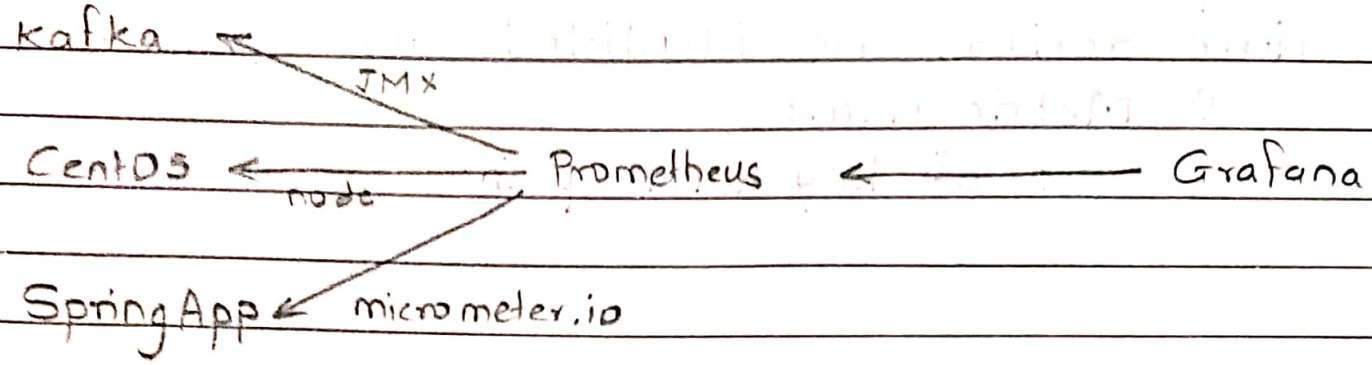
Prometheus - time Series database

Date :.....
MON TUE WED THU FRI SA SUN



* prometheus scrape the endpoints
* prometheus query language

Prometheus

- open source monitoring solution
- time series database
- built by soundcloud - 2012-13
- Now, its standalone open source project (from 2013)
- joined CNCF in 2016
- ideal for monitoring on premise as well as cloud workloads.

MON TUE WED THU FRI SAT SUN

* In prometheus, we talk about Dimensional Data. time series are identified by
  1) Metric name
  2) set of key value pairs (label)

Metric Name	Label	Sample
temperature	location = outside	90

* prometheus includes flexible Query language

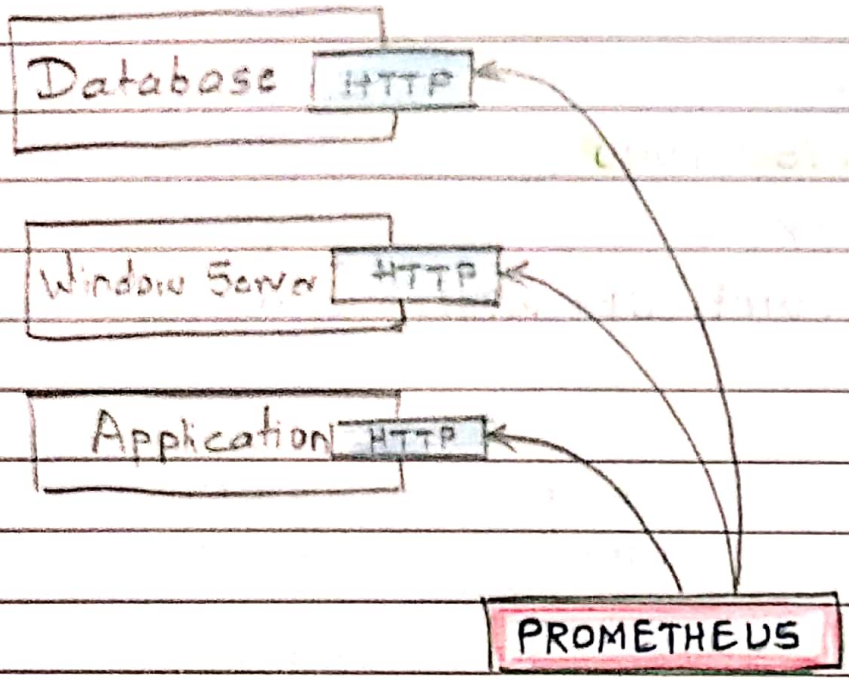
* Visualizations can be shown using a built-in expression browser or with integrations like Grafana.

