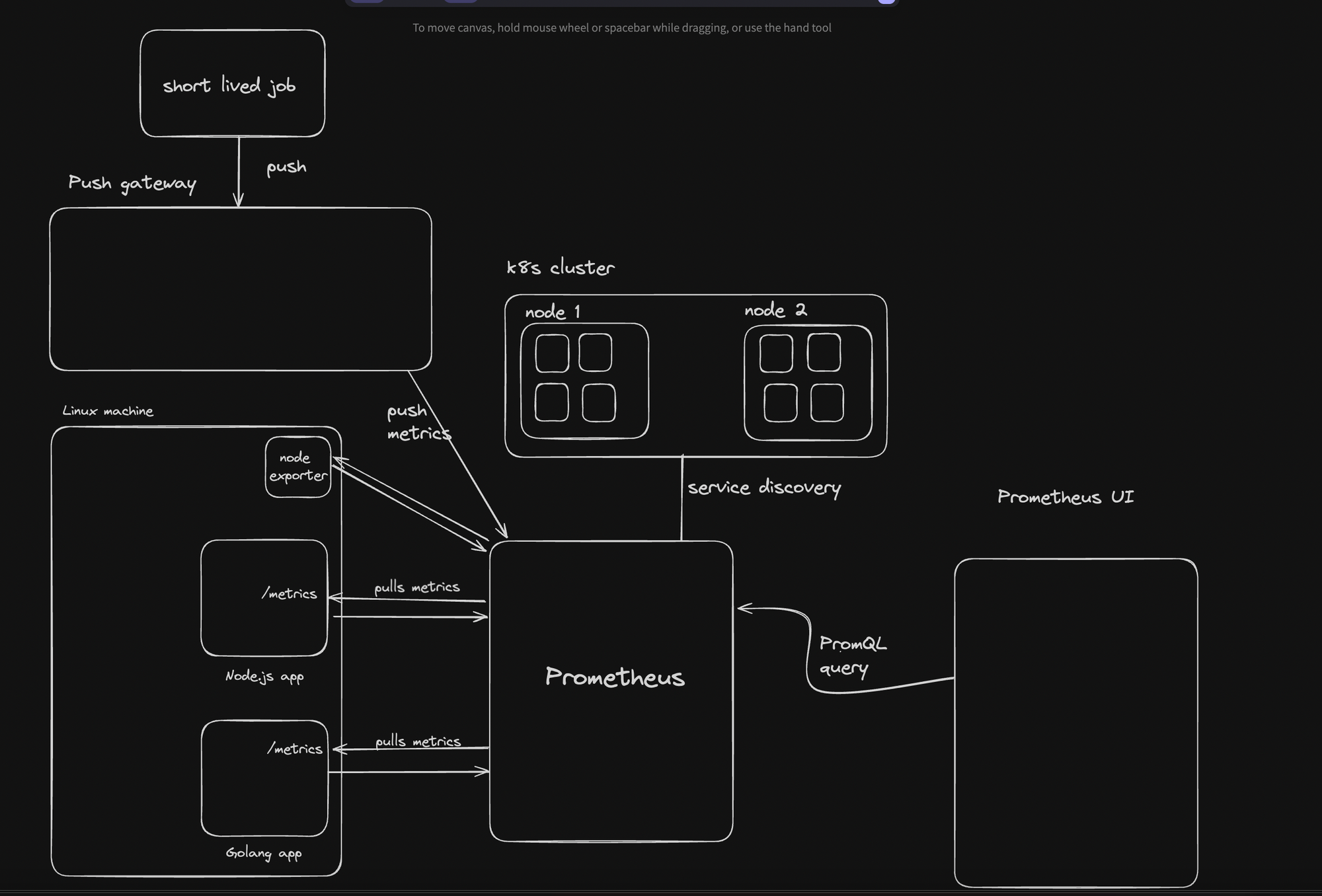
Prometheus And Grafana (Monitoring Logs using Dashboard)



