Monitoring and Telemetry for Production Systems

Kawtar Oukil

Table des matières

I.Introducttion:

Monitoring & Telemetry for Production Systems is a project about various methods of monitoring Production Servers to ensure Reliability of Web Services, Sites, Servers and Applications.

learning objectives:

- Monitoring & Telemetry Fundamentals.
- Linux Command line Monitoring Tools.
- Server & Docker Container Monitoring with Netdata & Cadvisor.
- Perform Application Monitoring & Telemetry with Prometheus & Grafana - Part I
- Perform Application Monitoring & Telemetry with Prometheus & Grafana - Part II
- Implement Log Monitoring & Analysis with ELK Stack.

For a successful DevOps setup, a good monitoring platform lets you monitor infrastructure and application performance, whether on-premise, in the cloud, or across containerized environments — so you have complete visibility into every system, all the time

Monitoring provides feedback from production. Monitoring delivers information about an application's performance and usage patterns.

One goal of monitoring is to achieve high availability by minimizing time to detect and time to mitigate (TTD, TTM). In other words, as soon as performance and other issues arise, rich diagnostic data about the issues are fed back to development teams via automated monitoring. That's TTD. DevOps teams act on the information to mitigate the issues as quickly as possible so that users are no longer affected. That's TTM. Resolution times are measured, and teams work to improve over time. After mitigation, teams work on how to remediate problems at root cause so that they do not recur. That time is measured as TTR.

A second goal of monitoring is to enable “validated learning” by tracking usage. The core concept of validated learning is that every deployment is an opportunity to track experimental results that support or diminish the hypotheses that led to the deployment. Tracking usage and differences between versions allows teams to measure the impact of change and drive business decisions. If a hypothesis is diminished, the team can “fail fast” or “pivot”. If the hypothesis is supported, then the team can double down or “persevere”. These data-informed decisions lead to new assumptions and prioritization of the backlog.

Telemetry is the mechanism for collecting data from monitoring. Telemetry can use agents that are installed in the deployment environments, an SDK that relies on markers inserted into source code, server logging, or a combination of these. Typically, telemetry will distinguish between the data pipeline optimized for real-time alerting and dashboards and higher-volume data needed for troubleshooting or usage analytics.

Monitoring is often used to “test in production”. A well-monitored deployment streams the data about its health and performance so that the team can spot production incidents

immediately. Combined with a Continuous Deployment Release Pipeline, monitoring will detect new anomalies and allow for prompt mitigation. This allows discovery of the “unknown unknowns” in application behavior that cannot be foreseen in pre-production environments.

Effective monitoring is essential to allow DevOps teams to deliver at speed, get feedback from production, and increase customers satisfaction, acquisition and retention.

Here are some monitoring terminologies that you will find useful:

Server monitoring tools help users identify and solve any application hosting and performance issues by tracking and monitoring server performance. Monitoring provides visibility and insight into the performance of various servers in your IT infrastructure.

Server : In computing, a server is a piece of computer hardware or software that provides functionality for other programs or devices, called "clients". This architecture is called the client–server model.

Cloud: Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user. The term is generally used to describe data centers available to many users over the Internet.

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

Spring Boot Microservices are microservice applications built using the Java Spring boot framework which provides a set of features to quickly build microservices. This framework allows microservices to work together and can quickly setup services, with minimal configurations.

Application Performance Monitoring (APM) : An area of IT that focuses on monitoring the performance of software application programs in order to provide end-users with a quality experience.

Prometheus: Originally developed by SoundCloud is an open source and community-driven project that graduated from the Cloud Native Computing Foundation. It can aggregate data from almost everything Microservices, Multiple languages, Linux servers, Windows servers which can be used for generating visual representations with Grafana.

Grafana is a multi-platform open-source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources like Prometheus.

Docker is a set of platform as a service product that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.

cAdvisor (short for container Advisor) analyzes and exposes resource usage and performance data from running containers (Docker).

ELK stack is an acronym used to describe a stack that comprises of three popular open-source projects: Elasticsearch, Logstash, and Kibana. Often referred to as Elasticsearch, the ELK stack gives you the ability to aggregate logs from all your systems and applications, analyze these logs, and create visualizations for application and infrastructure monitoring, faster troubleshooting, security analytics, and more.

E= **Elasticsearch** Elasticsearch is an open-source, RESTful, distributed search and analytics engine built on Apache Lucene. Support for various languages, high performance, and schema-free JSON documents makes Elasticsearch an ideal choice for various log analytics and search use cases. L = **Logstash** Logstash is an open-source data ingestion tool that allows you to collect data from a variety of sources, transform it, and send it to your desired destination. With pre-built filters and support for over 200 plugins, Logstash allows users to easily ingest data regardless of the data source or type.

K = **Kibana** is an open-source data visualization and exploration tool for reviewing logs and events. Kibana offers easy-to-use, interactive charts, pre-built aggregations and filters, and geospatial support and making it the preferred choice for visualizing data stored in Elasticsearch.

Task-1 : Monitoring & Telemetry Fundamentals.

The importance of monitoring is to keep our software entities alive, be it website or be it web application, android, or iPhone app …etc.

The second importance of monitoring is to enable validated learning by tracking usage.

Ensuring the availability of the application, and the second goal is to enable learning.

Telemetry is the mechanism for collecting data for monitoring.

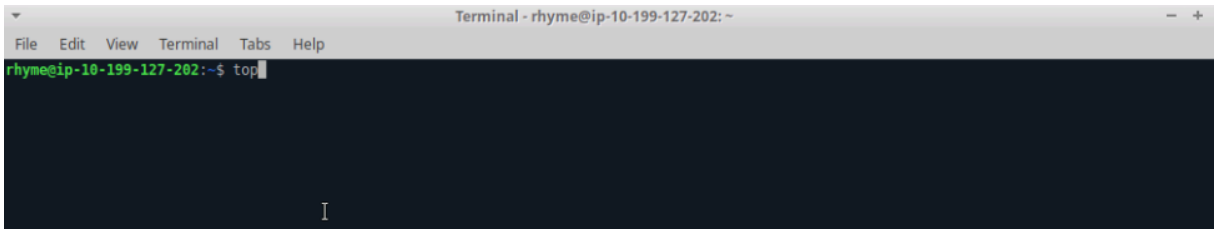
Task-2: Linux Command line Monitoring Tools

We gonna using commands to know the health of a system, to view log files and so on

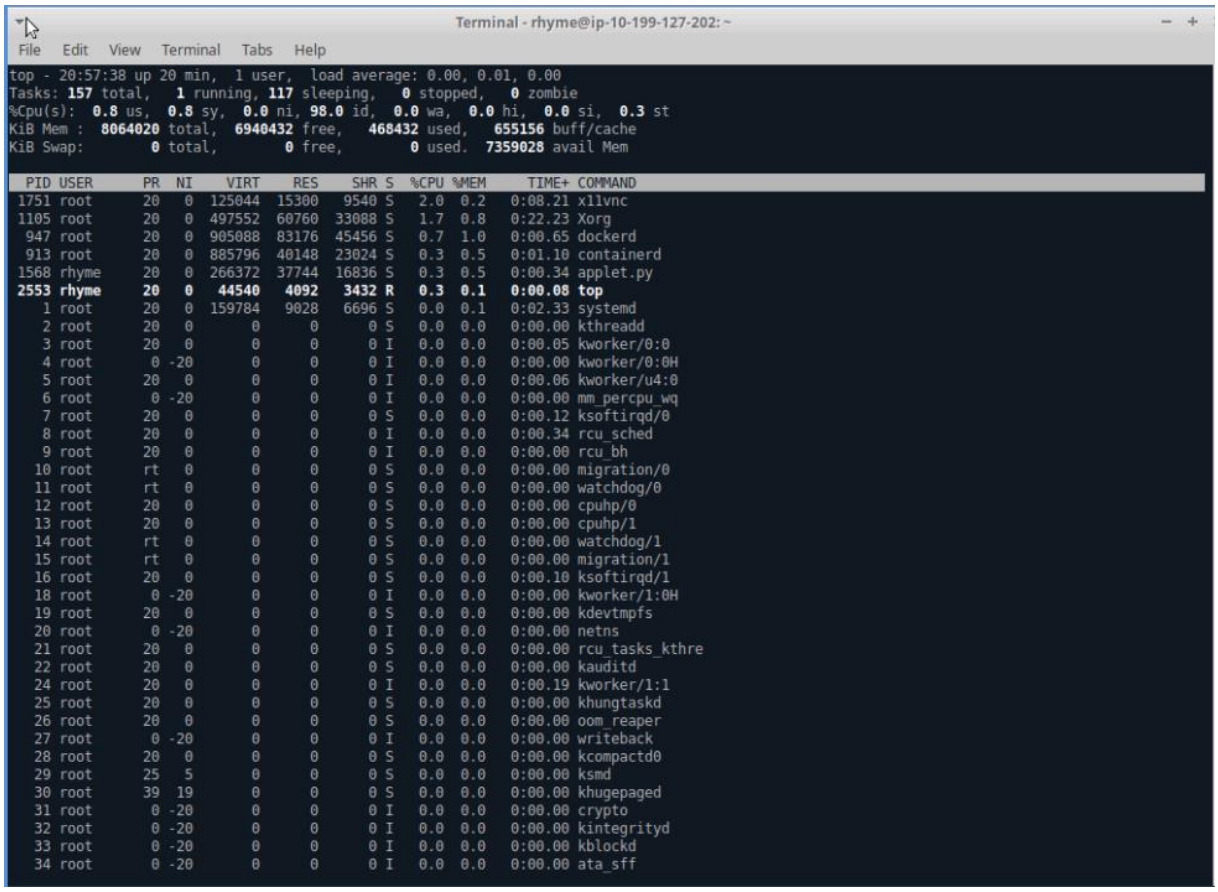
1. Top

Top is a basic command which majority of system admins use as part of their daily job. You don't need to install TOP as it's already part of every Linux distribution. Top commands display running tasks, their CPU, memory and swap usage. You can do whole lot of things to get the output of your liking and need. It has a large number of options/switches which you can use to make your life easy.

The Top command displays running tasks:



this command displays these metrics:



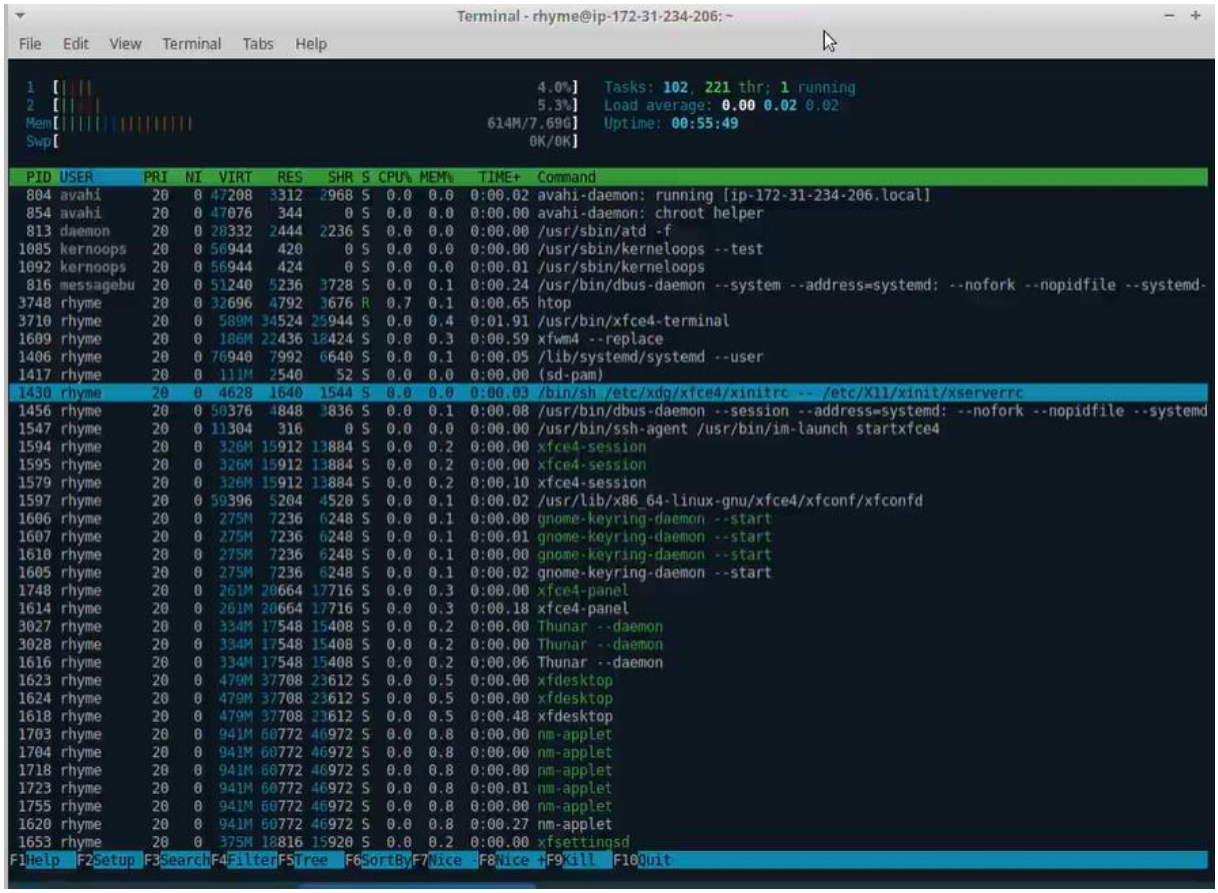
Number one is to see how much CPU is being used, so if we have a single core CPU or a multi core based on that, it can exhibit how much is being used. The second thing that the Top command shows us is the load average.

2. Htop

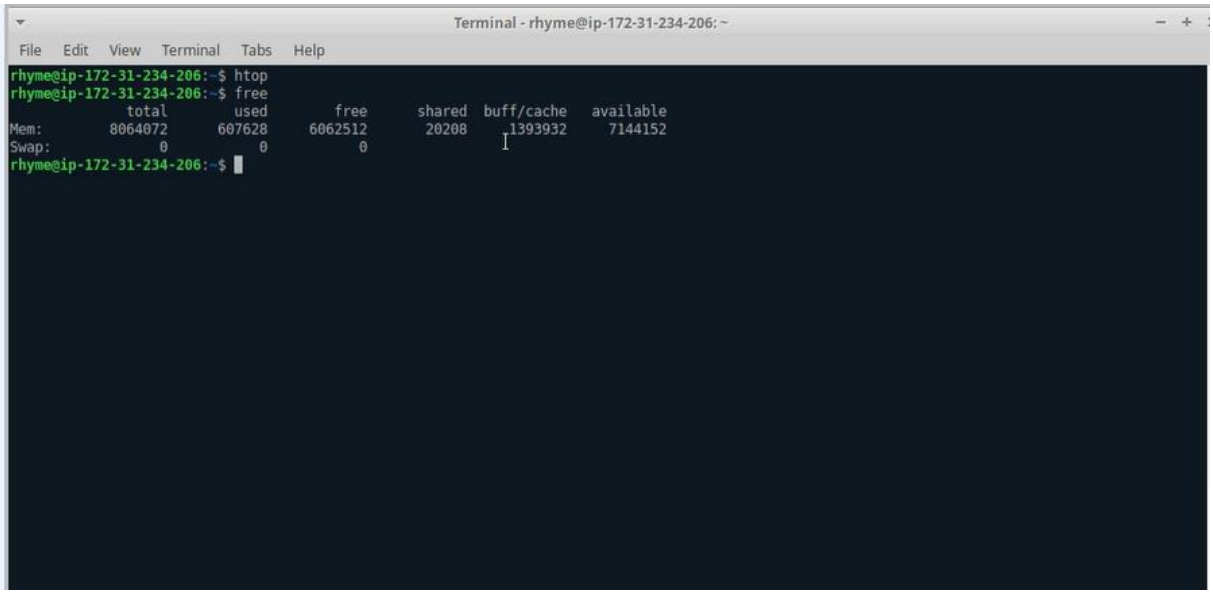
Htop is another tool just like "top" to monitor your systems processes. It comes with an interactive overview and you can kill processes by just going on them and pressing the desired button. It's better than top because it has different mediums of showing memory and swap. You can install htop by the following command.

yum install htop or apt-get install htop

So, htop is similar to top command but it has a better view, and then it's a bit more interactive than top.