* It stores metrics in memory or local disk in an own custom, efficient forma

* It is written in Go

* Many client libraries and integrations are available

How does Prometheus Works ?



- prometheus collects metrics from monitored targets by scraping metrics HTTP endpoints
  - Rather than using custom scripts that check on perticular services and systems, the monitoring data itself is used

  - Scaping endpoints is much more efficient than other mechanisms, like 3rd party agents

- A single prometheus server is able to ingest upto one million samples per second as several million time series

* To check prometheus →

ps aux | grep prometheus

* It is running on

165.85.18.78 : 9090

ops server

you will see web UI which contains

Graph and Console

element (label) Value (Sample)

* Grafana runs on

165.85.18.78 : 3000

admin

Basic Concepts

Date :.....

MON TUE WED THU FRI SA SUN

① All data is stored as time series
- every time series is identified by the **metric name** and set of key-value pairs, called **labels.**

example,

go_memstat_alloc_bytes {instance = "localhost:9090", job = "node_expoter"}

metric name

labels (key-value pairs)

② The time series data also consist of actual data called "samples"
- it can be float64 value
- or millisecond precision timestamp

③ Notation of time-series

<metric name> {<labelname> = <label value>, ...}

Prometheus Configuration

Date : .....

MON TUE WED THU FRI SAT SUN

- The configuration is stored in prometheus configuration file, in yaml format.
  - The configuration file can be changed and applied, without having to restart prometheus
  - A reload can be done using
    `kill -SIGHUP <pid>`

- You can also pass parameters at startup time to ./prometheus
  - Those parameters cannot be changed without restarting prometheus

- The configuration file is passed using the flag `--config.file`

- The default configuration contains ~~three~~ 4 main parts -

① global → global config
- ① scrape_interval
- ② evaluation_interval
- ③ scrape_timeout

② alerting → Alertmanager config

```
alerting:
  alertmanagers:
    static_config:
      targets:
        - alertmanager:9093
```

③ rule_files → load rules once & periodically evaluate them according to global evaluation time

④ Scrape-configs:

```
- job_name:
  static_configs:
    - targets
```

eg. To scrape metrics from prometheus itself,

```
scrape_config:
    - job_name: 'prometheus'
    # metrics_path (default '/metrics')
    # scheme (default 'http')
      static_configs:
        - targets: ['localhost:9090']
```

Monitoring Nodes (Servers) with Prometheus

Date : 2.17.1

MON TUE WED THU FRI SAT SUN

- To monitor nodes, you need to install node-exporter
- The node exporter will expose machine metrics of Linux / UNIX machines.
  eg. CPU usage, memory usage
- The node exporter can be used to monitor machines, and later on, you can create alerts based on these integrated metrics.
- For windows, there is VMI exporter

node-exporter on port 9100

```
curl localhost:9100
curl localhost:9100/metrics
```

```
ps aux | prometheus
systemctl reload prometheus
kill -HUP <pid>
journalctl -n100
```