Prometheus collects logs from various sources like docker, node, system using third party library. Generally every application logs its details for example docker dumps its log /var/lib/docker/

Step1-So whenever I will be using third party docker files I will map this host:docker folder for that open source to collect metrics.  
Step2-I will create service in Prometheus which will integrate these open source and in defined interval periodicly checks those open source and collect metrics. These metrics has logs refined in form of cpu vs time, memory vs time,etc.  
Step3-Integrate Prometheus into Grafana and set data source like which open source docker or nodejs wala and then plot these data in graph. Inside Grafana we have PromQL lets you query on top of all your timeseries data. sum(http\_requests\_total{job="api-server", status="500"})

Theory

1. Time series collection happens via a pull model over HTTP

Prometheus primarily uses a pull model to collect metrics:

1. Pull Model: Prometheus periodically scrapes metrics from configured targets by making HTTP requests to the /metrics endpoint exposed by the targets. This approach allows Prometheus to control the scraping intervals and retry logic.
2. Targets expose their metrics in a specific format that Prometheus understands, typically using client libraries provided by Prometheus for various languages and environments.

2- Prometheus offers several ways to visualize and interact with the collected metrics:

* 1. Prometheus UI: A built-in web interface for ad-hoc queries and simple graphing.
  2. Grafana: A popular open-source dashboarding tool that integrates well with Prometheus, providing rich visualization and dashboarding capabilities.

3- This is because prometheus is not a DB, it just exposes all the metrics on an endpoint. That endpoint cant server all the data, and hence prometheus doesnt store the exact values

\*NodeJS Application Monitoring (Prom-Client to collect metrics)

UseCase-> Based on endpoints of api /user /metrics etc we can collect metrics as total http request, total active request, histogram vs time taken by that request. It tells how my api is behaving response time plot.

<https://projects.100xdevs.com/tracks/prom-graf-1/Prometheus-and-Grafana-1> (Complete Project Details)

prom-client-> client.counter .gauge .histogram different way to capture and return metrics.  
we are basically creating out own metrics and exposing this endpoint api/metrics and sending this in response so that Prometheus can understand this.

The rate function in Prometheus is used to calculate the per-second average rate of increase of a counter over a specified time range\*\*

1. Build normal express app listening at 3000
2. Create a middleware that tracks the total time to handle a request: Just for implementing middleware .Now we will create middleware to get metrics using open source- prom-client
3. Metrics/ requestCounter = new client.Counter requestCounter.inc(…). This will create metrics counter how many time that complete label occurred . Example first tell counter name and add label[method,route,status] increment the value using this label. Counter uses label and matches that with value and increment

# HELP http\_requests\_total Total number of HTTP requests

# TYPE http\_requests\_total counter

http\_requests\_total{method="GET",route="/metrics",status\_code="200"} 3

http\_requests\_total{method="GET",route="/users",status\_code="404"} 1

http\_requests\_total{method="GET",route="/user",status\_code="304"} 1

1. Adding gauge-> total count no need of label matching as in counter

activeRequestsGauge = new client.Gauge(name de do) activeRequestsGauge.inc(); activeRequestsGauge.dec(); //increase and decrease this count value .

# HELP active\_requests Number of active requests

# TYPE active\_requests gauge

active\_requests 1

1. Histograms let you store data in various buckets in a cumulative fashion. Count of requests wrt time that it occurred <5ms 5 request , <10ms 10 request aisa kuch. this is cumulative. Number of requests being handled in less than 5000ms = Number of requests being handled in less than 3000ms + Number of requests that took between 3000-5000ms.

httpRequestDurationMicroseconds = new client.Histogram

httpRequestDurationMicroseconds.observe({},duration=>endTime-startTime kis bucket mein jayega)

Step- Prom Docker image

Since we made our nodejs app using prom-client now Prometheus will build image of prom as this is our third party library which goes to target of nodejs and extract /metrics endpoint api which is returning prom client data. PromClient create and collect metrics inside nodejs and prom as a docker service is collecting those using /metrics endpoint.

app.get("/metrics", async (req, res) => {

const metrics = await client.register.metrics();

res.set('Content-Type', client.register.contentType);

res.end(metrics);

})

This endpoint will return all registered metrics from prom-client, client.counter ,client.histogram, client.gauge, etc.