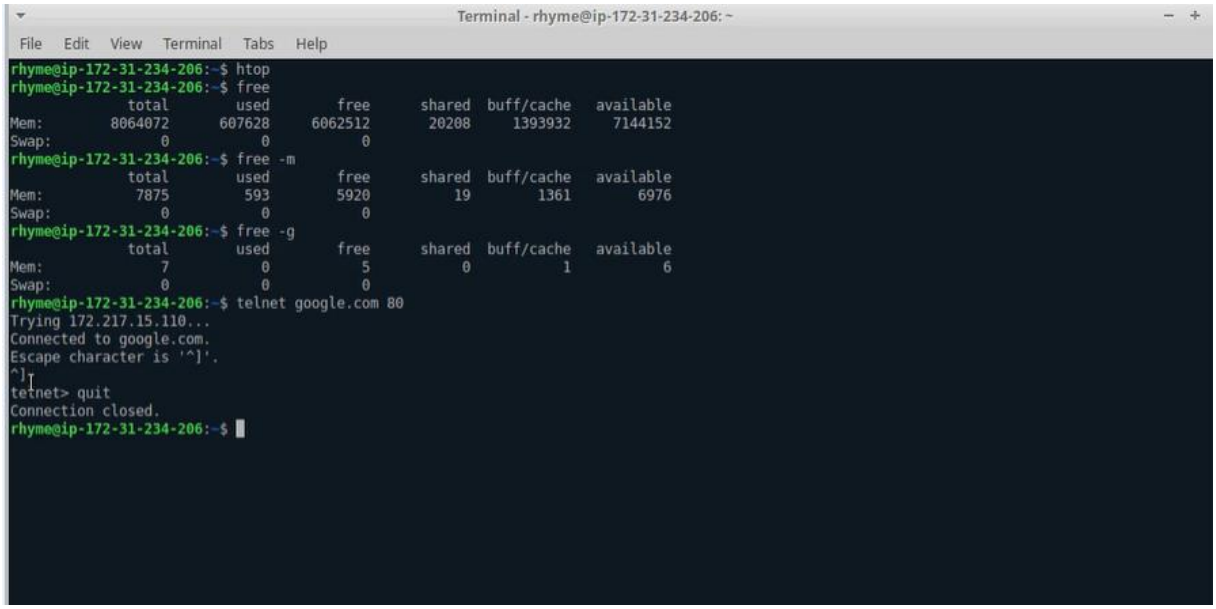
The command free comes preinstalled with Linux Distribution and is used to check the memory usage. It shows buffers cache, memory and the total memory



If we want to check if the website is open or not we can tape this command below:

So, if it is open, if it is connected, it will say connected to their particular domain and Escape character

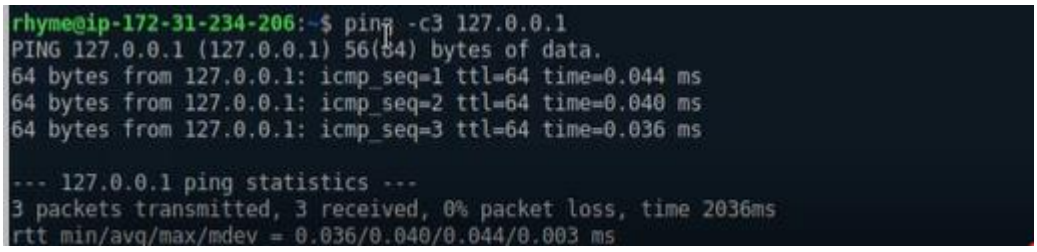
We press control and square closing bracket



Another command that is useful is to check, whether a particular website is available are reachable or not, or a particular IP is reachable or not.

So like we see above, In this case, it is no reachable.

Let's try it with IP address



So, here like we see that the site of this IP address s reachable in term of sending acknowledgement.

3. iostat

Let's try now another command: **iostat**

```
rhyme@ip-172-31-234-206:~$ iostat
Linux 4.15.0-1065-aws (ip-172-31-234-206)       12/16/20        _x86_64_        (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.43    0.00    0.85    1.85    0.64   95.23

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0.02         0.30         0.00       1086          0
loop1             2.71         2.79         0.00      10173          0
loop2             0.09         0.37         0.00       1350          0
loop3             0.06         0.14         0.00        507          0
loop4             0.01         0.09         0.00        328          0
loop5             0.06         0.14         0.00        508          0
loop6             3.18         3.45         0.00      12580          0
loop7             0.09         0.37         0.00       1345          0
nvme0n1          20.48       277.90        69.65    1012283     253700
loop8             0.00         0.00         0.00          8          0
```

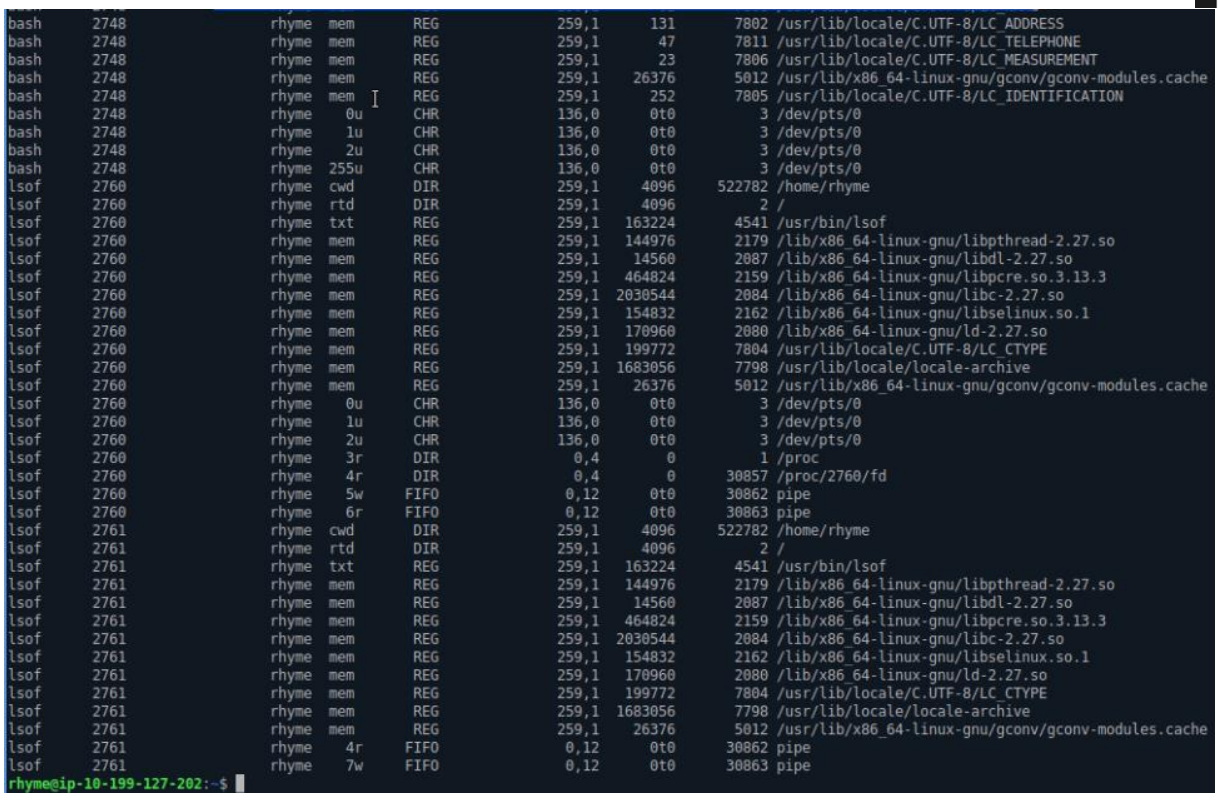
iostat command reports us the CPU and disc input output statistics.

4. Lsof

Now, we gonna see another command called '**lsof**':

```
rhyme@ip-172-31-234-206:~$ lsof
```

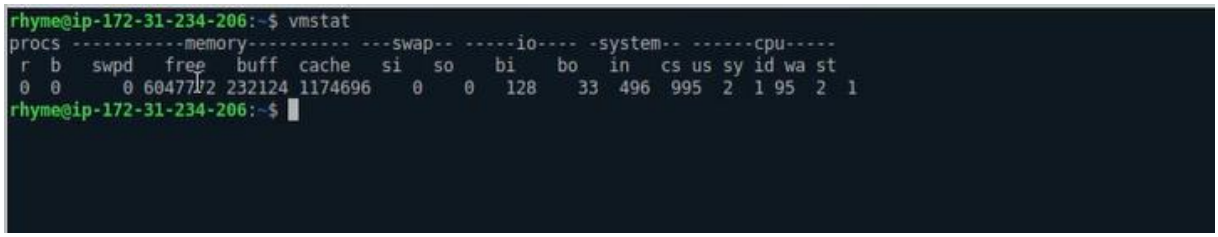
This command lsof prints tabular format of rows and columns. We know that in every Linux System, each and every process opens file on the backend. So, in Linux or Unix, everything is a file. So, if we want to check that the process respond or not, we have to check the file opened for that process in respective partition or directory



So, in this table above, we can see the file opened.

5. Vmstat

Another command that allow us to monitor is '**vmstat**'



So, we can see in the result the size of virtual memory, of swap, and how much free memory is being, so this vmstat command is similar to 'free' and 'mnstat' command.

6. netstat

Another useful command is '**netstat**'

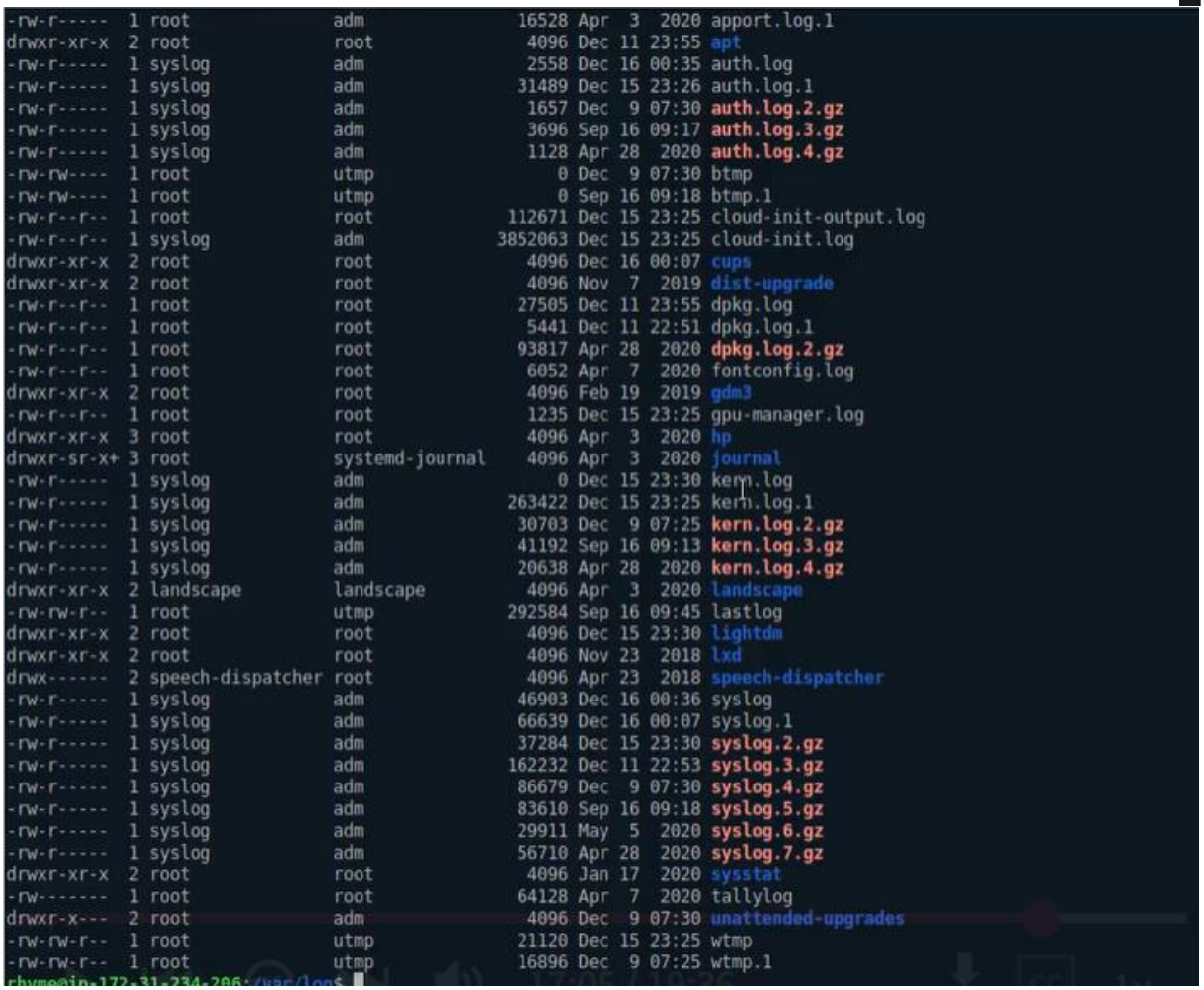
```
rhyme@ip-172-31-234-206:~$ netstat -tulpn
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:45425         0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:36202           0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 :::36202                :::*                    LISTEN      -
tcp6       0      0 :::5900                 :::*                    LISTEN      -
udp        0      0 0.0.0.0:48280           0.0.0.0:*                           -
udp        0      0 127.0.0.53:53           0.0.0.0:*                           -
udp        0      0 172.31.234.206:68       0.0.0.0:*                           -
udp        0      0 0.0.0.0:631             0.0.0.0:*                           -
udp        0      0 224.0.0.251:5353        0.0.0.0:*                           3029/chrome --passw
udp        0      0 224.0.0.251:5353        0.0.0.0:*                           3065/chrome --type=
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           -
udp6       0      0 :::5353                 :::*                                -
udp6       0      0 :::50716                :::*                                -
```

So, we see the information about the PID, the IP address, the ports in the system. This command allow us to see whether port is listening on certain interface in our system or not.

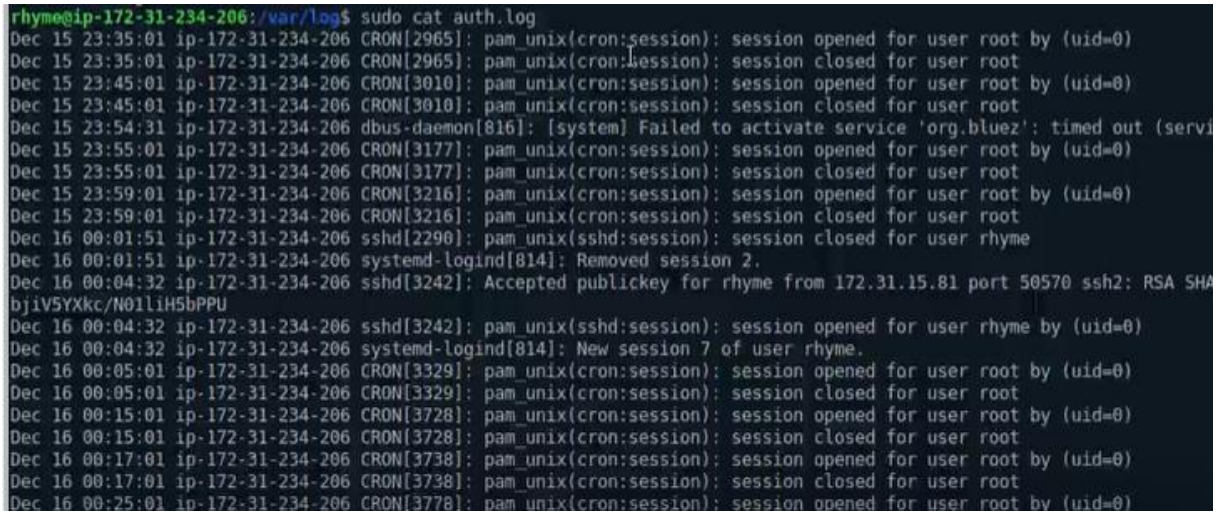
Another useful thing that we do from command line is view logs. In this order we tape:

```
rhyme@ip-172-31-234-206:~$ cd /var/log/
rhyme@ip-172-31-234-206:/var/log$
```

And tape the **'ls -l'** command:



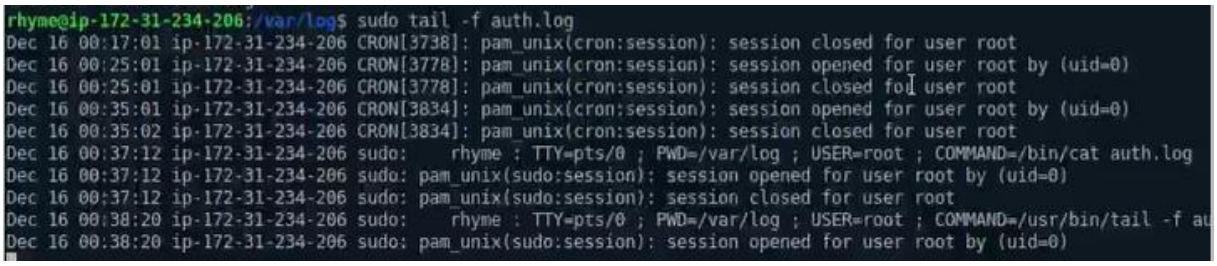
We can see that there is a logs file, and we can see the content of one of this log files, like ‘auth.log’



As we can see above, the content of the file have a lot of events that occurred in the time stamps and the date stamps and the IP address from which the request was coming for a log in session whether it was opened or not.