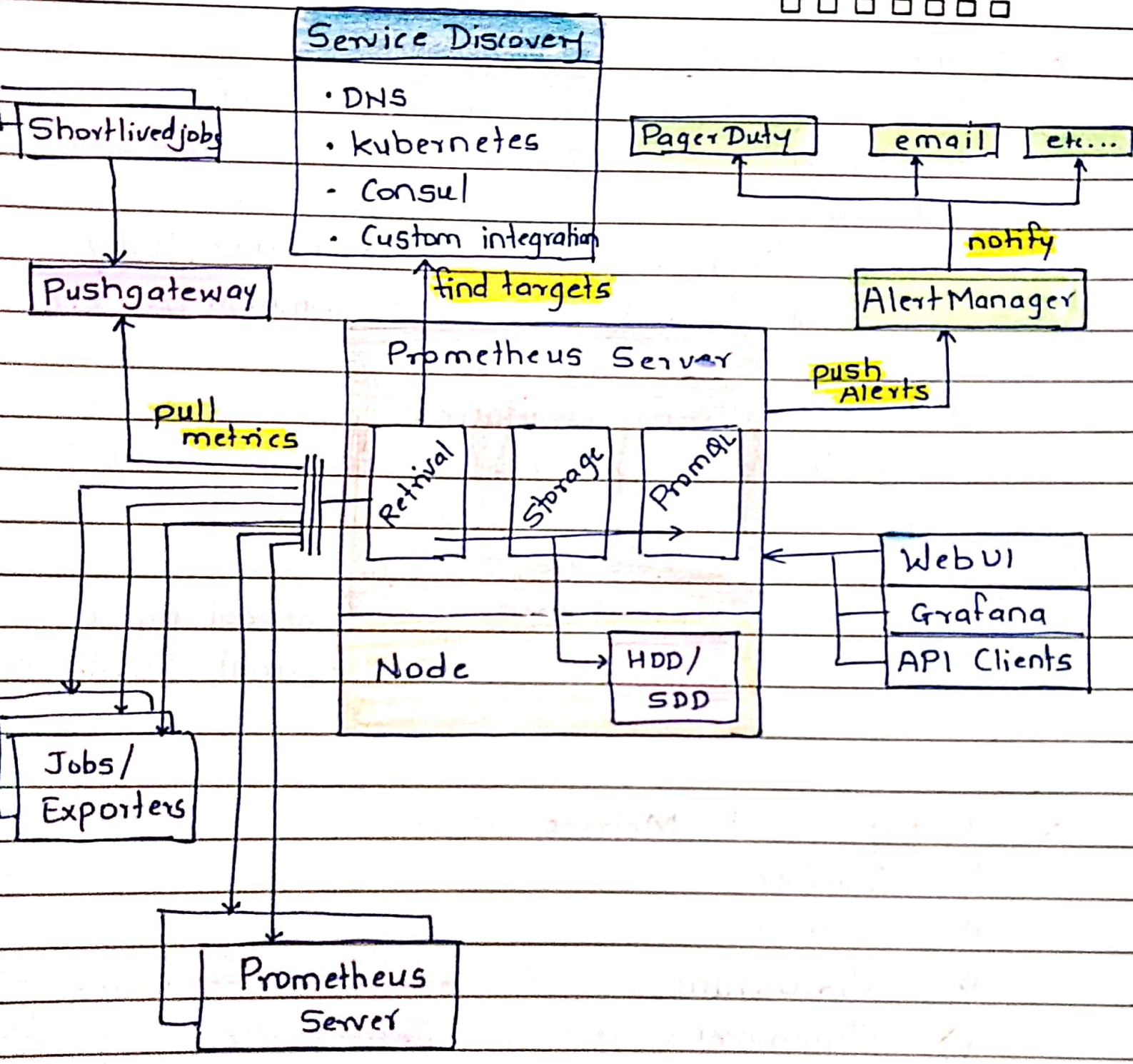
mv old source dest

left → right

Prometheus Architecture

Date :.....

MON TUE WED THU FRI SA SUN



Monitoring

Date :.....

MON TUE WED THU FRI SAT SUN

* Client libraries
  - Before monitoring the code, you should instrument your code
  - libraries
    - official : Go, Java/Scala, Python, Ruby
    - unofficial : Bash, C++, Common Lisp, etc.

  - No client library available?
    - implement it yourself in one of the supported exposition formats

Simple text-based format

Protocol buffer format (earlier v2)

* 4 types of Metrics
  1) Counter
  2) Gauge
  3) Histogram
  4) Summary

Date :.....

MON TUE WED THU FRI SA SUN

* Counter - value that only goes up
  - eg. visits to website

* Gauge - Single numeric value that can go up and down.
  eg. CPU load, temperature.

* Histogram - Samples observations. (eg. request durations or response sizes) and these observations get counted into buckets. Includes (-count & -sum)
  - Main purpose is calculating quantiles

* Summary - Similar to Histogram, a summary samples observations (eg. request durations or response sizes)
  A summary also provide total count of observations and sum of all observed values, it calculates configurable quantiles over sliding time window.

eg. you need 2 counters for calculating latency

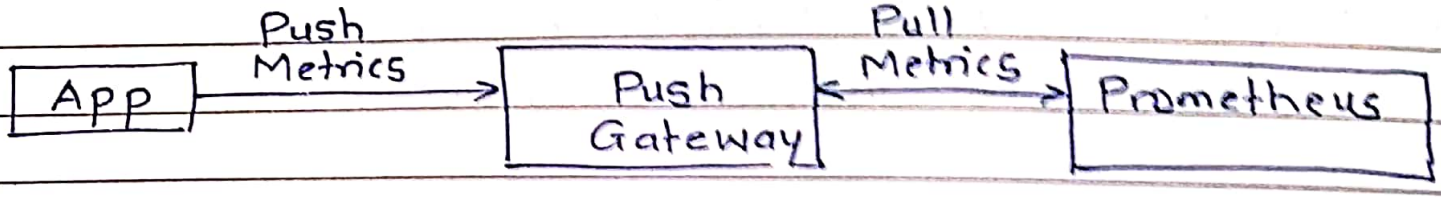
① total request (count)
② total latency of those requests (sum)

Average latency = Take rate() & divide

Pushing Metrics - Introduction

Date : .....

