What is required for monitoring?

- Gather metrics
  - And prepare | wrangle metrics (tag, filter, enrich, aggregate, …)

- To raise alert
  - To human (via ticket/SMS/…)
  - To automated handler/agent
- To support issue resolution (data for root cause analysis)
- To analyze trends + effects/impact of change

Type of metrics

- Primary metrics
  - Tied to SLA indicators
  - Relevant – representative of Key Performance Indicators
    - End user experience
    - Business activity throughput
    - Cost efficiency

- Secondary metrics
  - Predictors for primary metrics

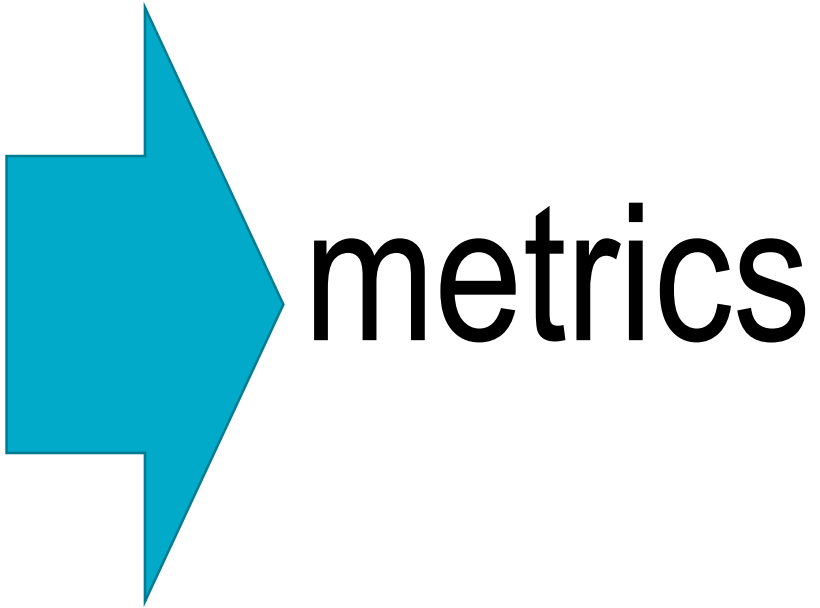
- Technical metrics
  - Hygiene factors – not directly tied to specific business indicators
  - Temperature, Remaining Free Storage, Network Load,

Dashboard

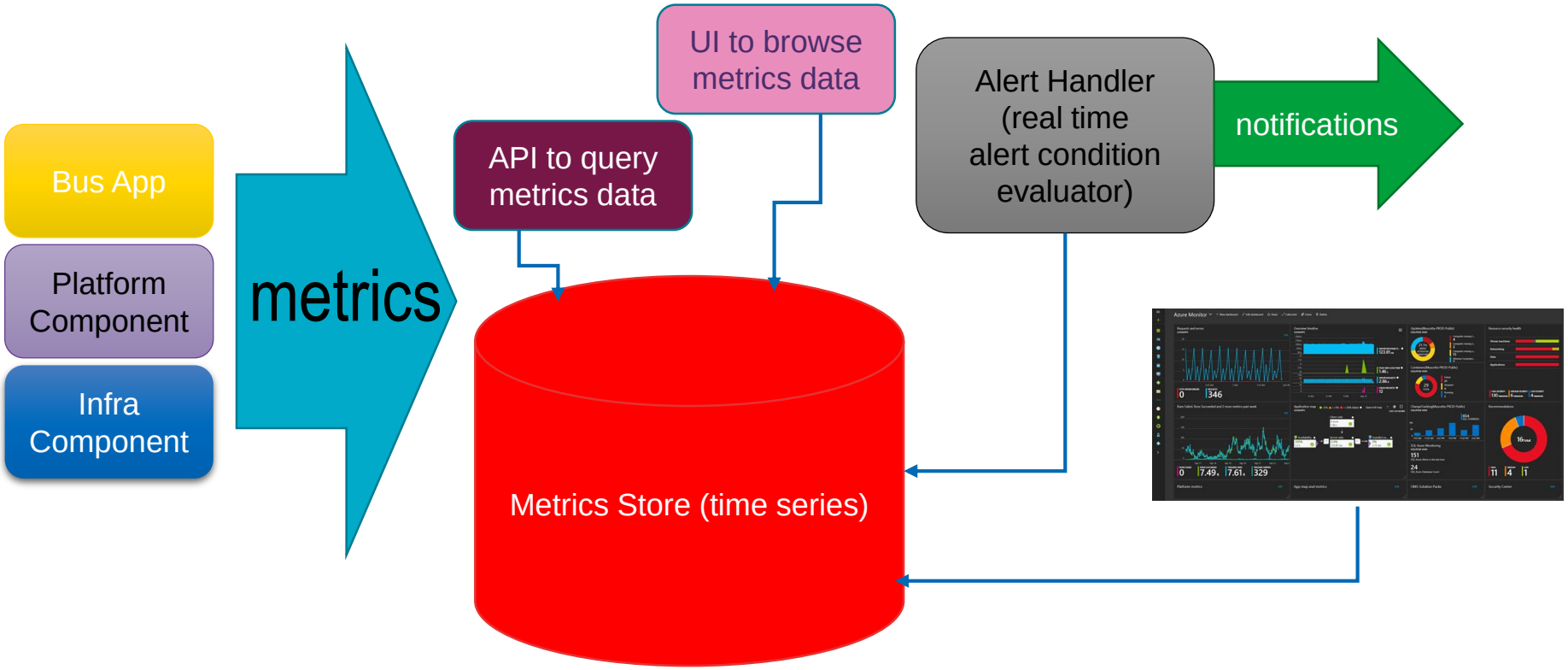
- Provide starting point to respond to an alert – start investigation (drill down, query)
- Demonstration purposes
  - Always looks nice – a dashboard

Metrics are collected across the stack

- Business Applications
  - SaaS, Standard Applications
  - Custom | Tailor made applications
- Platform
  - Web Server, Application Server
  - Database
  - LDAP
  - JVM/Node/.NET/… runtime
- Infra
  - Container, Container Platform (e.g. Kubernetes)
  - Operating System
  - Cache
  - Proxy, Load Balancer
  - Network
  - Storage, File System



Monitoring


Bus App
Platform Component
Infra Component
metrics
API to query metrics data
UI to browse metrics data
Alert Handler (real time alert condition evaluator)
notifications
Metrics Store (time series)

History and Status of Prometheus

- Written in Go Lang – all open source, available on GitHub
- Part of CNCF
- https://prometheus.io/
- Since 2012
- Defacto standard for gathering metrics (?)
- *treating time-series data as a data source for generating alerts is now accessible to everyone*

Dimensional data

Prometheus implements a highly dimensional data model. Time series are identified by a metric name and a set of key-value pairs.

Powerful queries

A flexible query language allows slicing and dicing of collected time series data in order to generate ad-hoc graphs, tables, and alerts.

Great visualization

Prometheus has multiple modes for visualizing data: a built-in expression browser, Grafana integration, and a console template language.

Efficient storage

Prometheus stores time series in memory and on local disk in an efficient custom format. Scaling is achieved by functional sharding and federation.

Simple operation

Each server is independent for reliability, relying only on local storage. Written in Go, all binaries are statically linked and easy to deploy.

Precise alerting

Alerts are defined based on Prometheus's flexible query language and maintain dimensional information. An alertmanager handles notifications and silencing.

Many client libraries

Client libraries allow easy instrumentation of services. Over ten languages are supported already and custom libraries are easy to implement.