So, If you have ngnix server, consider ngnix logs file. If you want the servers normal kernel messages, we can look at /var/log/messages and so on.

And for watching real time logs, we can also use this command **tail -f** and the name of the log file.



So here **tail -f**, as this shows us a real time log, like If some even dockers, we can see that in real time.

So, this is all about how we monitor, using Linux command line and how we follow various logs in the forthcoming session

Task-3: Server & Docker Container Monitoring with Netdata & cAdvisor.

In this task, we will understand how to do a graphical Monitoring using a tool called Net data, we will also see how we monitor Docker Containers and how will launch it.

Must of applications are containerized and run on orchestrators like Mesos and Kubernetes, so it's also important to know how docker containers are monitored and we will use graphical tools to monitor docker which is called cAdvisor

Netdata is a system monitoring, designed to be distributed, lightweight, flexible and open source.

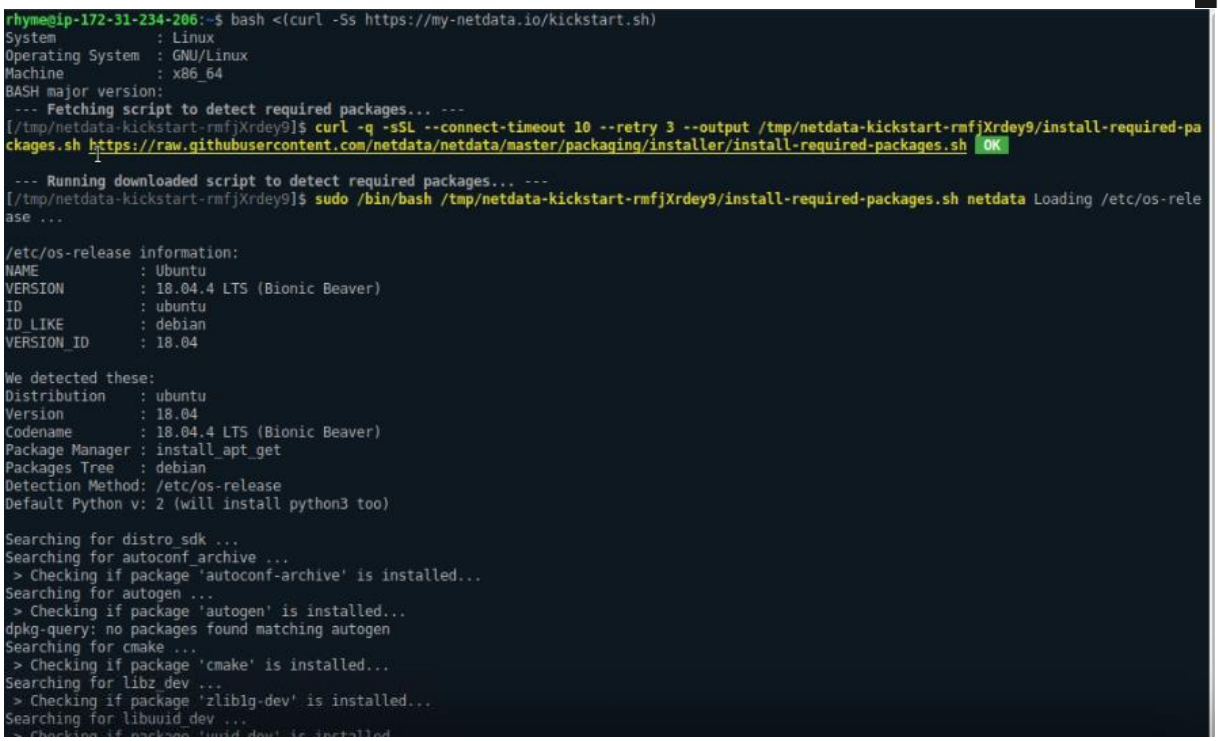
The application is able to detect hundreds of metrics automatically, simplifying their configuration and integration. The application core is written in C and hundreds of collectors are available to collect thousands of different metrics while being very lightweight and not requiring additional computing power. Netdata can either run autonomously, without any third-party components, or it can be integrated to existing monitoring toolchains like Prometheus, Graphite, OpenTSDB, Kafka, Grafana, and more.

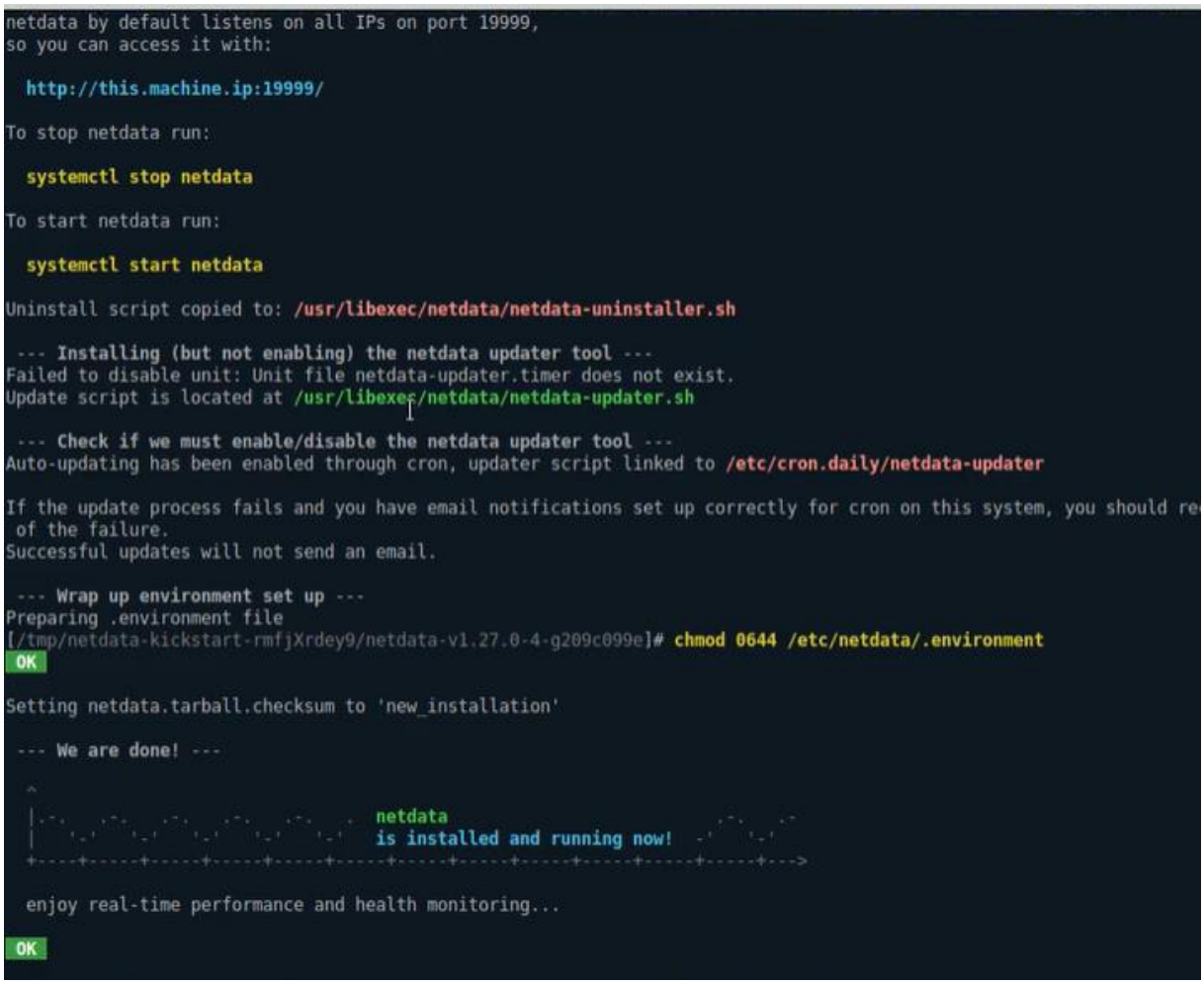
Let's get started by installing Netdata, firstly we open up the terminal and let's install it.

1. Installing Netdata:

We gonna install Netdata with this command:

#bash <(curl -Ss https://my-netdata.io/kickstart.sh)





To stop Netdata run: #**systemctl stop netdata**

To start Netdata run: #**systemctl start netdata**

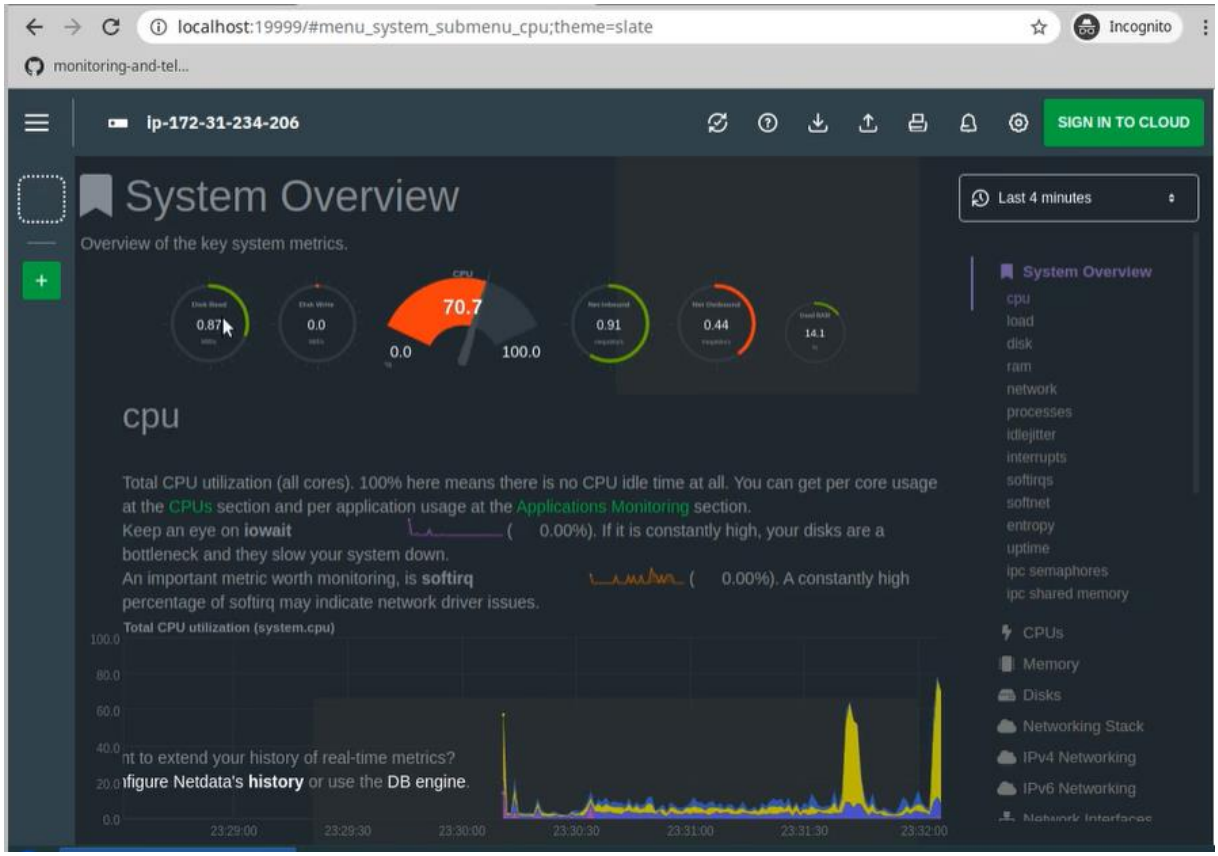
To update netdata execute this script: **/usr/libexec/netdata-updater.sh**

To access to netdata we use this URL: **http://localhost:19999**

2. Accessing Netdata:

1 . Open a web browser on your local computer, for example Chrome or Firefox.

2 . Point it to **http://localhost:19999**

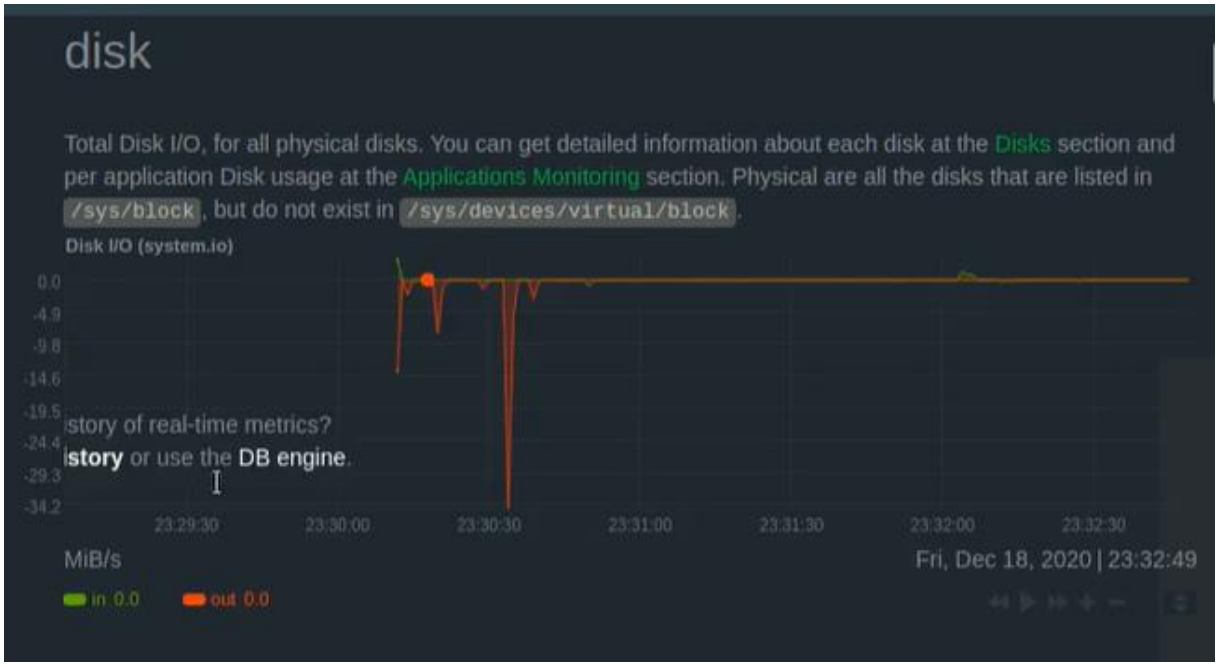


We can see the netdata dashboard is running, It shows how much cpu our system is using, how much memory disc network band with unbound traffic.