MON TUE WED THU FRI SAT SUN



- Prometheus by default prefers pull based matrics collection. i.e. prometheus will collect metrics at certain interval from your systems and store it in database.

- But, for example, a batch job may not exist long enough to be scraped. This job can instead push the metrics to push gateway.

- push gateway is also component from prometheus project and can be installed separately.

* Sometimes, metrics cannot be scraped eg. batch jobs, servers are not reachable due to NAT, firewall

* Pushgateway is used as an intermediary service which allows you to push metrics

* Pitfalls of push gateway

1) Most of the time, this is the single instance, so this results in SPOF (single point of failure)

2) Prometheus's automatic instance health monitoring is not possible

3) The push gateway never forgets the metrics unless they are deleted via the api.

eg. curl -X DELETE http://localhost:9091/metrics/job/...

* Only one valid use-case for pushGateway.
  - Service-level batch jobs and not related to specific machine.

* If NAT or/both firewall is blocking you from using pull mechanism.
  - Move prometheus server on the same network

Querying Metrics - Introduction

Date :.....

MON TUE WED THU FRI SAT SUN

- Prometheus provides a functional expression language called PromQL
  - provides built in operators and functions
  - vector based calculations like Excel
  - Expressions over time series vectors

- PromQL is read only.
- Example

```
100 - (avg by (instance) (irate(node-cpu-seconds_total{
    job='node-exporter', mode="idle"}[5m]))*100)
```

Instant vector : A set of time series containing a single sample for each time series, all sharing the same timestamp.
eg. node-cpu-seconds-total

Range vector : a set of time series containing a range of data points over time for each time series
eg. node-cpu-seconds-total[5m]

Scalar : a simple numeric floating point value
eg. -3.14

Date :.....

MON TUE WED THU FRI SA SUN

String - a simple string value; currently unused
eg. foobar

* OPERATORS

- Arithmetic binary operators
  - `-` substraction
  - `*` multiplication
  - `/` division
  - `%` modulo
  - `^` power / exponentiation

- Comparison binary operators
  - `==` equl
  - `!=` not equal
  - `>`
  - `<`
  - `>=`
  - `<=`

- logical / set binary operators
  - and (intersection)
  - or (union)
  - unless (complement)

- Aggregation Operators
  sum - calculate sum over dimensions
  min
  max
  avg
  stddev - population standard deviation
  stdvar - —"— —"— variance
  count
  count_values → count no. of elements with same value
  bottomk → smallest k elements by sample value.
  topk → largest k ...
  quantile → calculate ϕ-quantile ($0 \leq \phi \leq 1$)

* Querying Examples
  http://prometheus.io/docs/prometheus/latest/querying/examples

Prometheus

Date :.....

MON TUE WED THU FRI SA SUN

Service Discovery

* Having to manually update list of machines in configuration file gets quite annoying after a while.

* Service discovery allows you to automatically discover and monitor your service or services.

* Service discovery is automatic detection of devices and services offered by these devices on computer network.

*
```
static_configs:
    - targets: ["localhost:9090"]
```
this is not really a service discovery mechanism

* Prometheus has support for
  - cloud providers (AWS, AZURE, Google)
  - cluster managers (kubernetes, Marathon, etc)
  - generic mechanisms (DNS, Consul, Zookeeper)

Prometheus

Date :.....