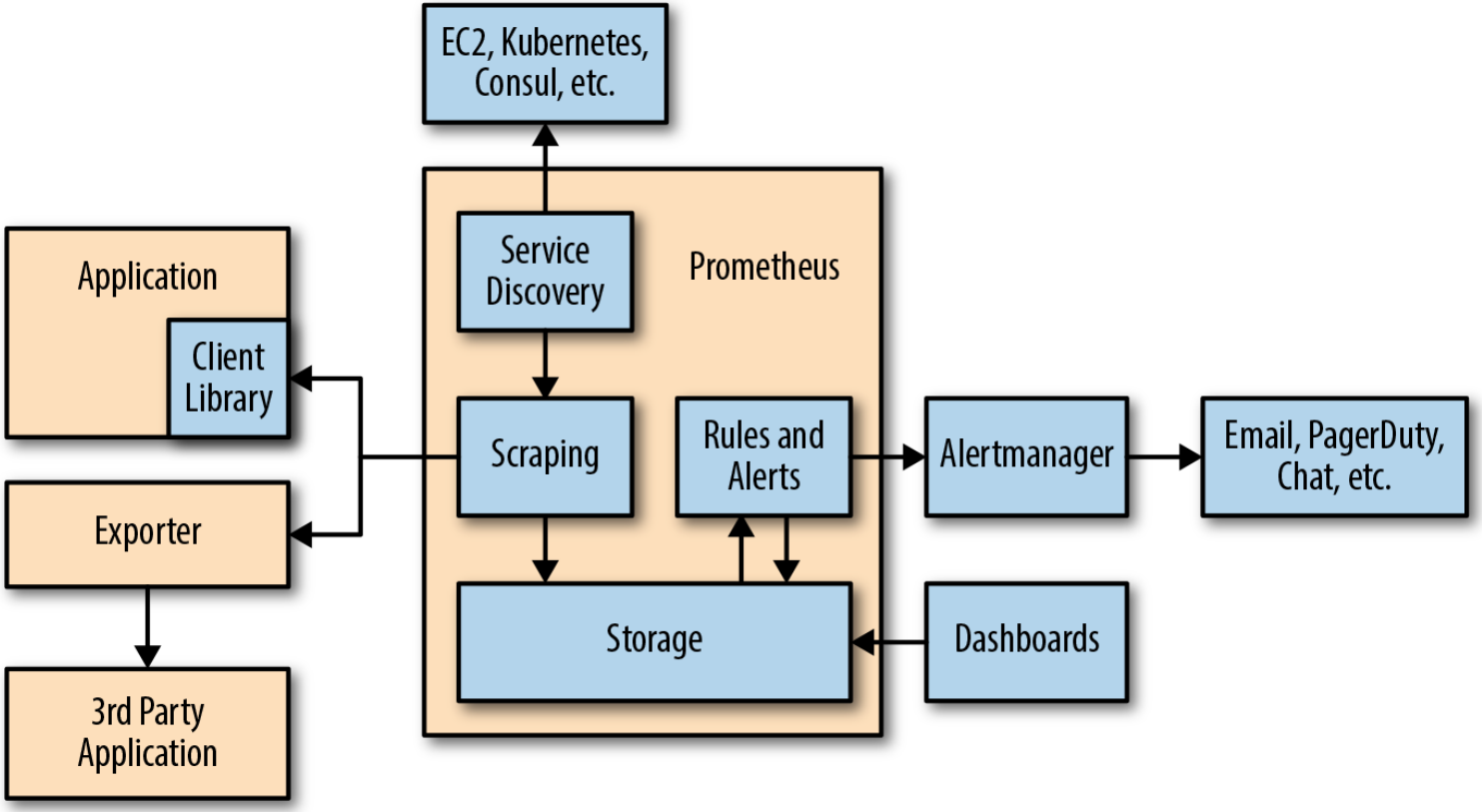
Many integrations

Existing exporters allow bridging of third-party data into Prometheus. Examples: system statistics, as well as Docker, HAProxy, StatsD, and JMX metrics.

Prometheus

- Gather metrics into database
  - Scheduled pull |harvest| scrape actions – HTTP/TCP requests
  - Accessing Exporters and built in (scrape) endpoints
- Make metrics availabe to consuming systems and humans
  - Such as Grafana (for dashboarding)
  - REST APIs
  - Prometheus UI – Graphs, Console, PromQL
- Analyze metrics according to [alert] rules
  - Determine if alerts are “firing”
- Act on firing alerts
  - Send notifications
- Supports federation – global view over local environments and recovery of local environment

Prometheus Architecture

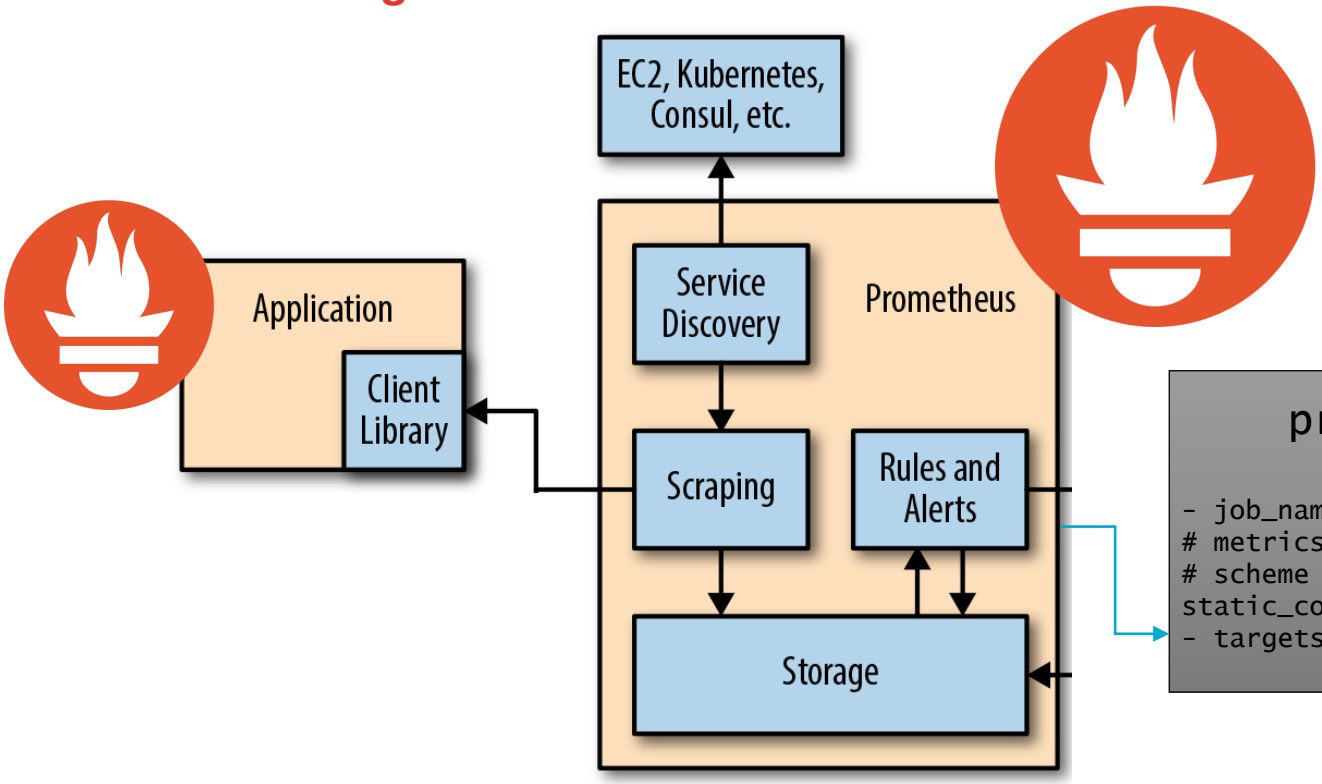


Prometheus Metrics

- All are numeric
- Uniquely identified by name and set of labels
- All can be labeled (associated with dimensions)
  - Aggregation is done *grouped by dimension*
  - (labels should have limited number of values)
  - Filtering and Drill down is also done using labels