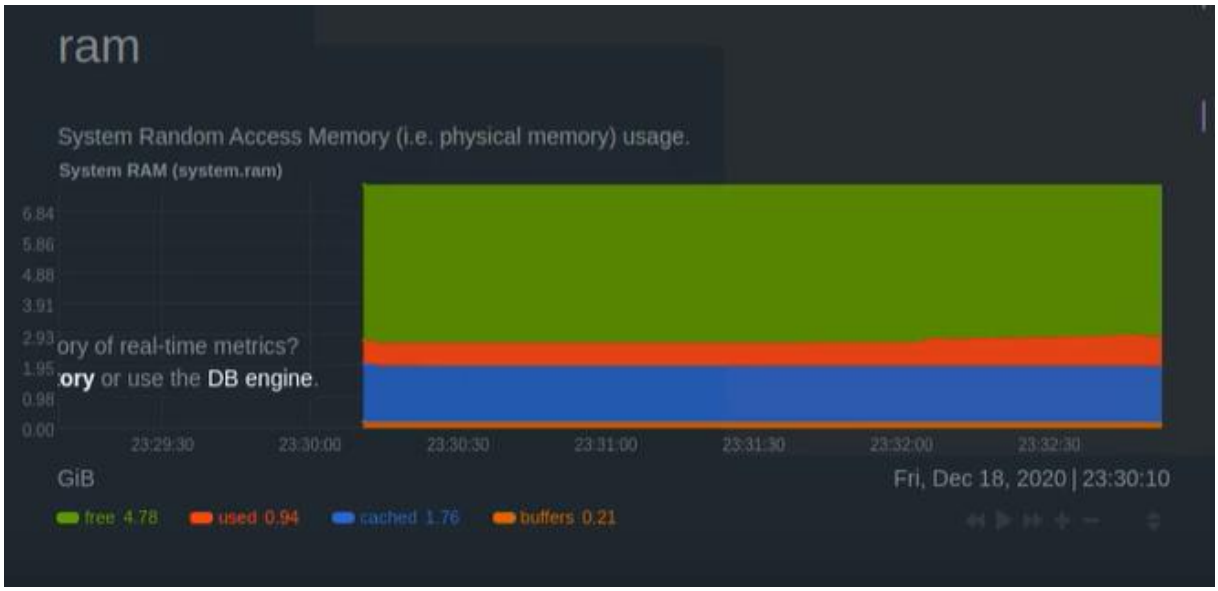
And this the real time graph of the cpu usage:

And the real time graph of the load average time:

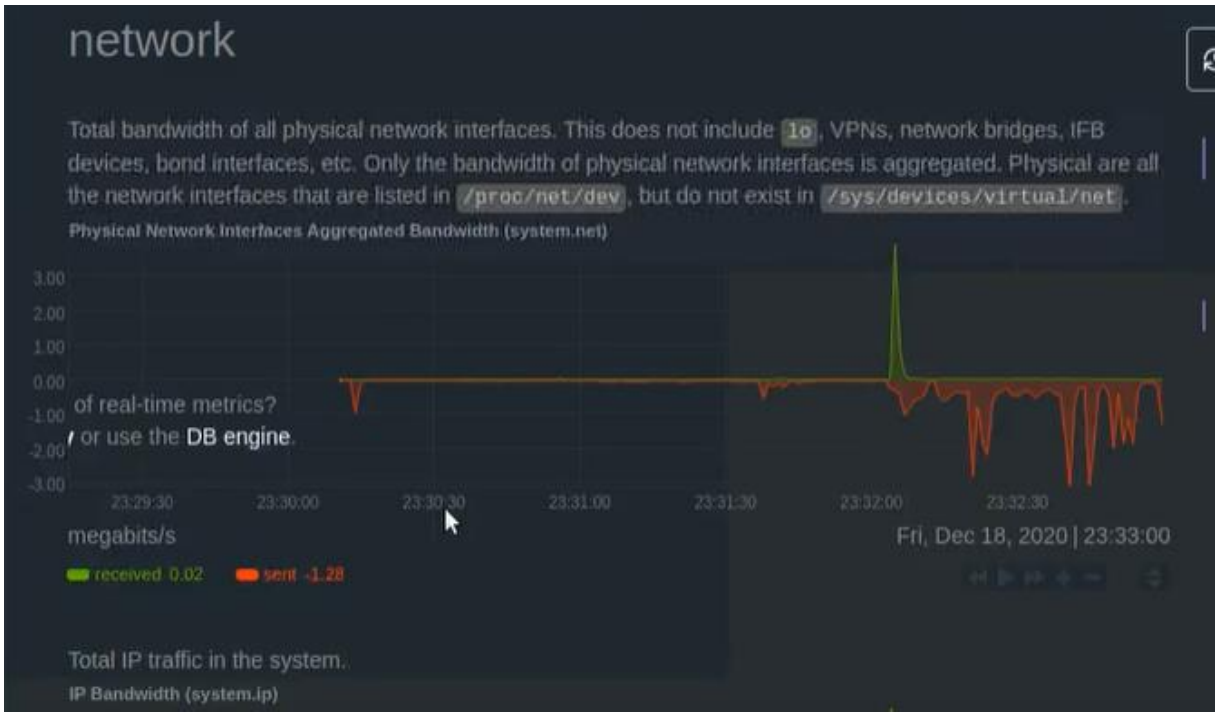
And the real time graph of disk IO:



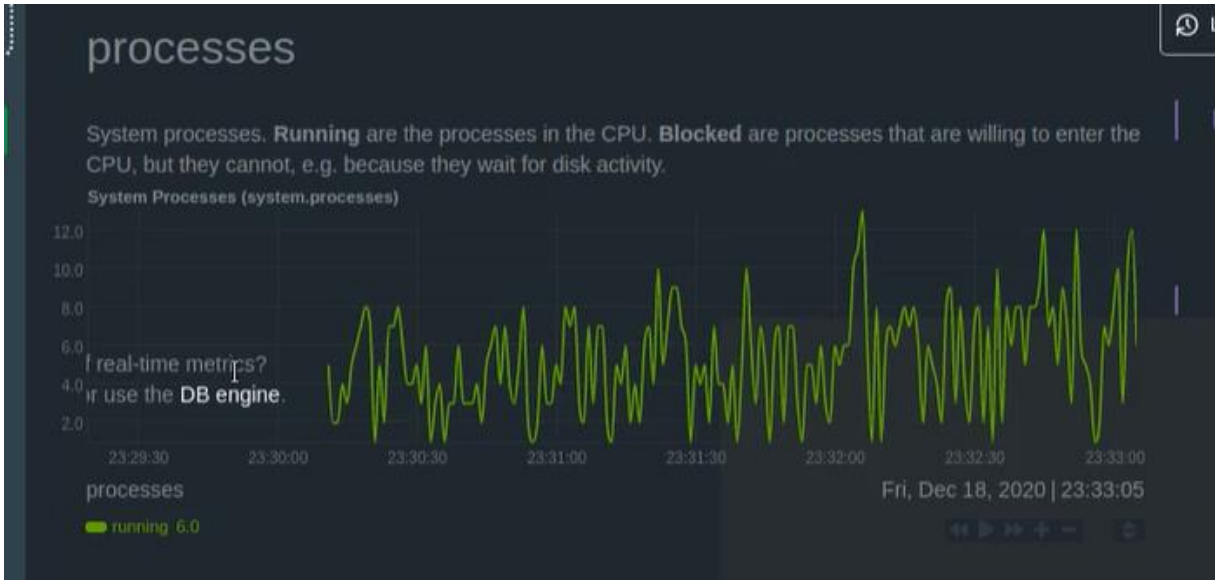
It shows also other important metrics which we can use to monitor, for example, the memory being used:



the network bandwith being consumed:



We have also a presentation of the system processes running:



How much cpu they are blocking.

So, if something is slow, we can detect here whether it's due to the memory of cpu which were spiking, that could be the probable reason of the slowness. So, we can have a good real time view of the entire processes that is going on.

There are also a lot of metrics that we can explore in this system for server Monitoring:

So like we see, Netdata is a simple and open source monitoring tool to use when we want in detail server Monitoring.

3. Basic Docker Monitoring

Monitoring Docker, no matter if used purely or integrated into one of the systems mentioned above, should include aspects of health, performance, and resource usage of the containers. Failures in the daemon directly influence the health of the system as a whole. There are many ways to monitor basic Docker indicators.

The easiest tool to use and monitor Docker containers is **Docker Stats**, which is built into the actual Docker CLI (command line interface). Replicating much of the style known from famous Linux tools like **top** or **htop**, it provides information about container names, CPU, memory and io (block device and network) usage.

#docker stats

So, firstly to launch a docker container, we can just type:

For ngnix container:

#sudo docker run -d ngnix

```
rhyme@ip-172-31-234-206:~$ sudo docker run -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
6ec7b7d162b2: Pull complete
cb420a90068e: Pull complete
2766c0bf2b07: Pull complete
e05167b6a99d: Pull complete
70ac9d795e79: Pull complete
Digest: sha256:4cf620a5c81390ee209398ecc18e5fb9dd0f5155cd82adcbae532fec94006fb9
Status: Downloaded newer image for nginx:latest
e915dc6f7cb932ee7180cef93488829f5bef735fc55319e5252e0505032cba33
```

And, for verify we will type: **#sudo docker ps**

```
rhyme@ip-172-31-234-206:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
e915dc6f7cb9   nginx     "/docker-entrypoint.…"   13 seconds ago   Up 11 seconds   80/tcp    inspiring_dirac
```

That shows the Docker processes, so we can see that the ngnix Containers is up for 11 seconds. Let's now launch more ngnix containers with the same previous command:

```
rhyme@ip-172-31-234-206:~$ sudo docker run -d nginx
4a0c85902cb91f15f733b96528b5923bdd5665ac237cc881e16551269754ee25
rhyme@ip-172-31-234-206:~$ sudo docker run -d nginx
ab216012caa05f38929477c123890d45df8f148c4c9c423564f8c40feee7a419
```

And, then when we verify we see that we have 3 containers with the same image.

```
rhyme@ip-172-31-234-206:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
ab216012caa0   nginx     "/docker-entrypoint.…"   19 seconds ago   Up 18 seconds   80/tcp    gifted_sanderson
4a0c85902cb9   nginx     "/docker-entrypoint.…"   25 seconds ago   Up 24 seconds   80/tcp    hopeful_nightingal
e 1
e915dc6f7cb9   nginx     "/docker-entrypoint.…"   52 seconds ago   Up 50 seconds   80/tcp    inspiring_dirac
```

So, now when we run the command: #docker stats we got:

```
rhyme@ip-172-31-234-206:~$ sudo docker stats
```

```
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT   MEM %     NET I/O       BLOCK I/O
PIDS
ab216012caa0   gifted_sanderson    0.00%     2.5MiB / 7.69GiB    0.03%     5.11kB / 0B   0B / 0B
2
4a0c85902cb9   hopeful_nightingale 0.00%     2.508MiB / 7.69GiB  0.03%     5.76kB / 0B   0B / 0B
2
e915dc6f7cb9   inspiring_dirac     0.00%     2.551MiB / 7.69GiB  0.03%     11.3kB / 0B   0B / 0B
2
```

Like we see, the command display the statistics of Docker Containers. We can see: the containers ID, The percentage of CPU has being used, the memory, the maximum available memory for the containers to use, the network IO (input/output) and the block IO in the disc.

Now moving on a graphical tool for Monitoring Docker, which called '**cAdvisor**'

cAdvisor runtime information together with other important metrics can be picked up with cAdvisor. cAdvisor can be launched as a docker container to monitor other docker containers. It can be launched via the following command:

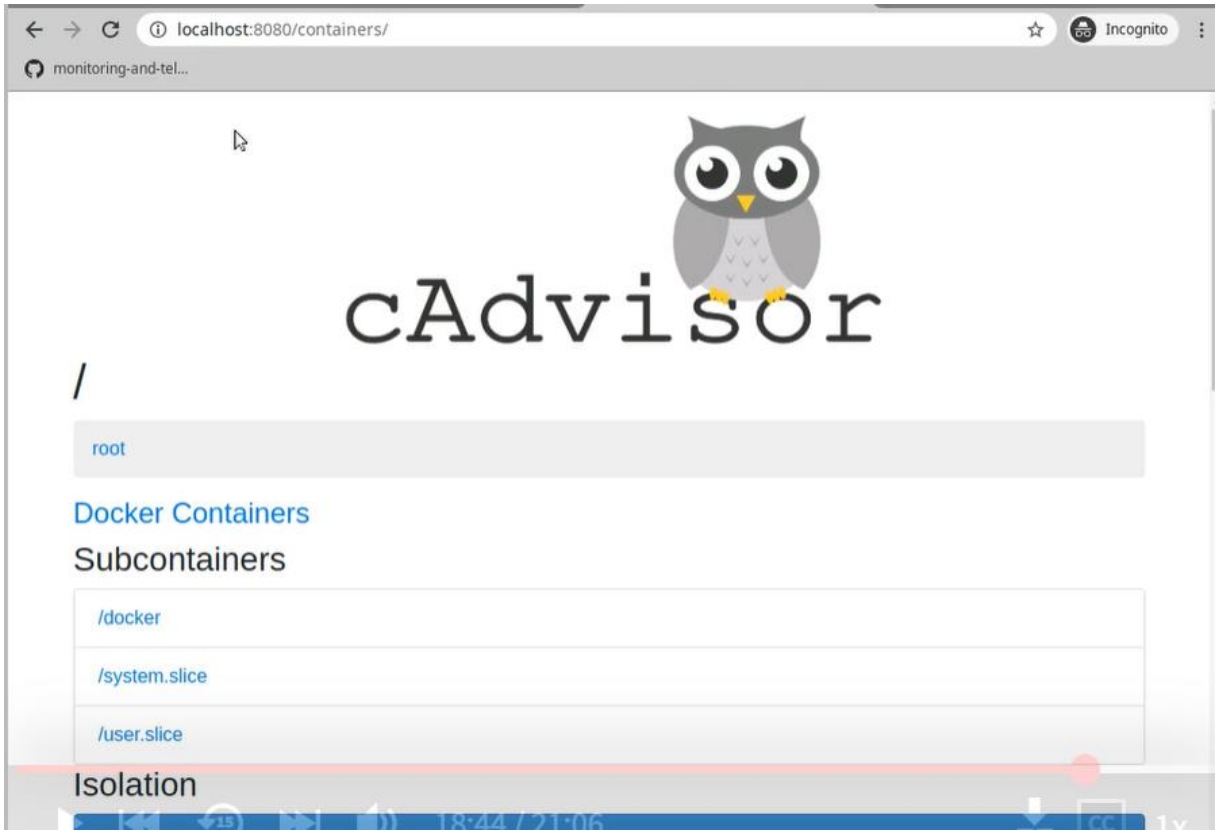
```
VERSION=v0.36.0 # use the latest release version from
https://github.com/google/cadvisor/releases
sudo docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:ro \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --volume=/dev/disk/:/dev/disk:ro \
  --publish=8080:8080 \
  --detach=true \
  --name=cadvisor \
  --privileged \
  --device=/dev/kmsg \
  gcr.io/cadvisor/cadvisor:$VERSION
```

```
rhyme@ip-172-31-234-206:~$ VERSION=v0.36.0 # use the latest release version from https://github.com/google/cadvisor/releases
rhyme@ip-172-31-234-206:~$ sudo docker run \
>   --volume=/:/rootfs:ro \
>   --volume=/var/run:/var/run:ro \
>   --volume=/sys:/sys:ro \
>   --volume=/var/lib/docker/:/var/lib/docker:ro \
>   --volume=/dev/disk/:/dev/disk:ro \
>   --publish=8080:8080 \
>   --detach=true \
>   --name=cadvisor \
>   --privileged \
>   --device=/dev/kmsg \
>   gcr.io/cadvisor/cadvisor:$VERSION
24777f8fc15bd79a6a547d2723349657b8a7204ba6bbb0cfcf2224b2d7aeb844
```