Step- Starting Prometheus, Grafanace and docker images (see docs)

* Make sure nodejs is running inside docker to make :3000 of node expose to prom can call /metrics inside docker. Create DockerFile and up it
* Create Prometheus.yml – all config for Prometheus: target endpoint: scrape interval
* Create docker.yml: nodejs app + Prometheus(config file location volume map)
* Open Prometheus on 9090 and you can see all your prom-client names of metrics declared
* http\_requests\_total, histogram, count-> can also add sum,avg,rate functions along with these metrics

3 Types of data collected-> now can use these and add more function sum,avg,rate inside this

client.counter()  
http\_requests\_total Total number of HTTP requests

http\_requests\_total{method="GET",route="/",status\_code="404"} 1

client.histogram() le is bucket type

http\_request\_duration\_ms Duration of HTTP requests in ms

http\_request\_duration\_ms\_bucket{le="0.1",method="GET",route="/metrics",code="200"} 0

client.gauge()

active\_requests Number of active requests

active\_requests 1

Variation on these metrics to get meaningful info -> can tell interviewer

1. **Average time it took to handle all requests**

http\_request\_duration\_ms\_sum / http\_request\_duration\_ms\_count

1. rate at which requests are coming. Rate(http\_requests\_total)
2. Average HTTP request duration with timeseries (5 minute buckets)

rate(http\_request\_duration\_ms\_sum[5m]) / rate(http\_request\_duration\_ms\_count[5m])

* Now the way we can query our metrics in prometheus use grafana add data source as prometheus and display in more good way. Docker-compose.yml add Grafana as service. Source url- <http://prometheus:9090> where Prometheus is running
* Based on endpoint /user /metrics create dashboard, you can filter these metrics as these metrics hase labels{endpoint:/metrics} and filter those metrics and plot sum,rate
* 

\*Docker Images Monitoring (CAdvisory to collect metrics)->

UseCase-> Docker instances taking cpu, memory, disk i/o, network i/o operations for different instances plot rate time graph ,cpu vs time for each instance of image.

Step->

* Cadvisor image map - /var/lib/docker/:/var/lib/docker:ro which means docker all metrics are store inside our computer at this location just map this location and cadvisor collects all metrics.
* CPU Usage: container\_cpu\_usage\_seconds\_total: This metric provides the total CPU time consumed by the container in seconds.

rate(container\_cpu\_usage\_seconds\_total{image!="",container\_name!=""}[5m])

The rate function calculates the per-second average rate of increase of the CPU time over the last 5 minutes.

* Memory Usage: container\_memory\_usage\_bytes: This metric provides the current memory usage of the container in bytes.

container\_memory\_usage\_bytes{image!="",container\_name!=""}

This provides a direct measure of the memory usage for each container.

* Network I/O: container\_network\_receive\_bytes\_total These metrics provide the total number of bytes received and transmitted by the containe

rate(container\_network\_receive\_bytes\_total{image!="",container\_name!=""}[5m])

rate(container\_network\_transmit\_bytes\_total{image!="",container\_name!=""}[5m])

The rate function calculates the per-second average rate of bytes received/transmitted over the last 5 minutes.

* Disk I/O: These metrics provide the total number of bytes read from and written to the filesystem by the container.

rate(container\_fs\_reads\_bytes\_total{image!="",container\_name!=""}[5m])

rate(container\_fs\_writes\_bytes\_total{image!="",container\_name!=""}[5m])