- Examples of labels:
  - Data Center, Region
  - Environment (Prod, Acc, Test)
  - Service, Application, Module
  - URL Path, request type, status, error code
  - Not for dynamic aspects such as date, time, user id, user IP

Demo – Getting started with Prometheus



prometheus.yml

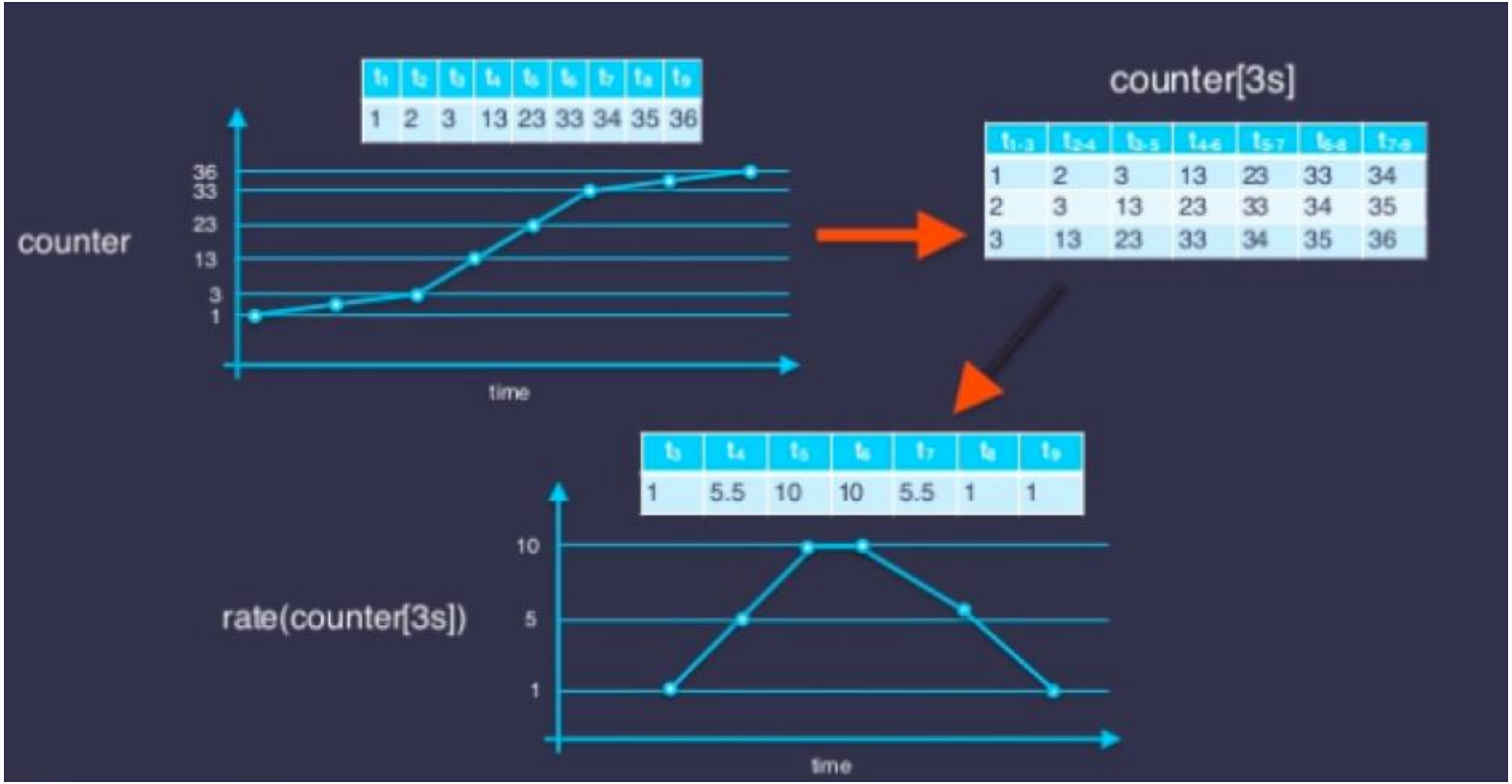
```
- job_name: 'prometheus'
# metrics_path defaults to /metrics
# scheme defaults to 'http'.
static_configs:
- targets: ['localhost:9090']
```

Prometheus – Types of Metrics

- Basic counters:
  - Gauge – current value (%disk free, response time, state of database)
  - Counter - #occurrences – and the derived rate - #occurrences/time unit - (number of requests, number of bytes transferred);
    - counters can be reset but not decremented
- Sampling Counters:
  - Summary – reports quantiles as well as total sum and total count (over sliding window)
  - Histogram – maps observations to buckets and reports number of instances per bucket (over sliding window)

Basic Counters	Sampling Counters
counter	histogram
gauge	summary

Counter – per time slice and as a rate



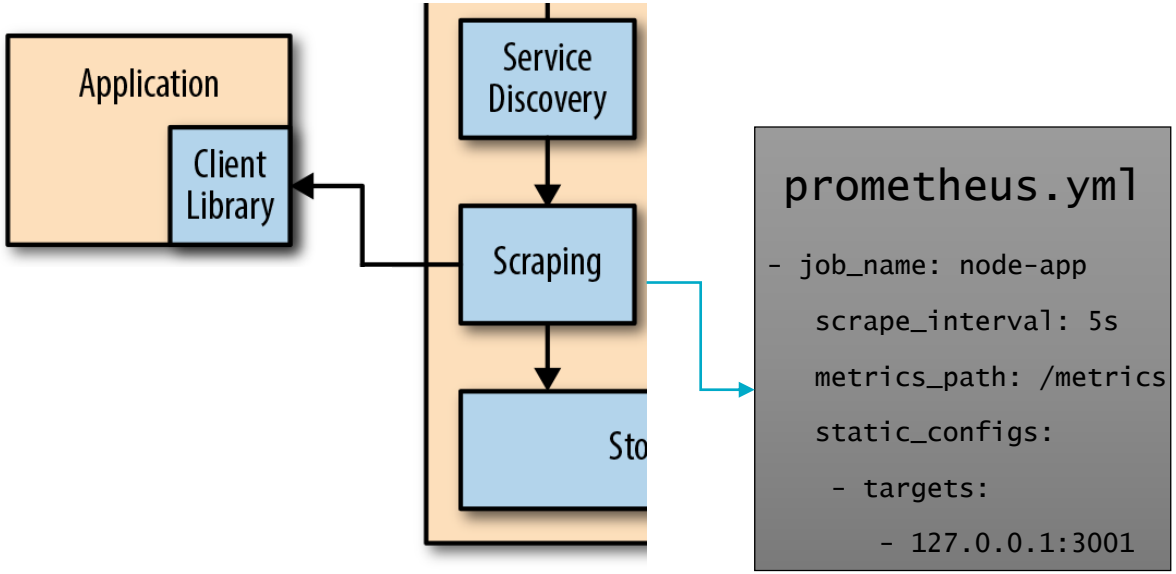
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
1	2	3	13	23	33	34	35	36

t_{1-3}	t_{2-4}	t_{3-5}	t_{4-6}	t_{5-7}	t_{6-8}	t_{7-9}
1	2	3	13	23	33	34
2	3	13	23	33	34	35
3	13	23	33	34	35	36

t_3	t_4	t_5	t_6	t_7	t_8	t_9
1	5.5	10	10	5.5	1	1

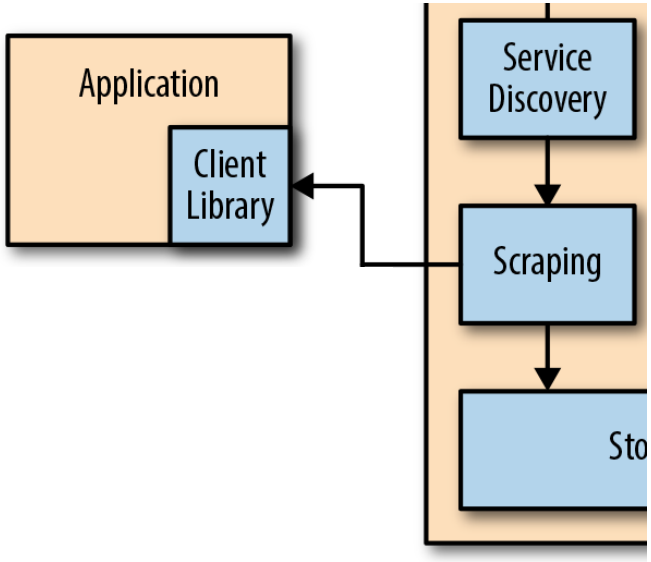
Exposing Metrics for Prometheus to Scrape