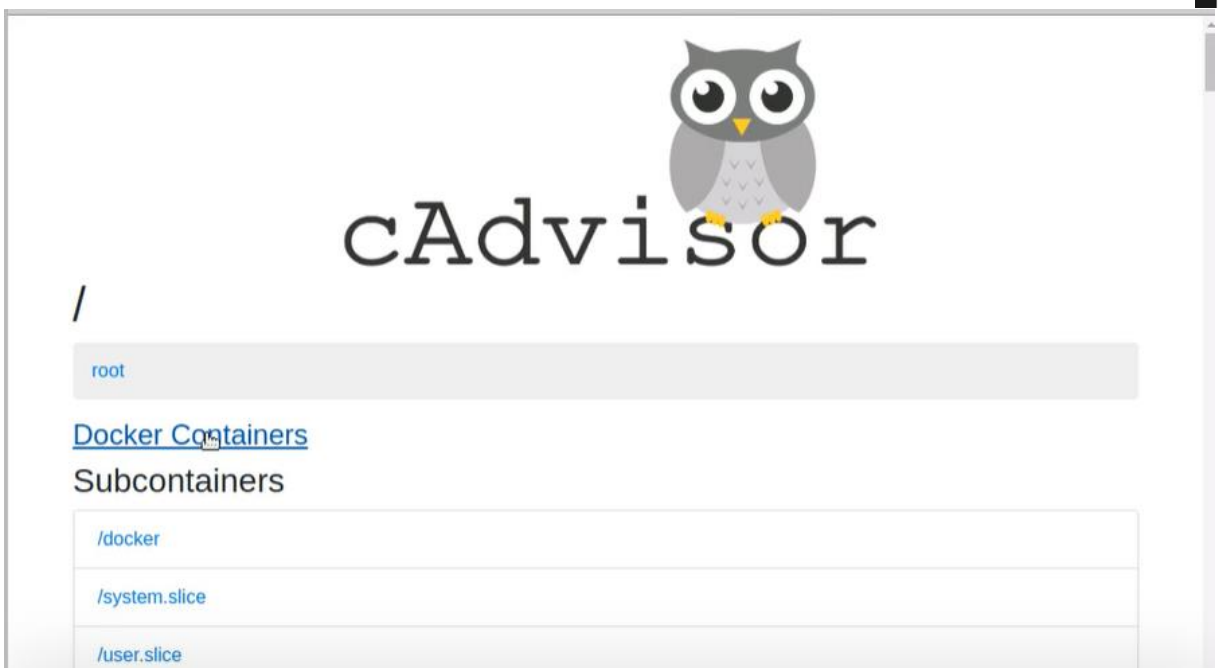
On typing #sudo docker ps we can see the container was created and it is in a starting state (healh:starting):

```
rhyme@ip-172-31-234-206:~$ sudo docker ps
CONTAINER ID        IMAGE                              COMMAND                  CREATED             STATUS                            PORTS
                    NAMES
24777f8fc15b        gcr.io/cadvisor/cadvisor:v0.36.0   "/usr/bin/cadvisor -…"   10 seconds ago      Up 10 seconds (health: starting)  0.0.0.0:
8080->8080/tcp      cadvisor
ab216012caa0        nginx                              "/docker-entrypoint.…"   4 minutes ago       Up 4 minutes                      80/tcp
                    gifted_sanderson
4a0c85902cb9        nginx                              "/docker-entrypoint.…"   4 minutes ago       Up 4 minutes                      80/tcp
                    hopeful_nightingale
e915dc6f7cb9        nginx                              "/docker-entrypoint.…"   4 minutes ago       Up 4 minutes                      80/tcp
                    inspiring_dirac
```

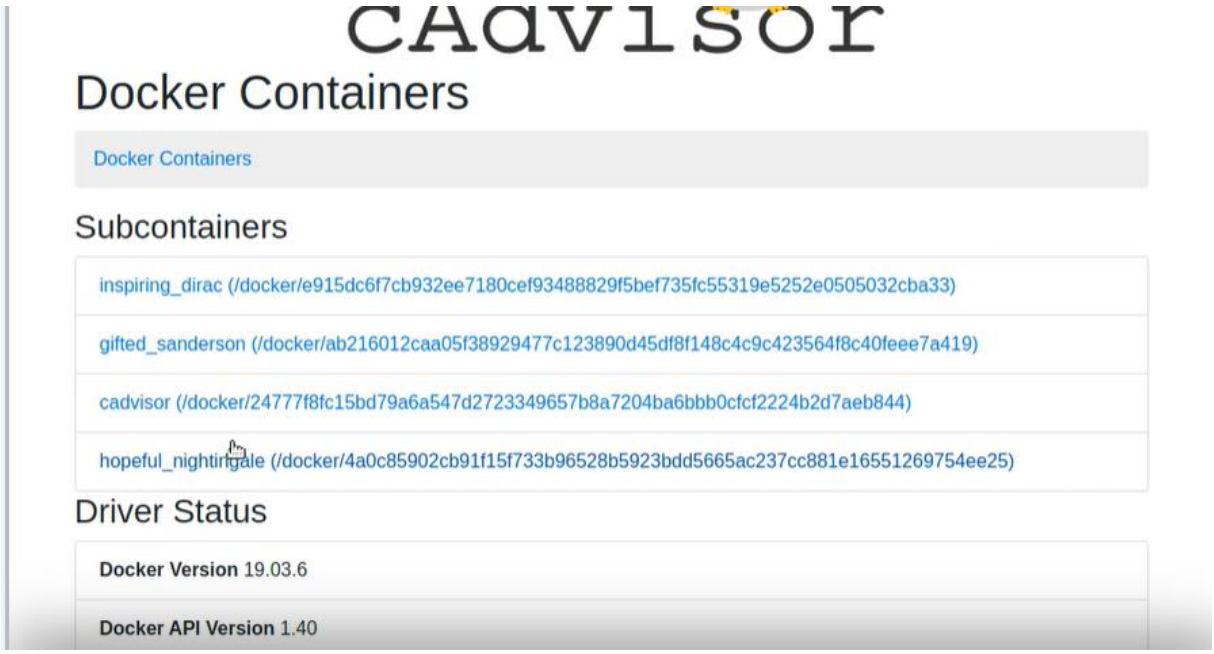
cAdvisor is now running (in the background) on http://localhost:8080. The setup includes directories with Docker state cAdvisor needs to observe.



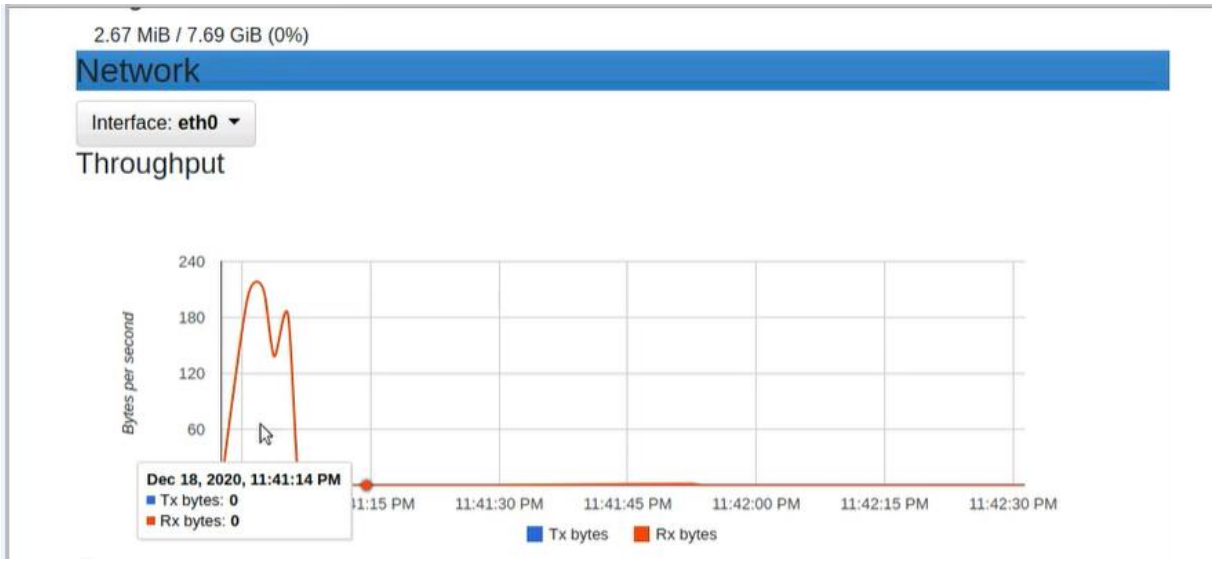
Like we see the cAdvisor is up, and if we click on Docker Containers:



We can see the four containers is running:



And for seeing the state of each container, we can click on it, and the graphs appearing:



Task-4: Perform Application Monitoring & Telemetry with Prometheus & Grafana - Part I

In this task, we will see how to monitor applications, such an applications which being deployed as microservices, and these applications need to be monitored based on the resources, they use and how performant and efficient they are.

For example, we have a microservices which is running on java, for instance Spring Boot java framework, it would be using JVM. So, to get the metrics from a JVM Stack and other to do that to the user tool called '**Prometheus**'.

Prometheus is integrated into the Spring boot java application and it gets all the metrics from the application itself while it is running and passes it on to a dashboard software which is called '**Grafana**'.

So that allows us to see whether the requests were successful or got an internal server error of any sort of other errors. So, that way, checking the health of for Microservices on real-time becomes easy.

In a distributed landscape where we are working with microservices, serverless applications, or just event-driven architecture as a whole, observability, which comprises monitoring, logging, tracing, and alerting, is an important architectural concern.

There are a few reasons why we want visibility in our highly distributed systems:

- Issues will occur, even when our best employees have built it.
- Distributed systems generate distributed failures, which can be devastating when we are not prepared in advance.
- Reveal mistakes early, which is great for improvement and learning.
- It keeps us accountable.

Reduce the mean time to resolution (MTTR).

1. Prometheus:

1.1 What is Prometheus?

Prometheus, originally developed by SoundCloud is an open source and community-driven project that graduated from the Cloud Native Computing Foundation. It can aggregate data from almost everything:

- Microservices
- Multiple languages
- Linux servers
- Windows servers

1.2 Why do we need Prometheus?

In our modern times of microservices, DevOps is becoming more and more complex and therefore needs automation.

It works very well with any language.

We have hundreds of processes running over multiple servers, and they are all interconnected.

If we would not monitor these services then we have no clue about what is happening on hardware level or application level.

There are many things which we want to be notified about, like:

- Errors
- Response latency
- System overload
- Resources

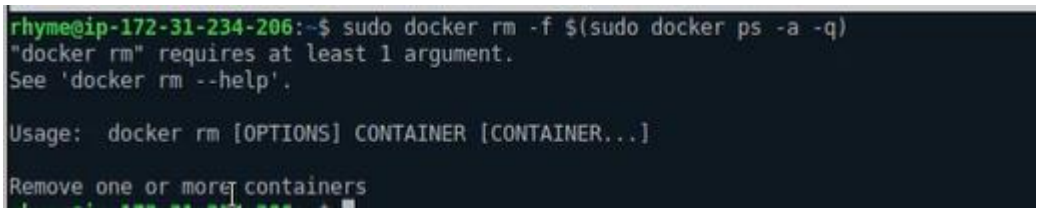
When we are working with so many moving pieces, we want to be able to quickly identify a problem when something goes wrong inside one of our services.

If we wouldn't monitor, it could be very time-consuming, since we have no idea where to look.

Grafana is a software which allows us to create our own customer dashboards. And, these dashboards allow us to see a visualized graphs. Grafana is a metric analytics and visualization application and Grafana is open-source. So, Prometheus and Grafana form a very good pair to get data and display data. Spring Boot contains a exporter called Microservices, we will be enable that and this will grab the data from the service and pass it on the Prometheus which will eventually pass it on to Grafana.

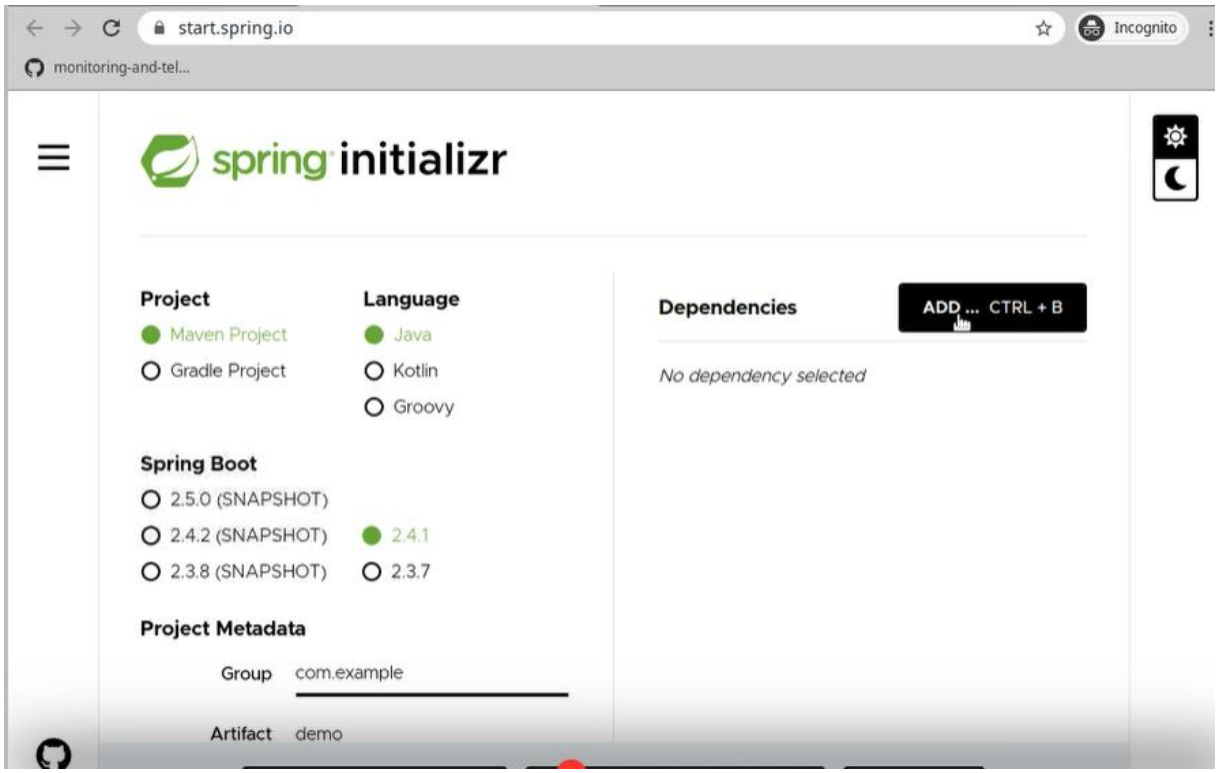
So, first of all we gonna build a Spring Boot microservice application and run it on a certain port.

We gonna clean Docker Containers because there are some ones running due to the processes of the previous task, we gonna type this command: **#sudo docker rm -f $(sudo docker ps -a -q)**

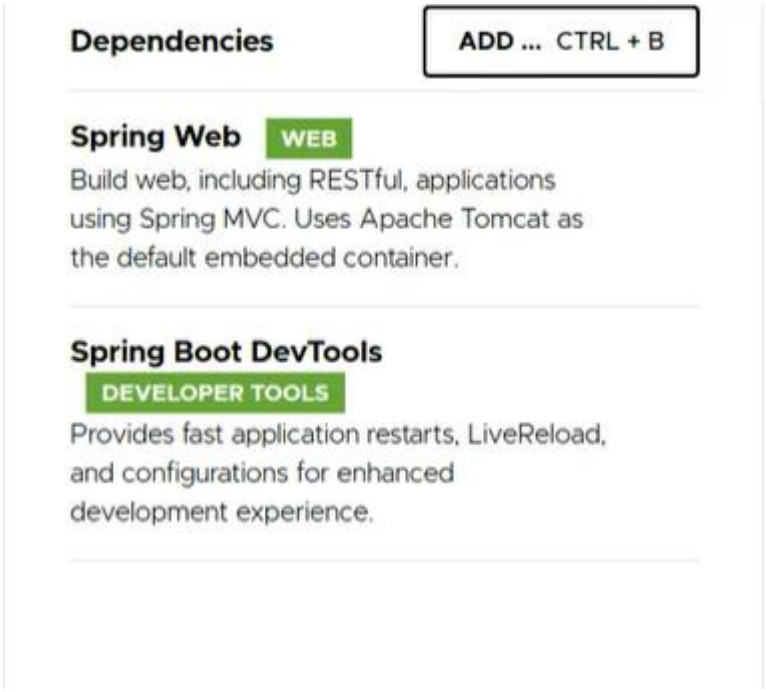


Like we see, we don’t have any containers that’s why we got an exception.