MON TUE WED THU FRI SAT SUN

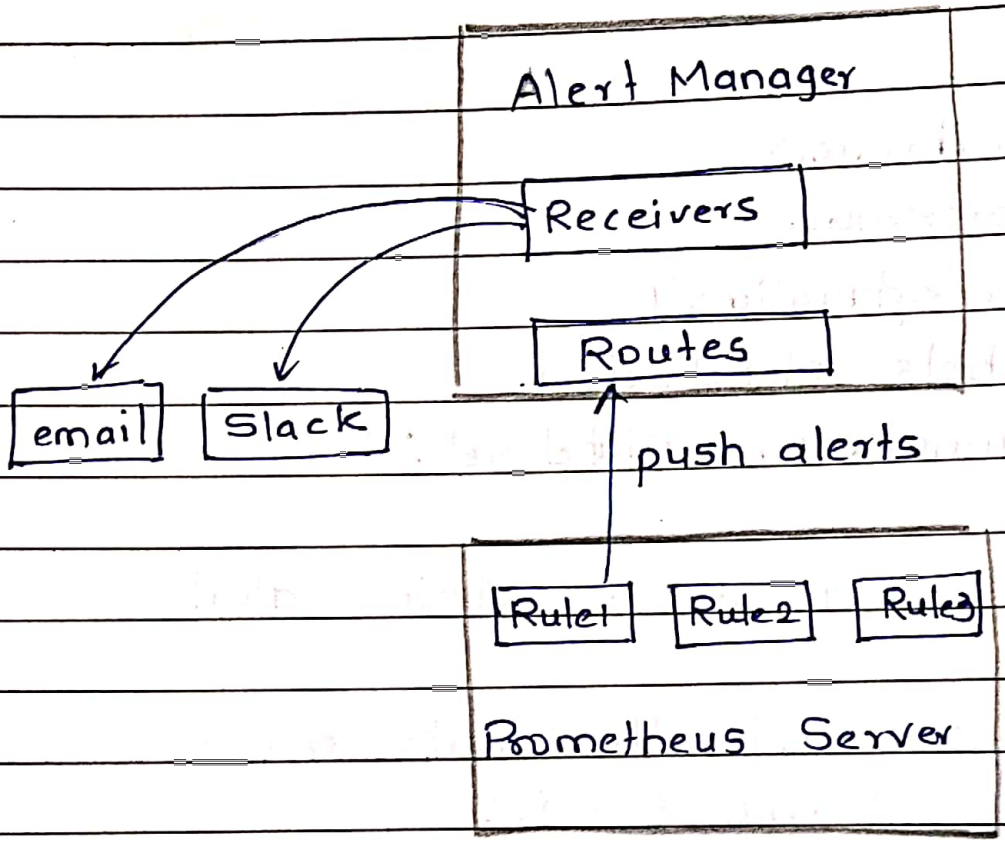
Exporters

- prometheus was initially developed for the purpose of monitoring web services.
- exporters are build for exporting prometheus metrics from existing 3rd party metrics.
- examples - MySQL server exporter
  Memcached exporter
  Consul exporter
  Node/ System metrics exporter
  MongoDB
  Redis ...

Alerting

Date :.....

MON TUE WED THU FRI SA SUN



Alerting in prometheus is separated in 2 parts

1) Alerting rules in prometheus server
2) Alertmanager

1.* Alerting Rules

- Rules live in Prometheus server config
- Best practice to separate the alerts from prometheus config.
  - Add an include in prometheus.yml

```
rule_files:
  - "etc/prometheus/alerts.rules"
```

Alert format

```
alert <alertname>
  if <expression>
    [ for <duration> ]
    [ labels <label set> ]
    [ annotations <label set> ]
```

- Alerting rules allow you to define alert conditions
- Alerting rules send the alerts being fired to an external service
* The format of these alerts in in Prometheus expression language.

Alertmanager –

- handles the alerts fired by prometheus server
- handles deduplication, grouping and routing of alerts
- Routes alerts to receivers (PagerDuty, Opsgenie, email, slack, etc.)
- Alertmanager configuration: /etc/alertmanager/alertmanager.yml

Alertmanager concepts -

1) Grouping : groups similar alerts into one notification
2) Inhibition : Silence /stop other alerts if one specified alert is already fixed.
3) Silences : A simple way to mute certain notifications.

- you can create a high available Alertmanager cluster using mesh config.
- Do not load balence this service
  - use list of Alertmanager nodes in Prometheus config
- All alerts are sent to all known alertmanager nodes
- Guarantees the notification is at least Send once.

Alert states

Inactive - No rule is met

Pending - Rule is met but can be suppressed due to validations

Firing - Alert is sent to configured channel (mail, slack, ...)

- Alertmanager runs on port 9093

Prometheus Storage

Date : .....

MON TUE WED THU FRI SA SUN

- You can use the default local on-disk storage, or optionally the remote storage system

- Local storage: A local time series database in custom prometheus format.