- Have the application or environment listen for HTTP requests at a specific endpoint (for example: *host:port/metrics*)
- Return Metrics in the proper format to GET requests to this endpoint
- Use a Client Library to easily compose the proper metrics response messages
- Configure the endpoint on the Prometheus server in the *prometheus.yml* file



Client Libraries for Exposing Metrics for Prometheus to Scrape from custom applications

- Go
- Java or Scala
- Python
- Ruby
- Bash
- C++
- Common Lisp
- Elixir
- Erlang
- Haskell
- Lua for Nginx
- Lua for Tarantool
- .NET / C#
- Node.js
- PHP
- Rust

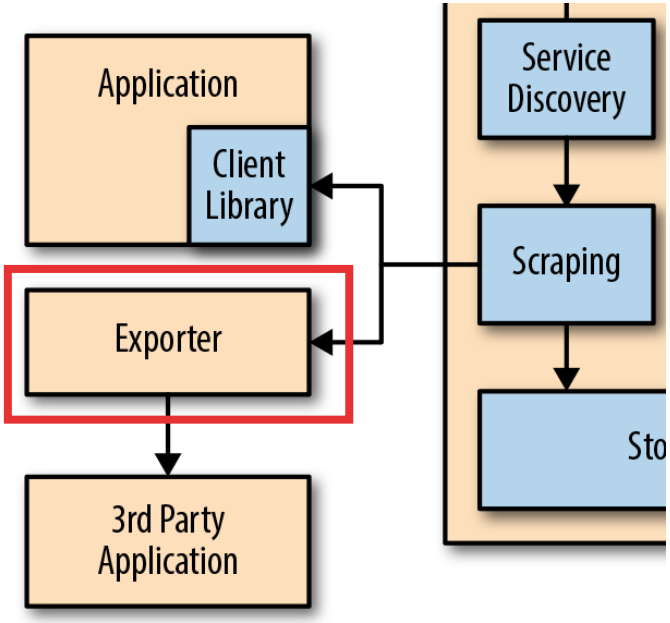


Example of Exposing Metrics from a Node application

```
const express = require('express')
const Prometheus = require('prom-client')
const app = express()
const port = process.env.PORT || 3001
const metricsInterval = Prometheus.collectDefaultMetrics()
const checkoutsTotal = new Prometheus.Counter({
  name: 'checkouts_total',
  help: 'Total number of checkouts',
  labelNames: ['payment_method']
})
const httpRequestDurationMicroseconds = new Prometheus.Histogram({
  name: 'http_request_duration_ms',
  help: 'Duration of HTTP requests in ms',
  labelNames: ['method', 'route', 'code'],
  buckets: [0.10, 5, 15, 50, 100, 200, 300, 400, 500]  // buckets for response time from 0.1ms to 500ms
})
// Runs before each requests
app.use((req, res, next) => {
  res.locals.startEpoch = Date.now()
  next()
})
app.get('/metrics', (req, res) => {
  res.set('Content-Type', Prometheus.register.contentType)
  res.end(Prometheus.register.metrics())
})
```

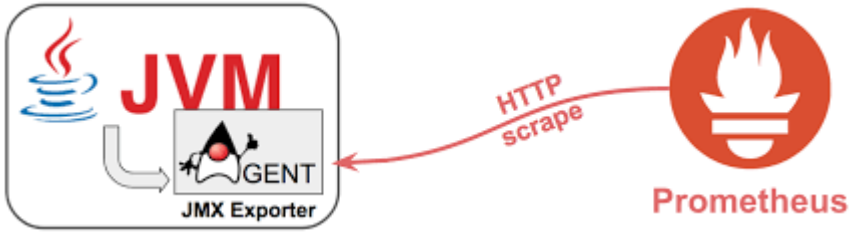
Prometheus Exporters

- Specialized adapters to expose metrics for specific technology components
  - Installed and configured for a specific component
  - Scraped by Prometheus based on config file prometheus.yml that references the endpoint exposed by the exporter



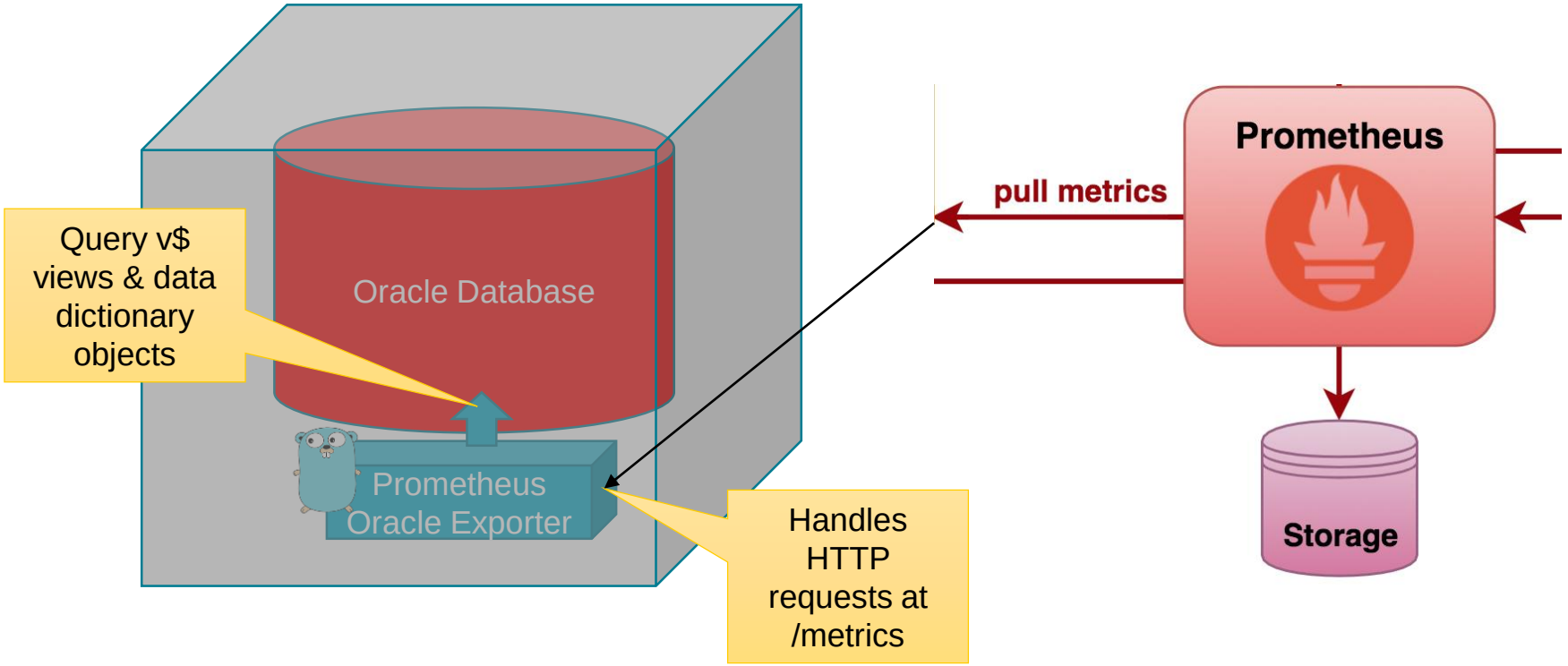
Prometheus Exporters

- Linux & Windows
- Databases
- Messaging Systems
- Storage
- HTTP
- APIs
- Logging
- Monitoring Systems
- Application Servers & Container Platforms
- Blackbox Exporter