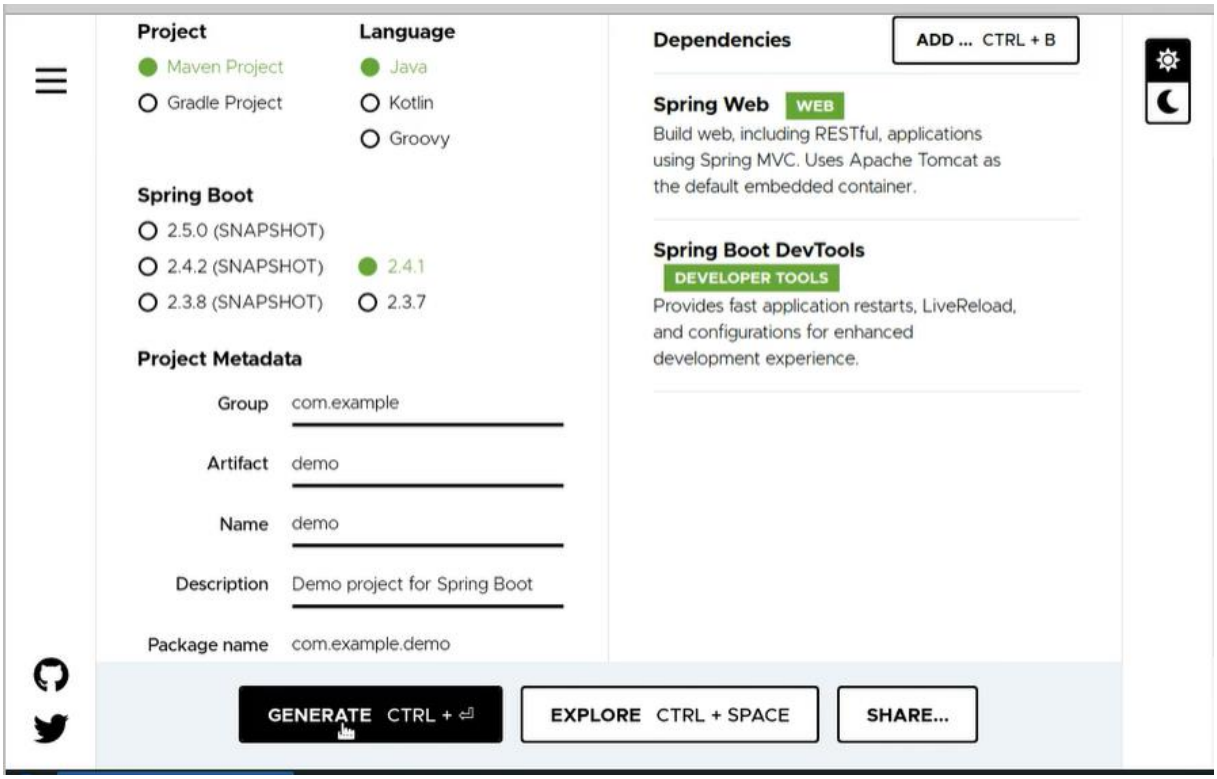
And now, Let’s install Spring Boot:



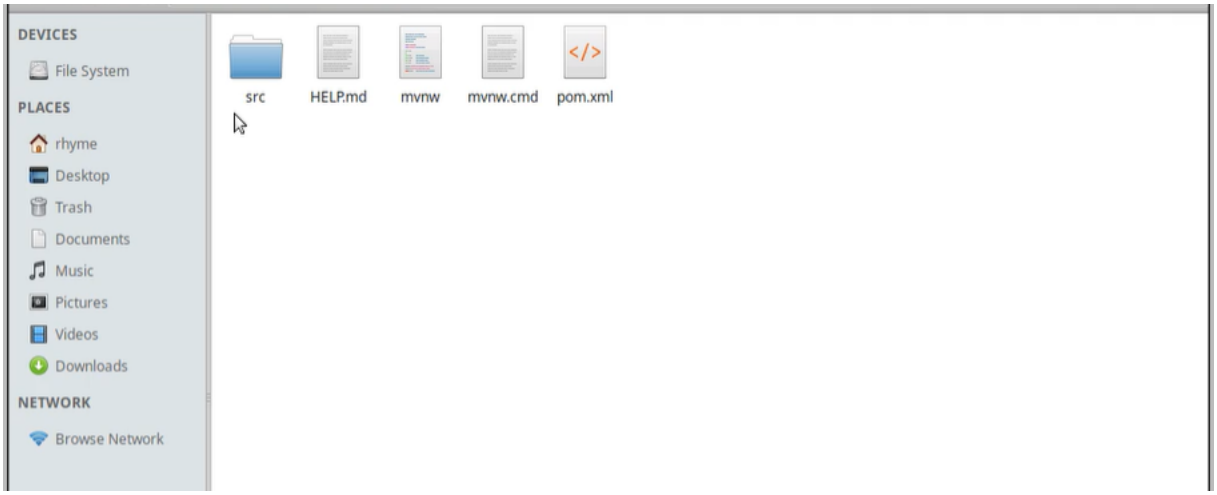
And we gonna add some dependencies:



Now, let’s click on GENERATE:



So, we will see there is a demo.zip file that has been downloaded. We gonna to downloads folder, and unzipping this file, and this is the file content:



Now, we gonna open the **pom.xml** file and add some dependencies on to be able to monitor this application and have Prometheus as an actuator here.

- Add dependency for Actuator:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
```

- Add dependency for Micrometer:

To monitor our Spring Boot application we will be using an exporter named Micrometer.

Micrometer is an open-source project and provides a metric facade that exposes metric data in a vendor-neutral format which Prometheus can ingest.

Micrometer provides a simple facade over the instrumentation clients for the most popular monitoring systems, allowing you to instrument your JVM-based application code without vendor lock-in. Think SLF4J, but for metrics.

Micrometer is not part of the Spring ecosystem and needs to be added as a dependency. In our demo application we will add this to our pom.xml file.

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
    <version>1.5.5</version>
</dependency>
```

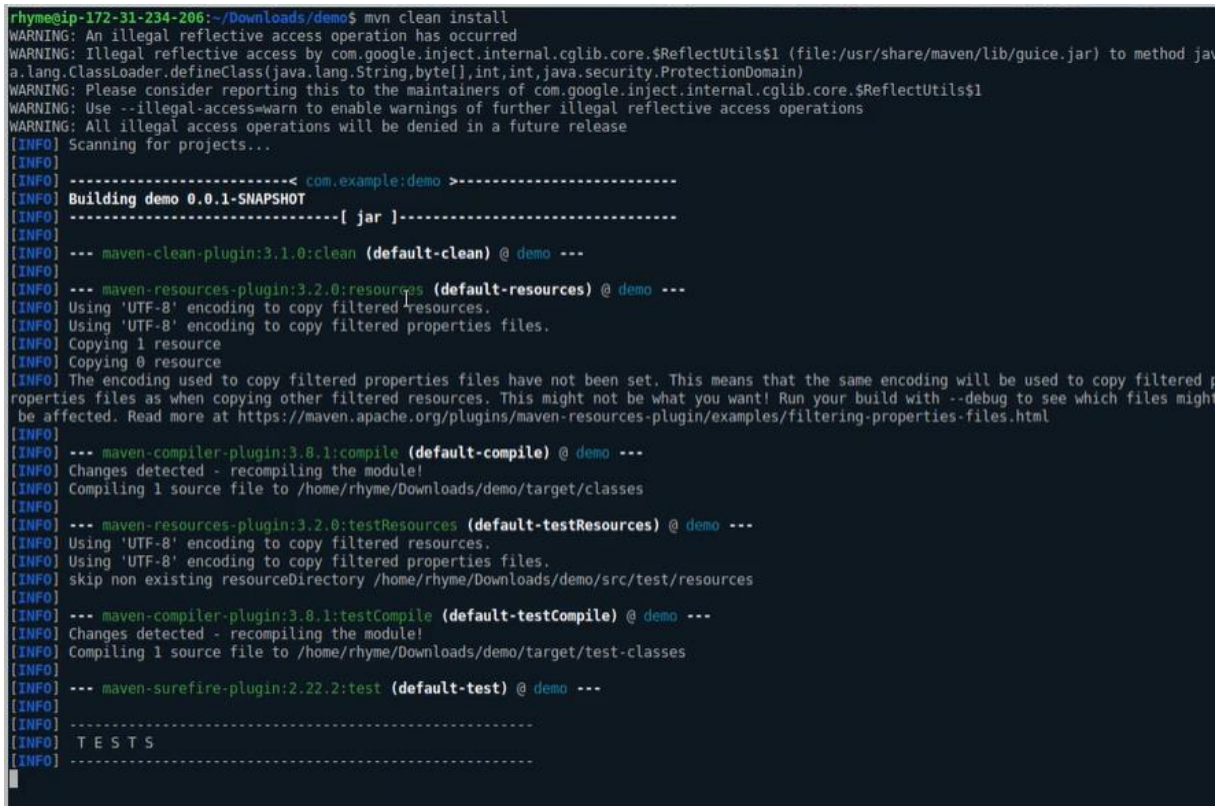
- Add Management endpoint to be exposed: in /src/main/resources/**application.properties:**

```
application.properties
home > rhyme > Downloads > demo > src > main > resources > application.properties
1  management.endpoints.web.exposure.include=prometheus
2  management.endpoint.health.show-details=always
3  management.metrics.tags.application= MonitoringSpringDemoProject
4
```

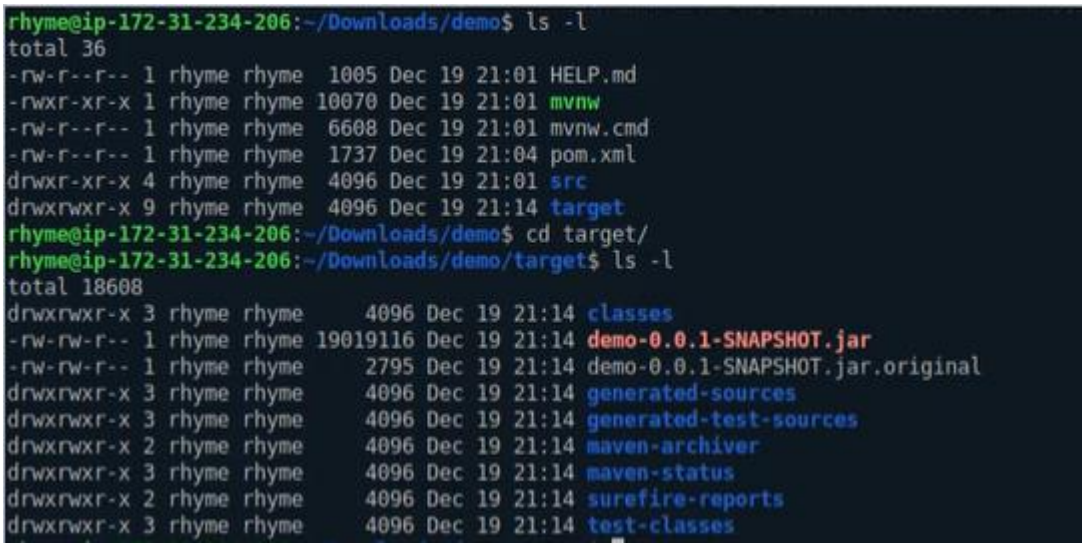
We should now installing maven with this command: **#sudo apt-get install maven**

```
rhyme@ip-172-31-234-206:~/Downloads/demo$ sudo apt-get install maven
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  gyp libaio1 libjemalloc1 libjs-async libjs-inherits libjs-node-uuid libjs-underscore libuv1-dev mariadb-client-core-10.1 mariadb-common
  node-abbrev node-ansi node-ansi-color-table node-archy node-async node-balanced-match node-block-stream node-brace-expansion
  node-builtin-modules node-combined-stream node-concat-map node-cookie-jar node-delayed-stream node-forever-agent node-form-data
  node-fs.realpath node-fstream node-fstream-ignore node-github-url-from-git node-glob node-graceful-fs node-hosted-git-info node-inflight
  node-inherits node-ini node-is-builtin-module node-isexe node-json-stringify-safe node-lockfile node-lru-cache node-mime node-minimatch
  node-mkdirp node-mute-stream node-node-uuid node-nopt node-normalize-package-data node-npmlog node-once node-osenv node-path-is-absolute
  node-pseudomap node-qs node-read node-read-package-json node-request node-retry node-rimraf node-semver node-sha node-slide
  node-spdx-correct node-spdx-expression-parse node-spdx-license-ids node-tar node-tunnel-agent node-underscore
  node-validate-npm-package-license node-which node-wrappy node-yallist
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  maven
0 upgraded, 1 newly installed, 0 to remove and 304 not upgraded.
Need to get 22.4 kB of archives.
After this operation, 129 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 maven all 3.6.0-1~18.04.1 [22.4 kB]
Fetched 22.4 kB in 0s (2028 kB/s)
Selecting previously unselected package maven.
(Reading database ... 224658 files and directories currently installed.)
```

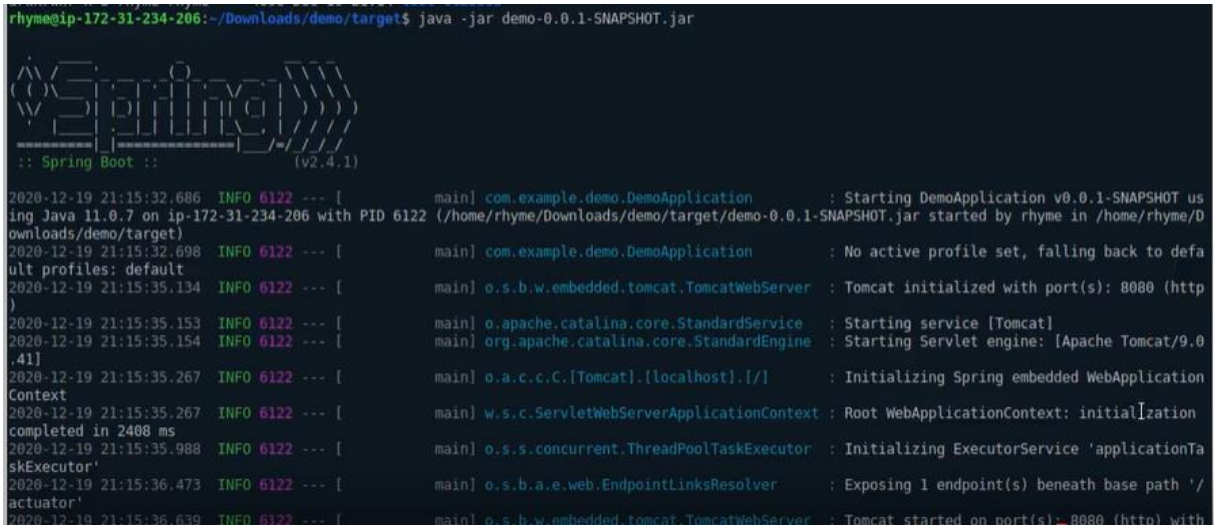
Now in the **/directory/demo** path we gonna compile the java code in jar file, for that we gonna use the maven command: #**mvn clean install**