- Remote storage: you can read/write samples to remote system in standardized format.
  → Currently it uses snappy-compressed protocol buffer encoding over HTTP, but might change in future. (to use gRPC or HTTP/2)

- Remote storage is primarily focussed on long term storage

- Currently there are adapters available for following solutions

AppOptics : write	Graphite : write
Chronix : write	InfluxDB : read & write
Cortex : read and write	OpenTSDB - write
CrateDB - read and write	PostgreeSQL/TimeScaleDB - Read & write
Gnocchi - write	SignalFix : write

Time Series Database

Date : .....

MON TUE WED THU FRI SAT SUN

- database which is completely optimised for timestamp or timeseries data.

- Time series is simply measurements or events that are tracked or monitored, collected or aggregated over a period of time
  eg. data collected from heartbeats or motion tracking sensors or collecting jvm metrics from JVM collecting trades in market collecting network data, etc

- Time series database are completely customised with time-stamped data which is indexed and which has been efficienty written in such a way that you can insert time series data and query it in much faster way than relational database

- In other databases, we query ~~values~~ with values of different datatypes however in time series database, we ask question only over time.

- How workloads are performed?
- How data life cycle is managed / summarised?

- Initially time series databases are used in stock markets for trading information

- TSDB helps in monitoring microservices or collecting stats from these microservices and from those stats, we can analyze & come up with patterns.

Local Storage

Date :.....

MON TUE WED THU FRI SAT SUN

- Prometheus >=2.0 v. uses a new storage engine which dramatically increases scalability
- Ingested samples are grouped in blocks of two hours.
- Those 2h samples are stored in separate directories (in the data directory of prometheus)
- Writes are batched and written to disk in chunks, containing multiple data points

directory 1	directory 2	directory 3
2h of data	2h of data	2h of data
chunks /00001	chunks /0000)	chunks/0000)
chunks /00002	chunks /0000	
index meta.json	index meta.json	index meta.json

- Every directory has an index file (index) and metadata file (meta.json)
- It stores the metric names and labels, and provides an index from metric names and labels to the series in chunk files.

- The most recent data is kept in memory
- You don't want to loose the in memory data during a crash, so the data also needs to be persisted to disk
  This is done using WAL write ahead logs

directory 1	directory 2	directory 3	WAL
chunks/000001	chunks/000001	chunks/000001	000001
chunks/000002		chunks/000002	000002
meta.json	meta.json	meta.json	
index	index	index	
tombstone	tombstone	tombstone	

- Write Ahead Log (WAL)
  → its quicker to append to file (like a log) than making (multiple) random reads/writes.
  → If there is server crash & data from memory is lost, then WAL will be replayed
  → This way, no data will be lost or currupted during crash

- **tombstone file**
  - → when series gets deleted, a tombstone file gets created for each directory
  - → This is more efficient than immediately deleting the data from chunk files as actual delete can happen at later time (eg. when there is not a lot of load)

* **Compaction** -

- The initial 2 hour blocks are merged in the background to form longer blocks this is called compaction.

directory 1
2h of data
chunks/000001
chunks/000002

directory 2+3
4h of data
chunks/000001
chunks/000002

* Block Characteristics -

- A block on filesystem is a directory with chunks
- you can see each block as fully independent database containing all time series for the window.
- every block of data except the current block, is immutable (no changes can be made)
- These non overlapping blocks are actually horizontal partitioning of ingested time series data.

* Benefits of Horizontal partitioning -

- When querying, the blocks not in time range can be skipped
- When completing block, data only needs to be added and not modified (avoids write amplification)
- Recent data is kept in memory, so can be queried quicker
- Deleting old data is only a matter of deleting directories on filesystem.

* **Compaction** -

- When querying, blocks have to be merged together to be able to calculate the results
- Too many blocks can cause too much merging overhead, so blocks are compacted
- 2 blocks are merged and form a newly created (often larger) block
- Compaction can also modify data: dropping deleted data or restructuring the chunks to increase the query performance

* **Index** -

- Having horizontal partitioning already makes most queries quicker, but not those that need to go through all the data to get the result.
- The index is an inverted index to provide better query performance, also in cases where all data needs to be queried.
- Each series is assigned a unique ID.

What about disk size?

- on average, prometheus needs 1-2 bytes per sample
- You can use following formula to calculate disk space needed -

Needed disk space = retention_time_seconds * ingested samples per second * bytes per sample.

How to reduce disk size?

① increase scrape interval, which will get you less data
② You can decrease the targets or series you scrape
③ reduce the retention (How long you keep the data)