- A growing number of components has built-in Promethes metrics publication

For example: Oracle Database Exporter



Prometheus Oracle Exporter

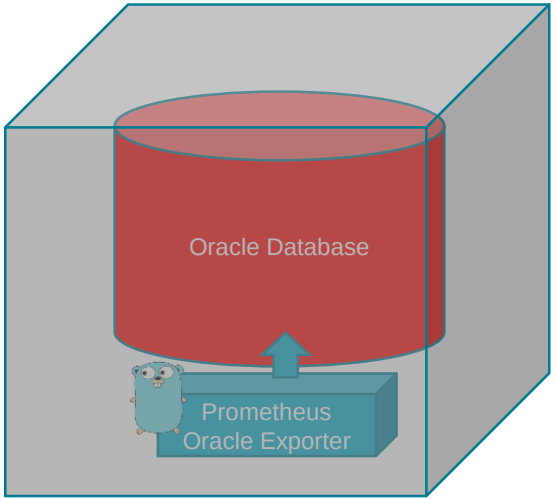
A Prometheus exporter for Oracle.

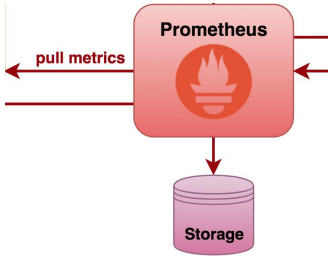
The following metrics are exposed currently. Support for RAC (databasename and instancename added via lables)

- oracledb_exporter_last_scrape_duration_seconds
- oracledb_exporter_last_scrape_error
- oracledb_exporter_scrapes_total
- oracledb_uptime (days)
- oracledb_session (view v$session system/user active/passive)
- oracledb_sysmetric (view v$sysmetric (Physical Read Total IO Requests Per Sec / Physical Write Total IO Requests Per Sec Physical Read Total Bytes Per Sec / Physical Write Total Bytes Per Sec))
- oracledb_sysstat (view v$sysstat (parse count (total) / execute count / user commits / user rollbacks))
- oracledb_waitclass (view v$waitclass)
- oracledb_tablespace (tablespace total/free)
- oracledb_asmspace (Space in ASM (vasm_disk/vasm_diskgroup))
- oracledb_interconnect (view v$sysstat (gc cr blocks served / gc cr blocks flushed / gc cr blocks received))
- oracledb_recovery (percentage usage in FRA from V$RECOVERY_FILE_DEST)
- oracledb_redo (Redo log switches over last 5 min from v$log_history)
- oracledb_cachehitratio (Cache hit ratios (v$sysmetric)
- oracledb_up (Whether the Oracle server is up)
- oracledb_error (Errors parsed from the alert.log)
- oracledb_error_unix_seconds (Last modified Date of alert.log in Unixtime)
- oracledb_services (Active Oracle Services (v$active_services))
- oracledb_parameter (Configuration Parameters (v$parameter))
- oracledb_query (Self defined Queries in Configuration File)

*TOOK VERY LONG, BE CAREFUL (Put the Metrics below in a separate Scrape-Config):

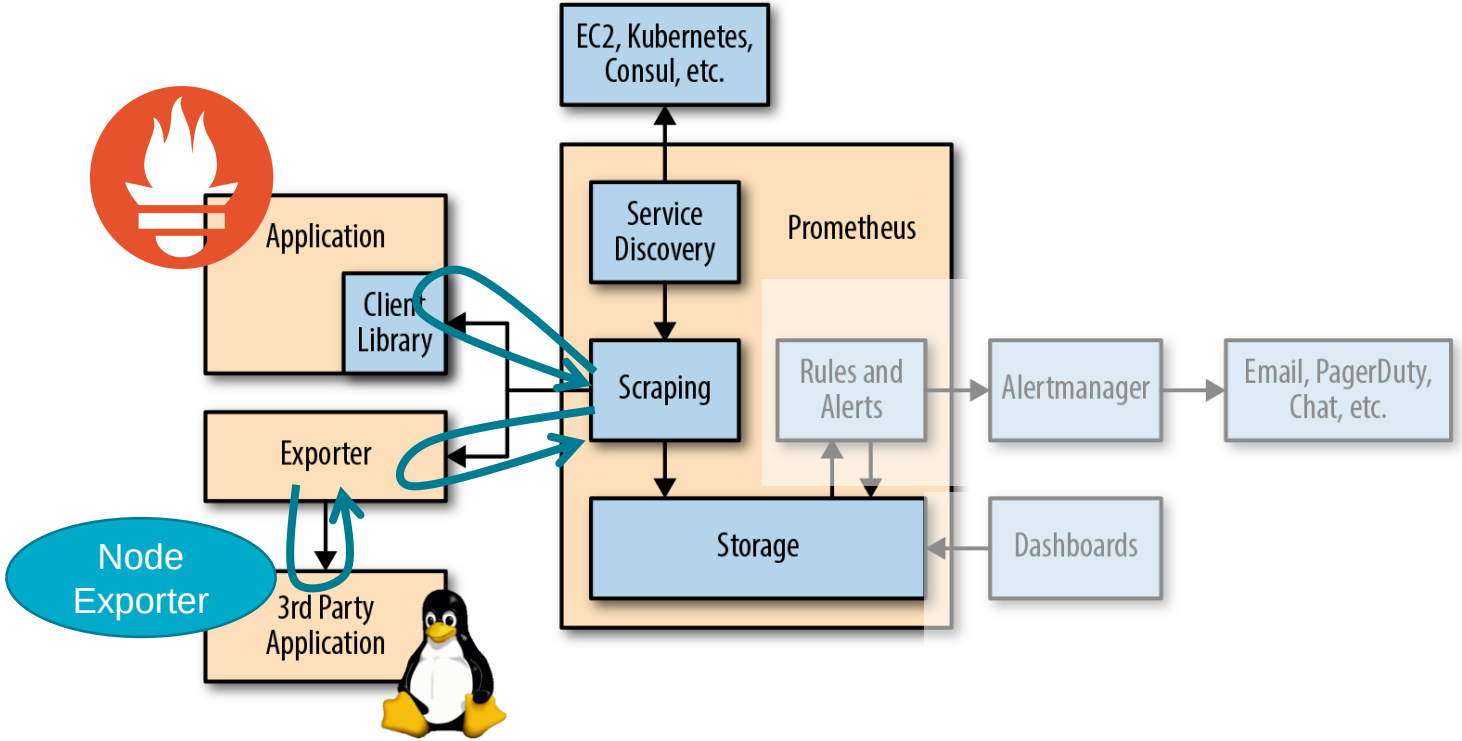
- oracledb_tablerows (Number of Rows in Tables)
- oracledb_tablebytes (Bytes used by Table)
- oracledb_indexbytes (Bytes used by Indexes of associated Table)
- oracledb_lobbytes (Bytes used by Lobs of associated Table)