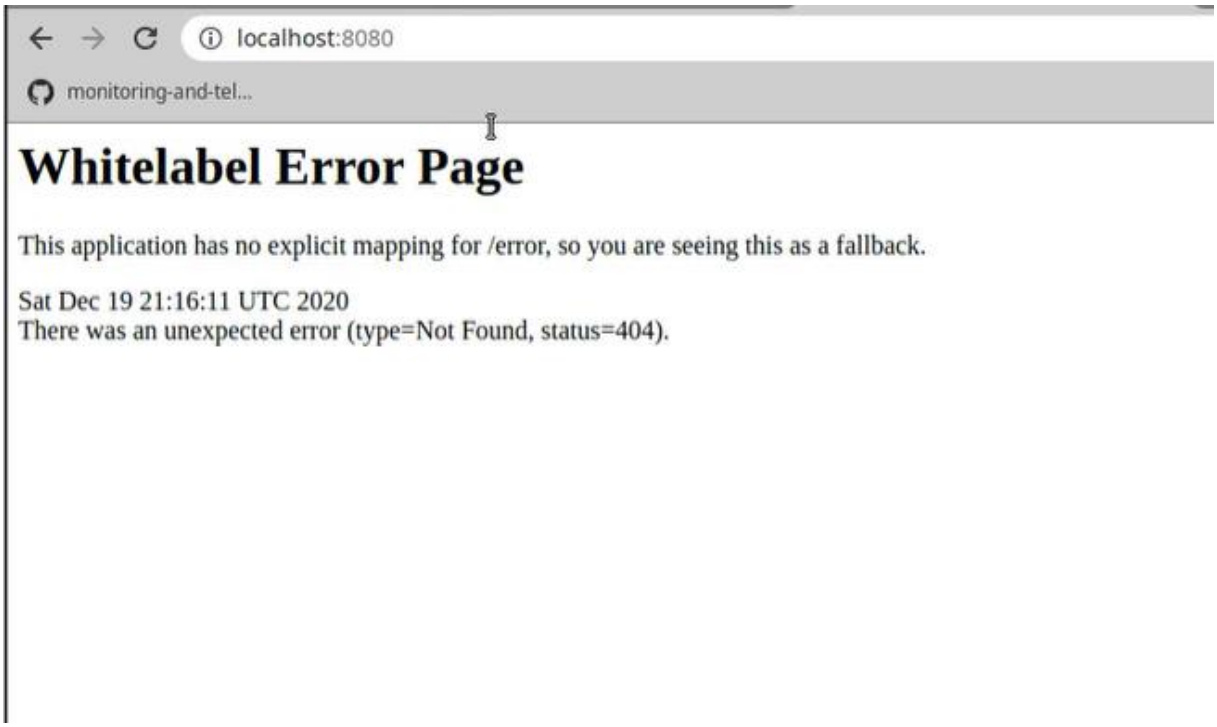
Now, if we type **#ls -l** command we can see a new file called target was created, and on going ahead of this file we can see a jar file called **demo-0.0.1-SNAPSHOT.jar**, this is the Java archive file which we can execute**.**



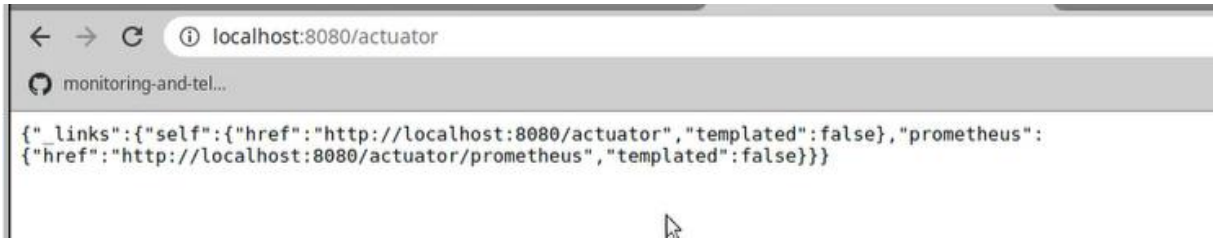
So, let's execute this file with this command: #**java -jar demo-0.0.1-SNAPSHOT.jar**



Ike we see on running the command, here it start launching the Spring Boot application and it started Tomcat on Port 8080 and we are up. If we are going to the browser, and type the URL address localhost:8080, we can see the application running:

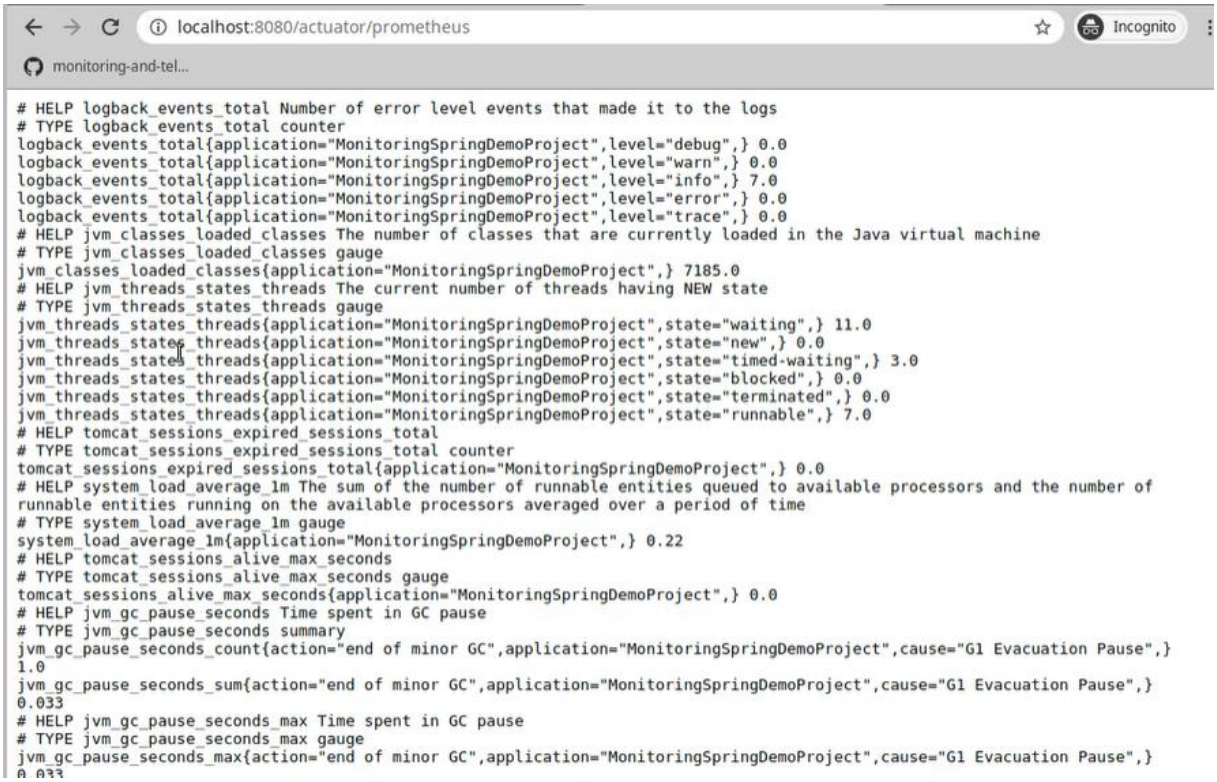


We now that The Prometheus agent that we set up **/actuator**:



so here we can see that there are some endpoints available that is actuator and Prometheus.

Now, on going to **localhost:8080/actuator/Prometheus**:



In this file we can see that there is a “HELP” comment which describes what the metric is, and we have a “TYPE” which can be one of four metric-types:

- Counter: how many times X happened (exceptions)
- Gauge: what is the current value of X now? (disk usage, cpu etc)
- Histogram: how long or how big?
- Summary: similar to histogram it monitors request durations and response sizes

we can see the various endpoints that is being monitored by Prometheus and the data is getting the heap sized JVM garbage collection parameters, how much memory is being used, how much threads are being used by java… and all possible information about the application.

But is this a very convenient for someone to monitor?

Say we have 100 Microservices and get metrics from Prometheus in this manner, it is humanly impossible to go ahead and understand these information and visualize them .. so in order to make it user friendly or human friendly to monitor, we use Grafana to make it a graphical view for Monitoring applications.

So, the first now would be to pull Prometheus Docker image, we gonna run Prometheus via a Docker image by the command: #**sudo docker pull prom/prometheus**

This command will pull the latest Prometheus image from the system.

```
rhyme@ip-172-31-234-206:~$ sudo docker pull prom/prometheus
Using default tag: latest
latest: Pulling from prom/prometheus
Digest: sha256:0eac377a90d361be9da35b469def699bcd5bb26eab8a6e9068516a9910717d58
Status: Image is up to date for prom/prometheus:latest
docker.io/prom/prometheus:latest
```

After we downloaded the image, we need to configure **our prometheus.yml** file. Since we want to demonstrate how to monitor a Spring Boot application, as well as Prometheus itself, it should look like this:

Now the image was downloaded. And now we gonna create the Prometheus.yml file. For that we type **#code /tmp/Prometheus.yml**

```
rhyme@ip-172-31-234-206:~$ code /tmp/prometheus.yml
```

prometheus.yml

```
1  global:
2      scrape_interval:     15s # By default, scrape targets every 15 seconds.
3
4  rule_files:
5    # - "first.rules"
6    # - "second.rules"
7
8  scrape_configs:
9  - job_name: 'prometheus'
10   scrape_interval: 5s
11
12   static_configs:
13     - targets: ['localhost:9090']
14
15 - job_name: 'spring-actuator'
16   metrics_path: '/actuator/prometheus'
17   scrape_interval: 5s
18
19     #target end point. We are using the Docker, so local host will not work. You can change it with
20     #localhost if not using the Docker.
21   static_configs:
22     - targets: ['192.168.0.9:8080']
```

In this configuration file we declare a few things:

1. global configs, like how often it will scrape its targets.
2. we can declare rule files, so when we meet a certain condition, we get an alert.
3. which services it needs to monitor.

In this example you can see that Prometheus will monitor two things:

- Our Spring Boot application
- Its own health

ALERT

With Prometheus, we have the possibility to get notified when metrics have reached a certain point, which we can declare in the .rules files. Prometheus has a component which is called the "Alertmanager", and it can send notifications over various channels like emails, Slack, PagerDuty, etc.

QUERYING OUR DATA

Since Prometheus saves all our data in a time series database, which is located on disk in a custom timeseries format, we need to use PromQL query language, if we want to query this database.

We can do this via the Prometheus WebUI, or we can use some more powerful visualization tools like Grafana.

Afterwards we can run the Prometheus image by running the following command:

#Sudo docker run -d -p 9000:9000 -V /tmp/prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus

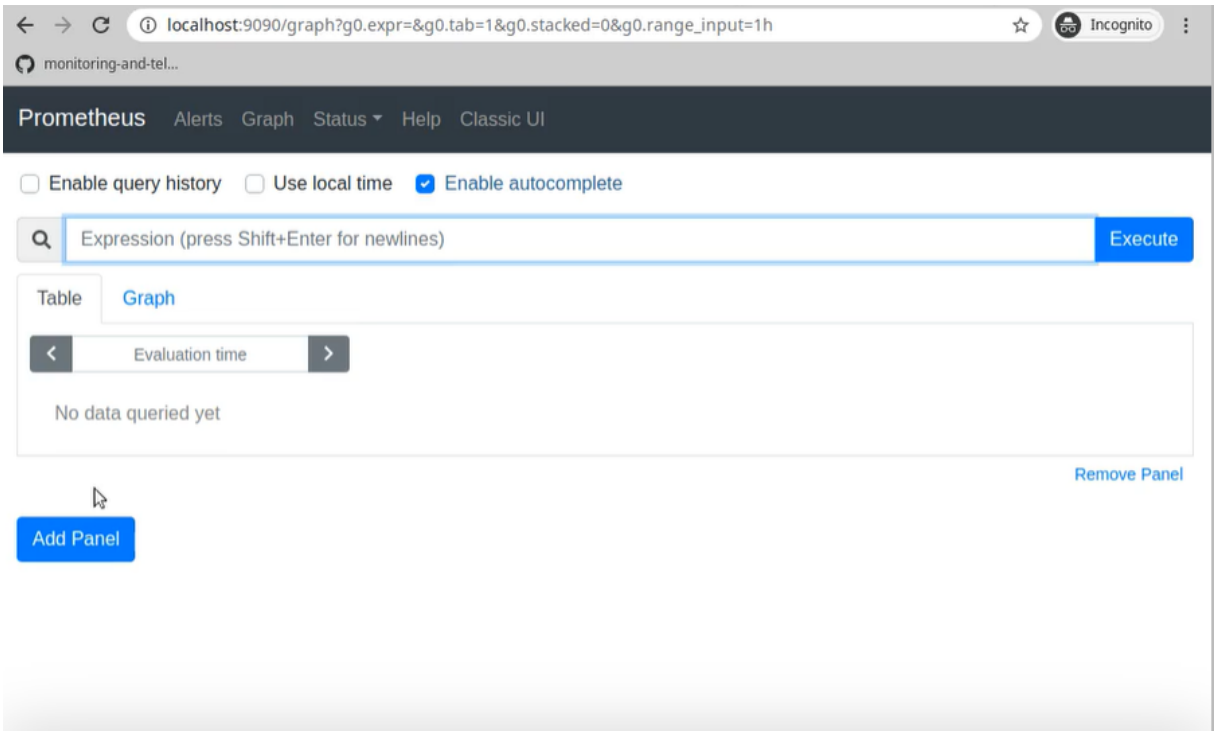
```
rhyme@ip-172-31-234-206:~$ sudo docker run -d -p 9090:9090 -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus
6a9b4173c7368a1287499ea85874999e7c5ccda6ac7ec8f3f725b575d39a1fd5
```

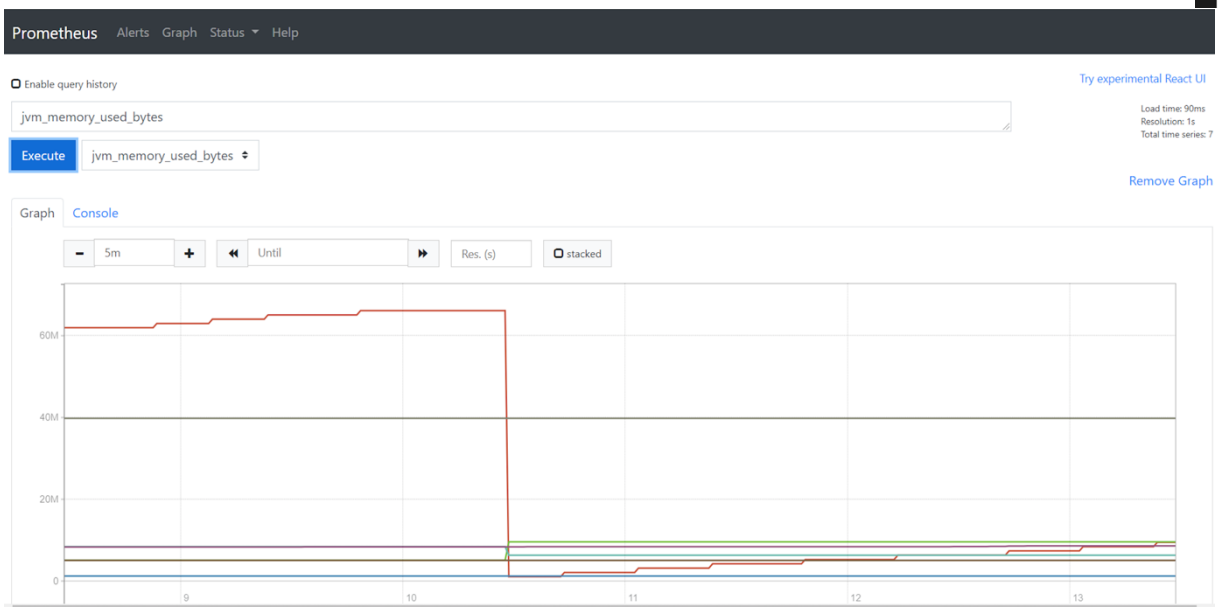
On running the command**: #sudo docker ps**, we got that the container is running on 9090 port

```
rhyme@ip-172-31-234-206:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
6a9b4173c736        prom/prometheus     "/bin/prometheus --c…"   8 seconds ago       Up 7 seconds        0.0.0.0:9090->9090/tcp   frosty_wiles
```