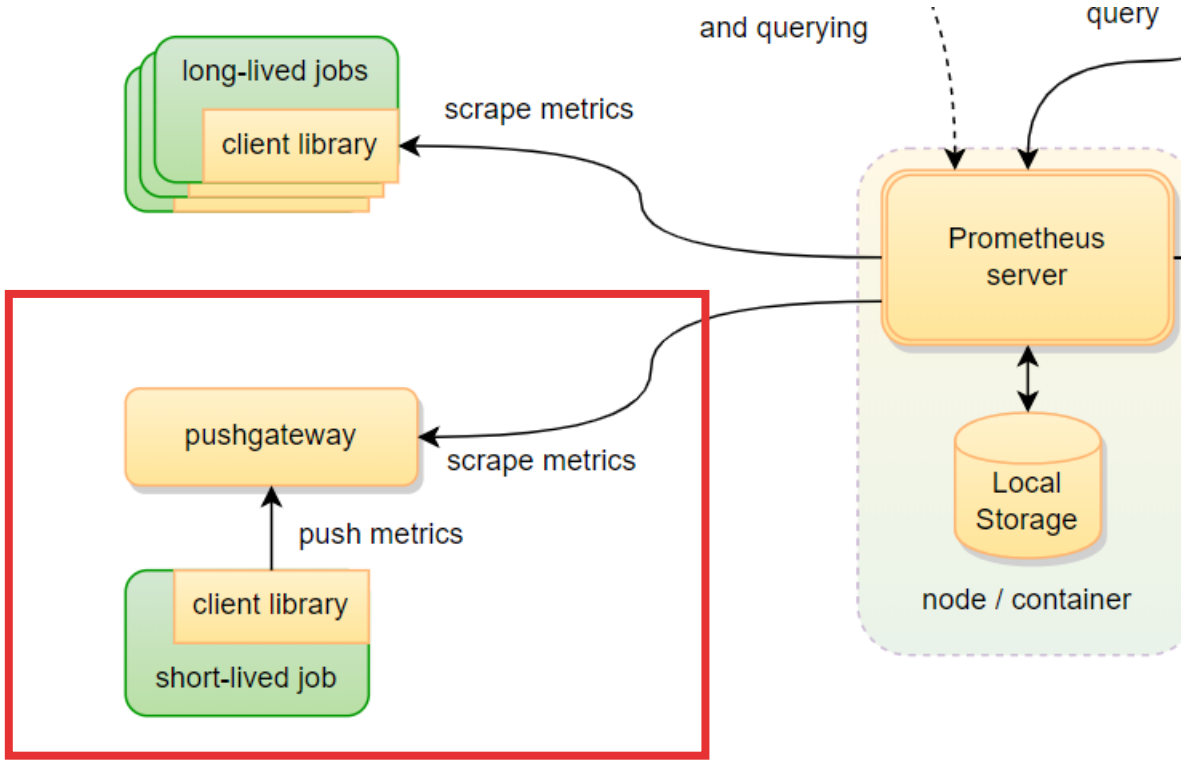
Demo – (Linux) Node Exporter

- Run node exporter as background job on Linux node

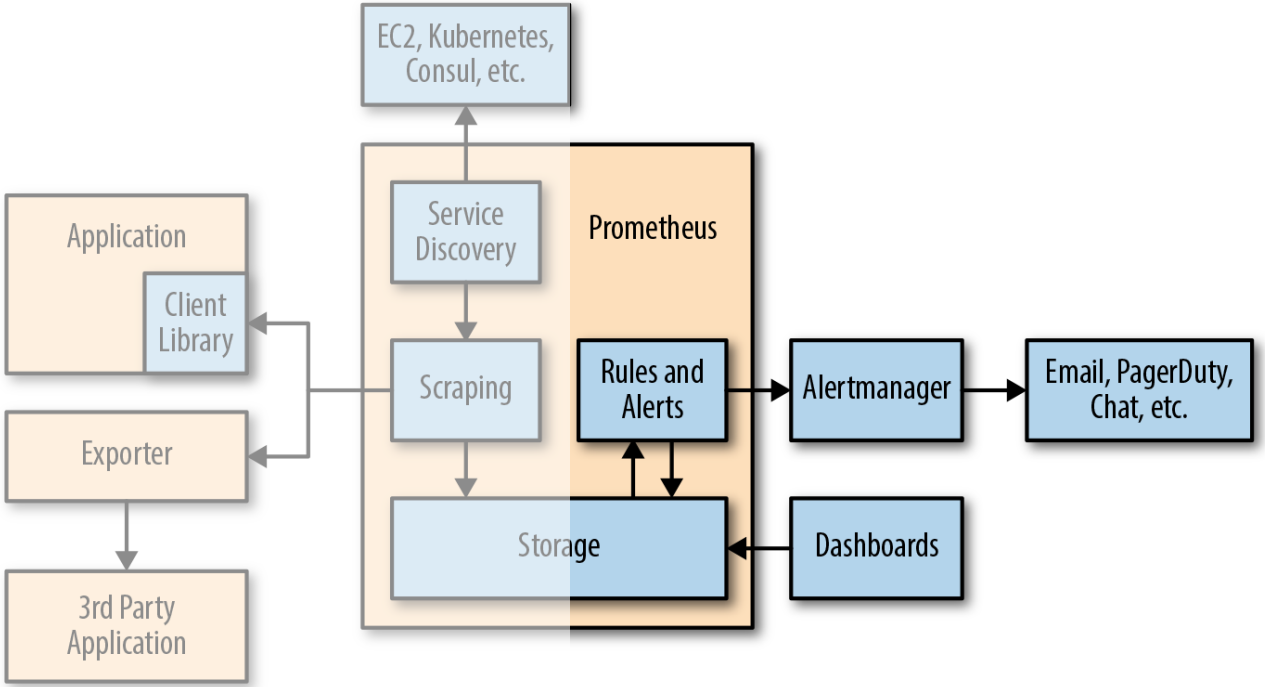


Pushgateway for Short-Lived Jobs

- Jobs that may be gone before their metrics are scraped



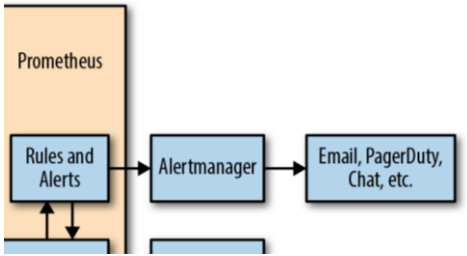
Prometheus Alerting



Prometheus Alerting

```
groups:
 - name: target_rules
   rules:
   - alert: InstanceDown
     expr: up == 0
     for: 1m
```

- Rules specify alert conditions
  - Expressed in Prometheus metrics
- Prometheus evaluates these rules –
  - As time progresses
  - As metrics get updated
- An alert is be triggered
  - When the *expr* evaluates to true
  - For at least as long as is specified in the *for* condition
- A triggering alert
  - Can be found in the Prometheus Web UI
  - Is notified to the Alert Manager
- The Alert Manager is configured for action
  - For example: send notification via specific communication channels



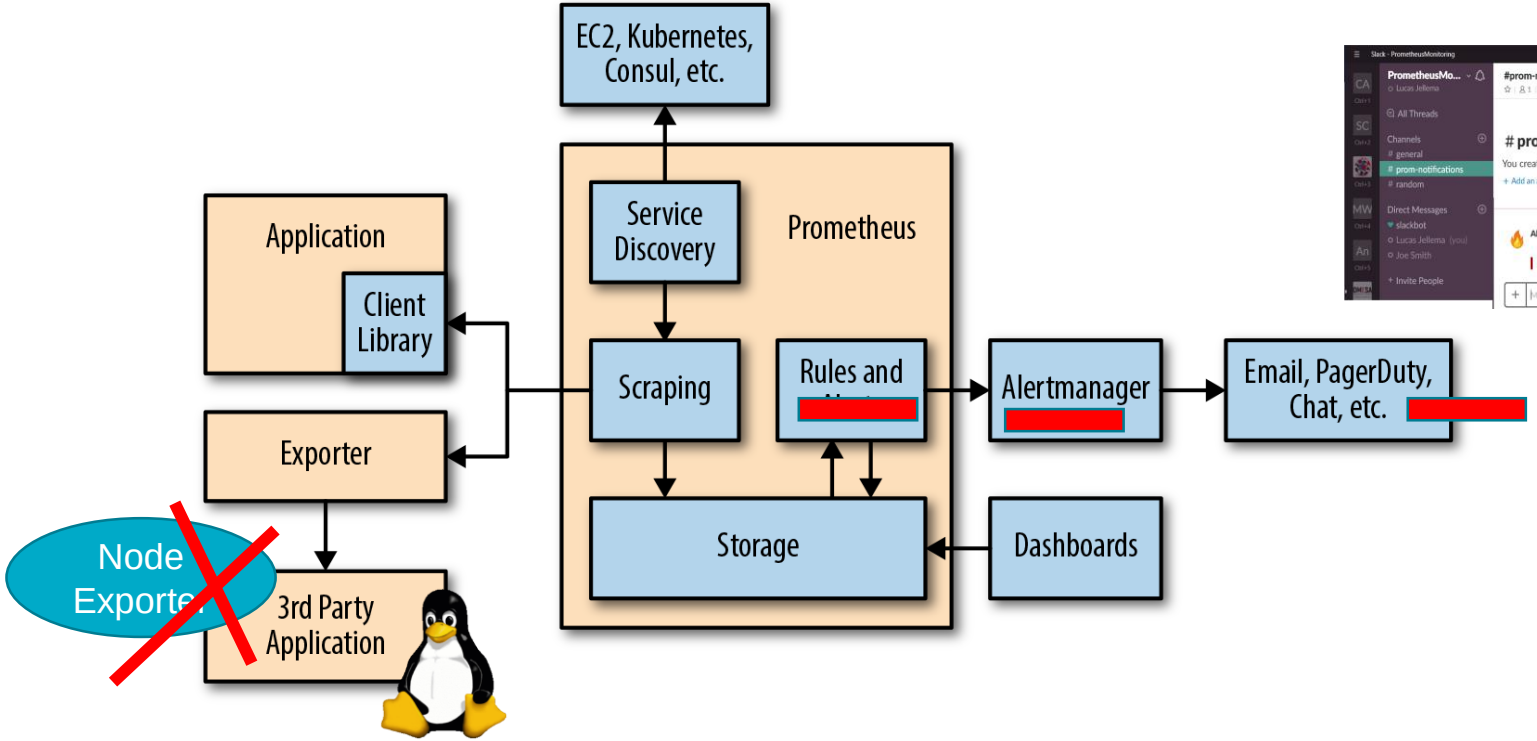
Alertmanager Configuration

```
route:
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 1h
  receiver: 'slack_alerts'
  routes:
  - match:
      alertname: CheckOutsOdd
    group_by: [ service ]
    receiver: businessticket
receivers:
- name: slack_alerts
  slack_configs:
  - api_url: https://hooks.slack.com/services/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
    channel: '#prom-notifications'
- name: businessticket
  slack_configs:
  - api_url: https://hooks.slack.com/services/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
    channel: '#prom-notifications'
    title: 'Alerts in Service {{ .GroupLabels.service }}'
    text: >
      {{ .Alerts | len }} alerts:
```

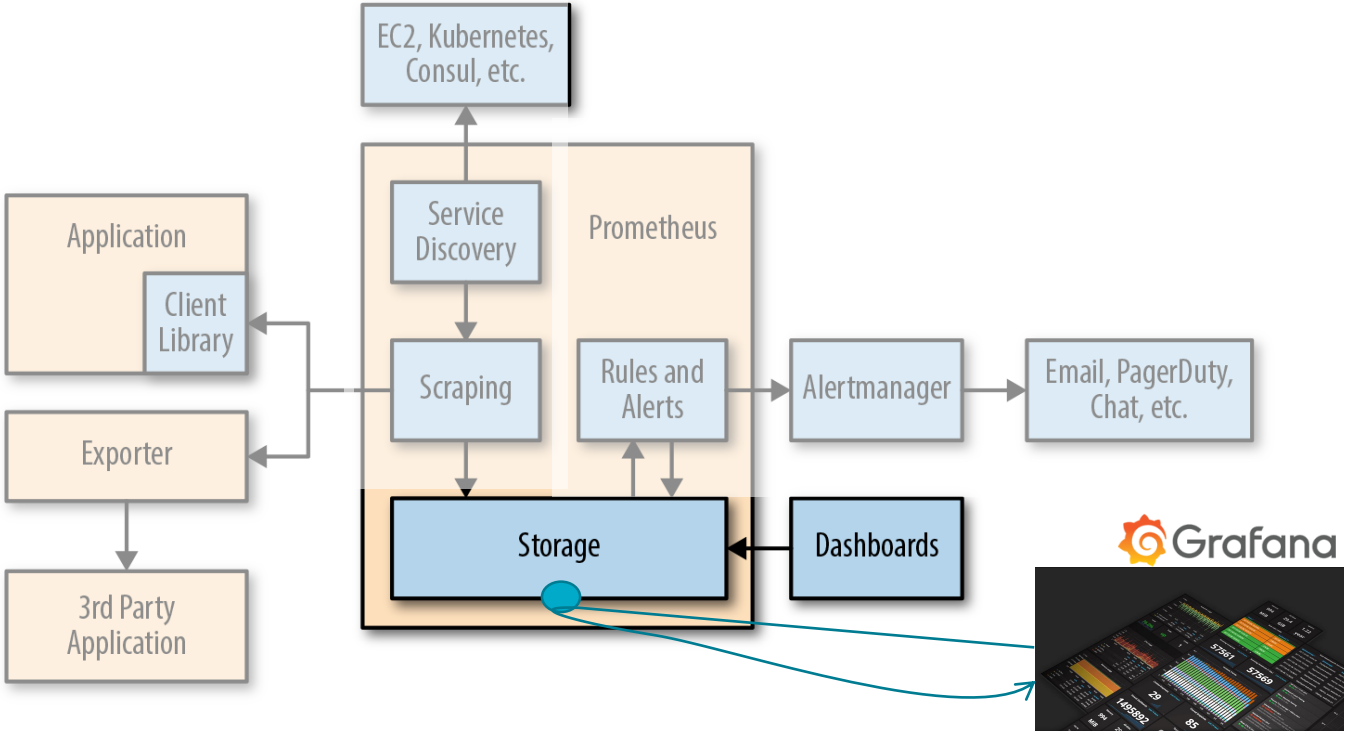
The route to apply depends on matching conditions

Each receiver can use a different communication channel and notification layout

Prometheus Alerting Demo



Dashboarding



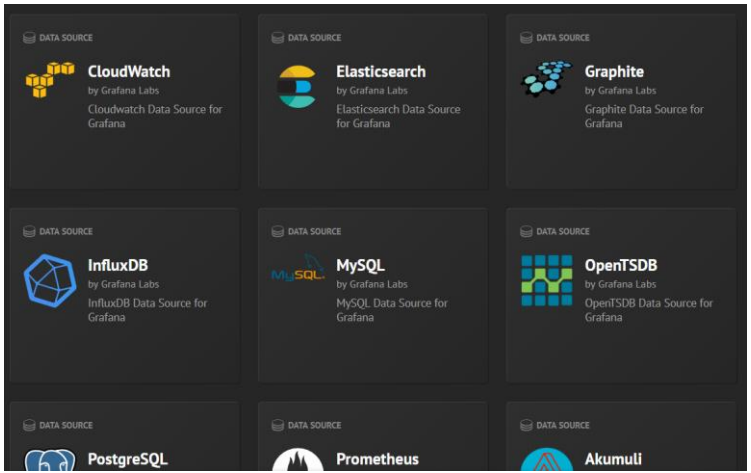
Grafana - dashboarding

- Grafana is a generic open source dashboard product
  - Supporting many types of data sources, of which Prometheus is but one
- Grafana queries data sources (such as Prometheus) periodically
  - Does not store any data
  - Refreshes visualizations
  - Evaluates alert conditions and triggers alerts/sends notifications
- Extensive library of pre-built dashboards available
  - Also plugins
- Supports user authentication and authorization and multi-tenancy
- Online Playground

Grafana – Data Sources

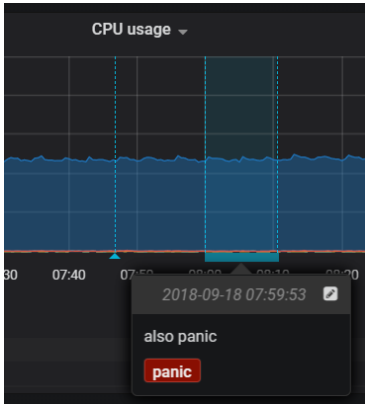
- Grafana can extract data from many types of data sources
  - Prometheus is but one
- Grafana can use multiple data source to provide data for a panel or chart

- Graphite
- Elasticsearch
- CloudWatch
- InfluxDB
- OpenTSDB
- Prometheus
- MySQL
- Postgres
- Microsoft SQL Server (MSSQL)

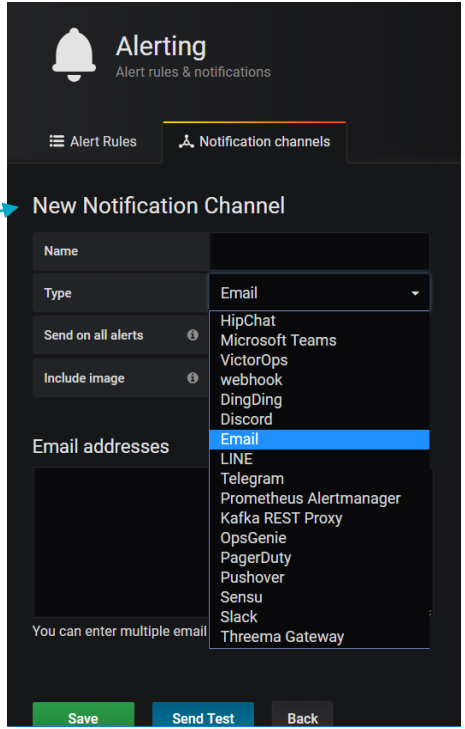


Grafana Visualizations

Annotations



Grafana Alerts and Notifications



Materials: **http://bit.ly/prometheus-amis**

Workshop

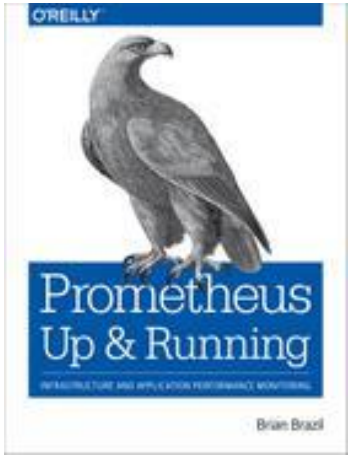
- Prometheus
  - First steps
  - Exporters for Linux, MySQL, Docker, Blackbox
  - Custom Application instrumentation (Node application)
  - Alerting
  - Alert Manager and Notifications (to Slack)
- Grafana
  - First steps
  - Preconfigured Prometheus dashboard
  - Custom Dashboard based on Prometheus metrics
  - Alerts and notifications

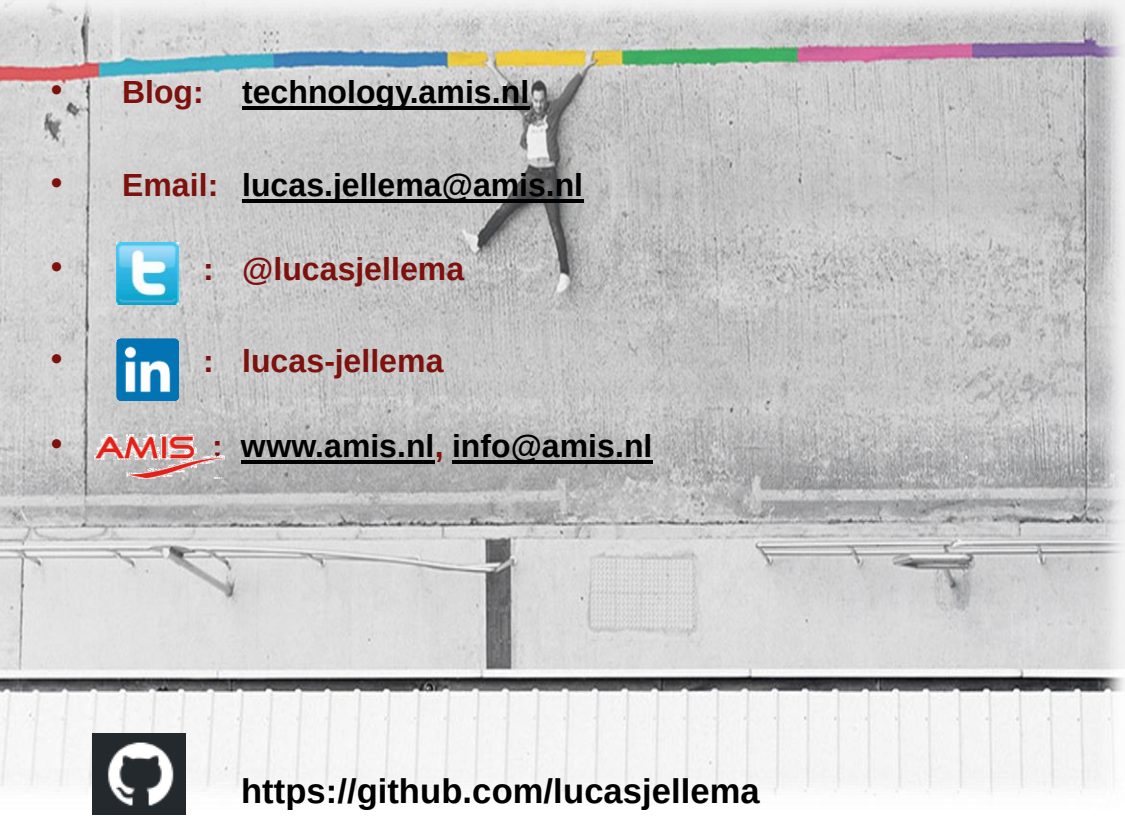
- Use prebuilt VM image or build VM from scratch
  - Need VirtualBox in both cases [and Vagrant for *build from scratch*]

Getting Started/Useful Resources

- KataCoda:
  - https://www.katacoda.com/courses/prometheus
    https://www.katacoda.com/courses/prometheus/creating-dashboards-with-grafana

- Book: Prometheus Up & Running (O'Reilly, 2018)





- Blog: technology.amis.nl
- Email: lucas.jellema@amis.nl
- : @lucasjellema
- : lucas-jellema
- AMIS : www.amis.nl, info@amis.nl

https://github.com/lucasjellema

धन्यवाद

dhanyavaad

Thank you

Dank je wel