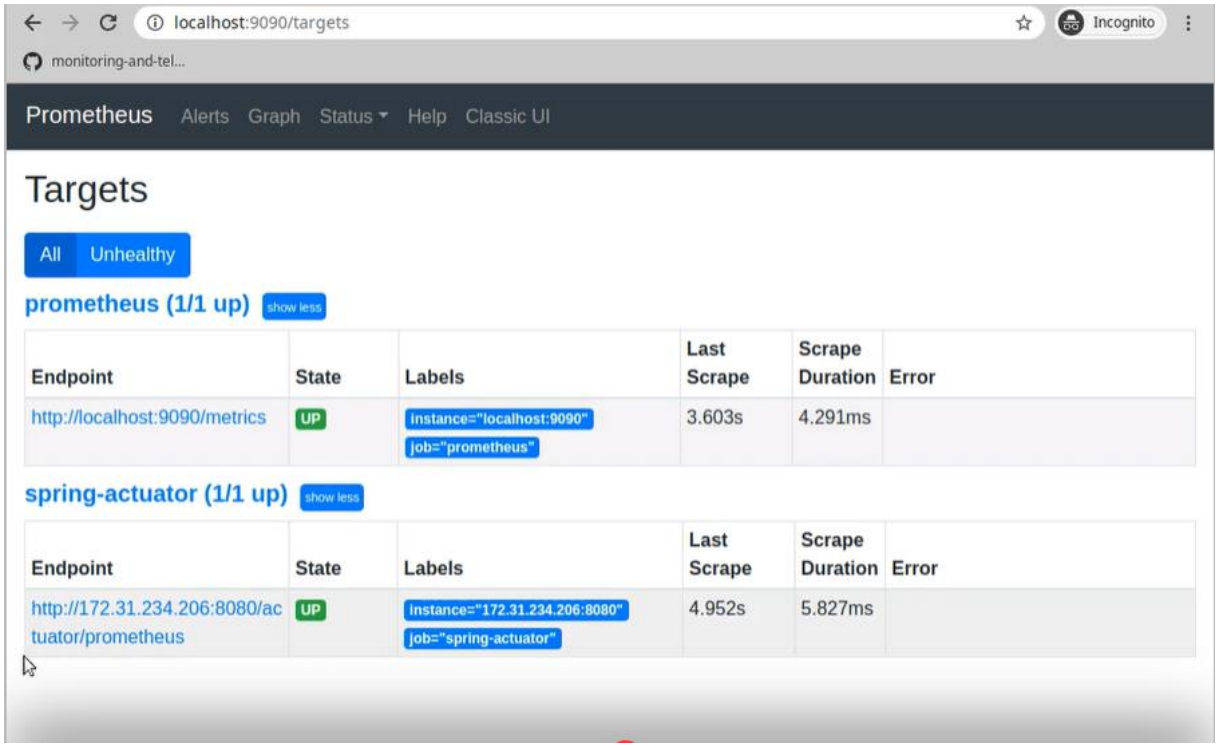
We mount the prometheus.yml config file into the Prometheus image and expose port 9090, to the outside of Docker.

When this is up and running we can access the Prometheus WebUI on localhost:9090.



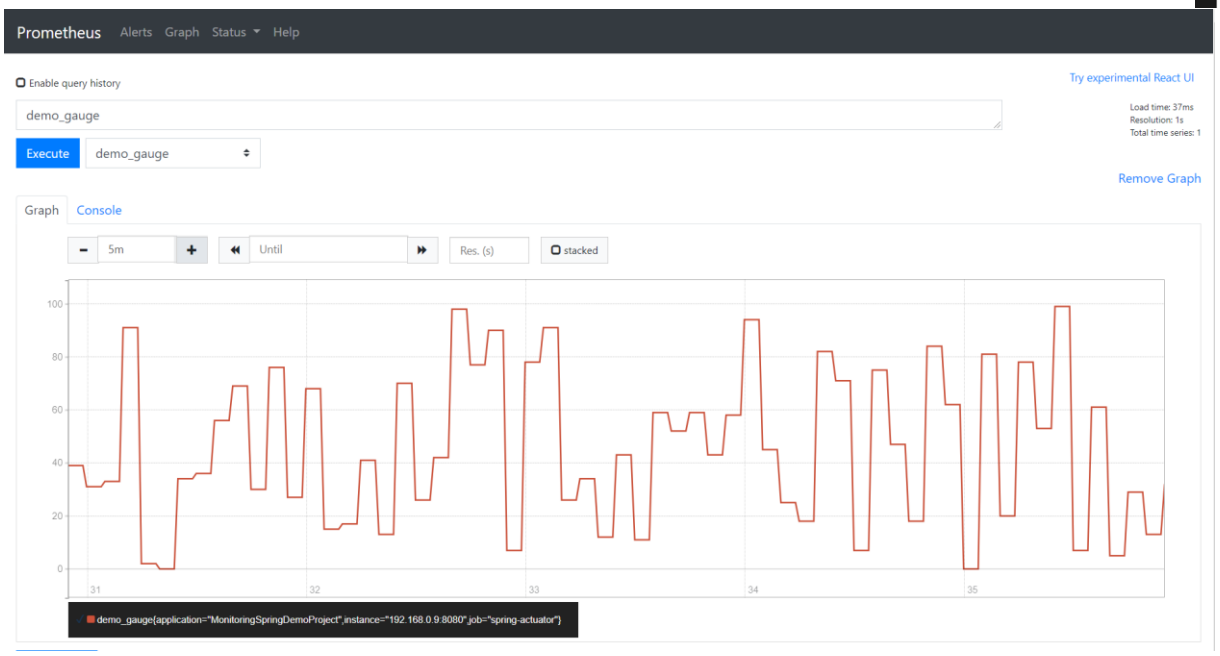


When we navigate to Status > Targets, we can check if our connections are up and are correctly configured by typing in the browser **localhost:9090/targets**



And, like we can see, we have metrics and the spring boot application is up.

Yet again, we can check our custom metrics in the Prometheus UI, by selecting the demo_gauge and inspecting our graph.



2. Grafana

2.1 What is Grafana?

Grafana is an open-source metric analytics & visualization application.

It is used for visualizing time series data for infrastructure and application analytics.

It is also a web application which can be deployed anywhere users want.

It can target a data source from Prometheus and use its customizable panels to give users powerful visualization of the data from any infrastructure under management.

2.2 Why Grafana?

One of the significant advantages of Grafana are its customization possibilities.

It's effortless to customize the visualization for vast amounts of data.

We can choose a linear graph, a single number panel, a gauge, a table, or a heatmap to display our data.

We can also sort all our data with various labels so data with different labels will go to different panels.

Last but not least, there are a ton of premade dashboard-templates ready to be imported, so we don't have to create everything manually.

2.3 Installing Grafana:

To run Grafana we will use the same approach as with Prometheus.

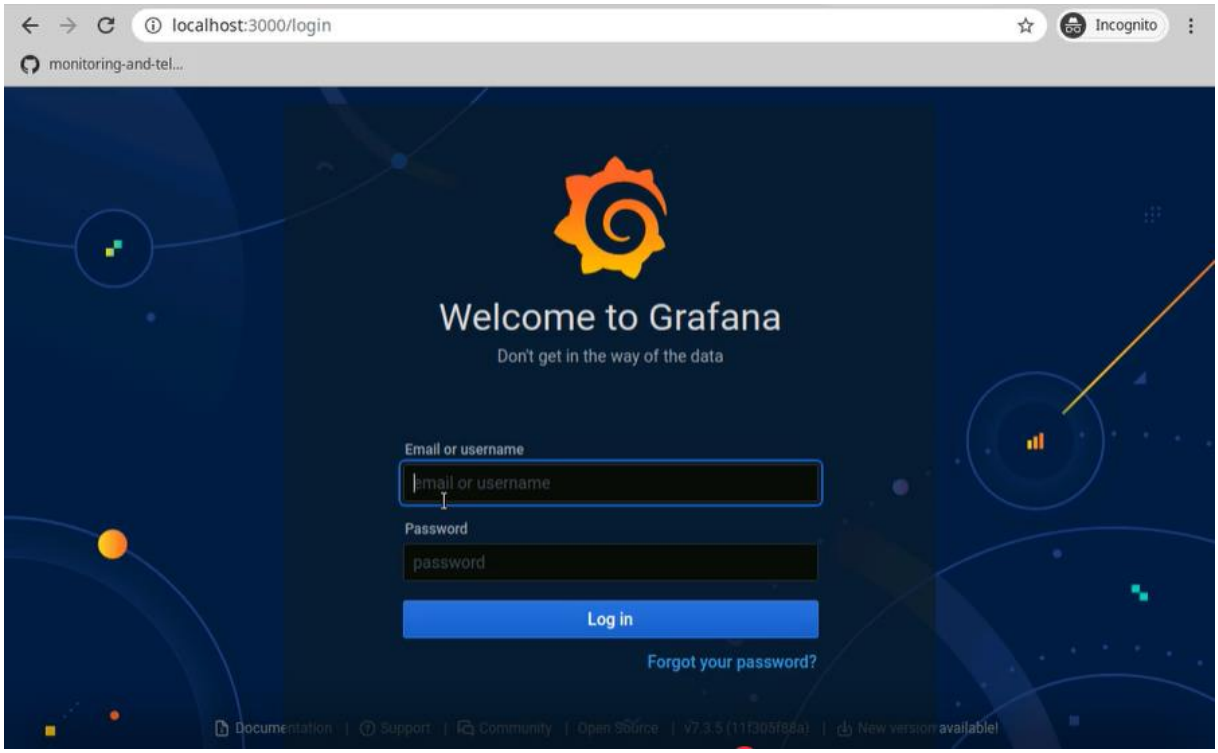
We download and run the image from Docker Hub by typing this command: **#sudo docker run -d -p 3000:3000 grafana/grafana**

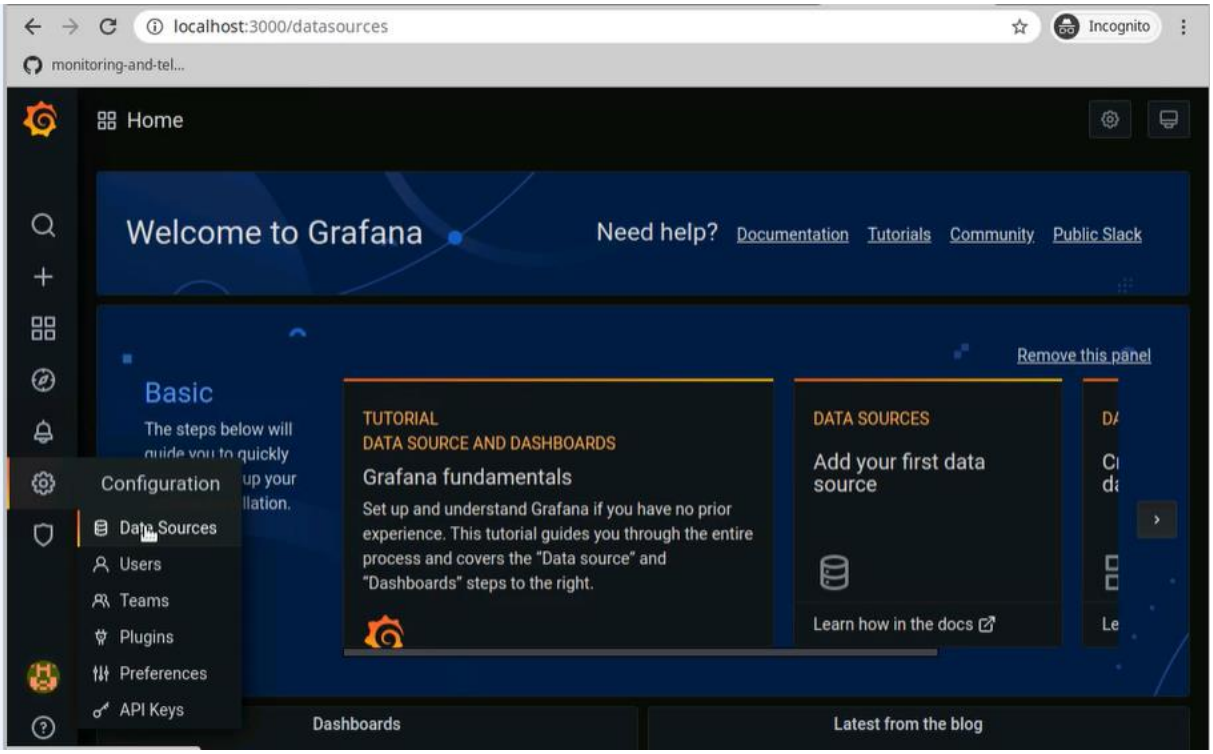
```
rhyme@ip-172-31-234-206:~$ sudo docker run -d -p 3000:3000 grafana/grafana
6c00bb8bd54667b8cb4f6765048afadaad8102f184abc143381ea250052d0bef
```

So here the grafana container was launched.

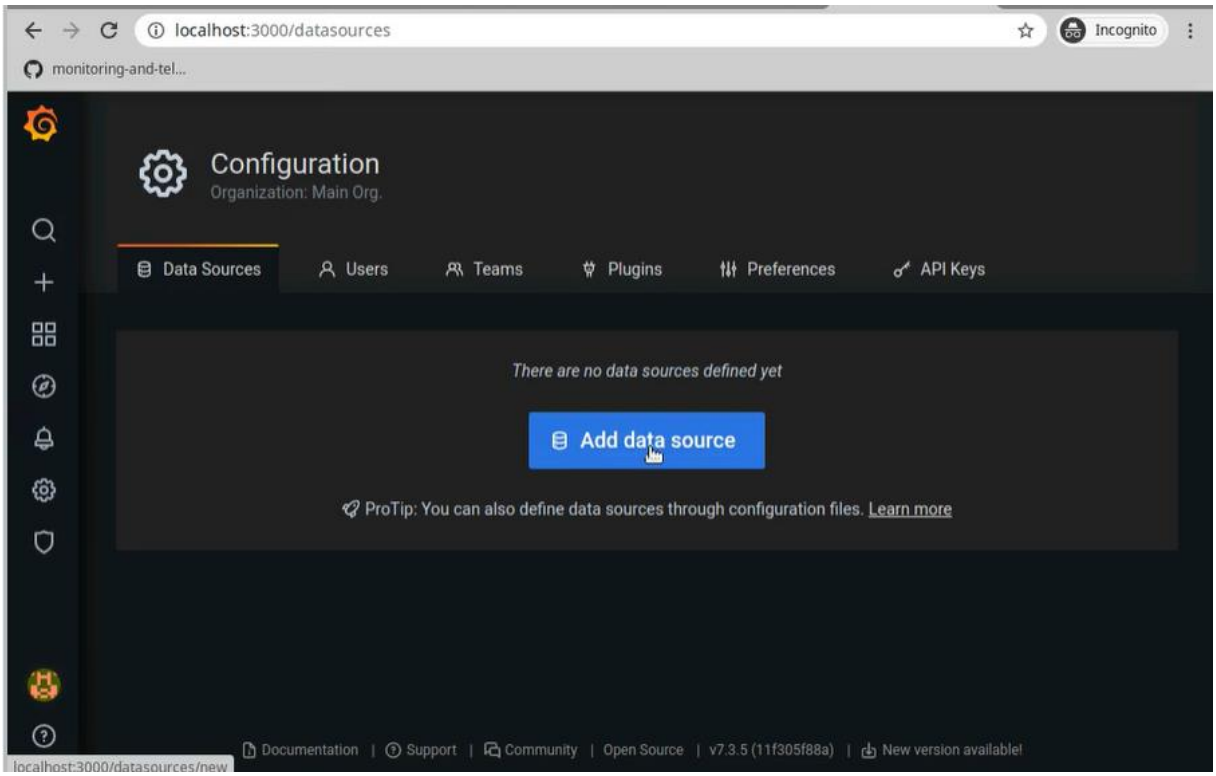
```
rhyme@ip-172-31-234-206:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
6c00bb8bd546        grafana/grafana     "/run.sh"                10 seconds ago      Up 9 seconds        0.0.0.0:3000->3000/tcp   nifty_mayer
6a9b4173c736        prom/prometheus     "/bin/prometheus --c…"   5 minutes ago       Up 5 minutes        0.0.0.0:9090->9090/tcp   frosty_wiles
```

And Grafana container is up. Now we can access the Grafana UI from localhost:3000, where we can enter “admin” as login and password.

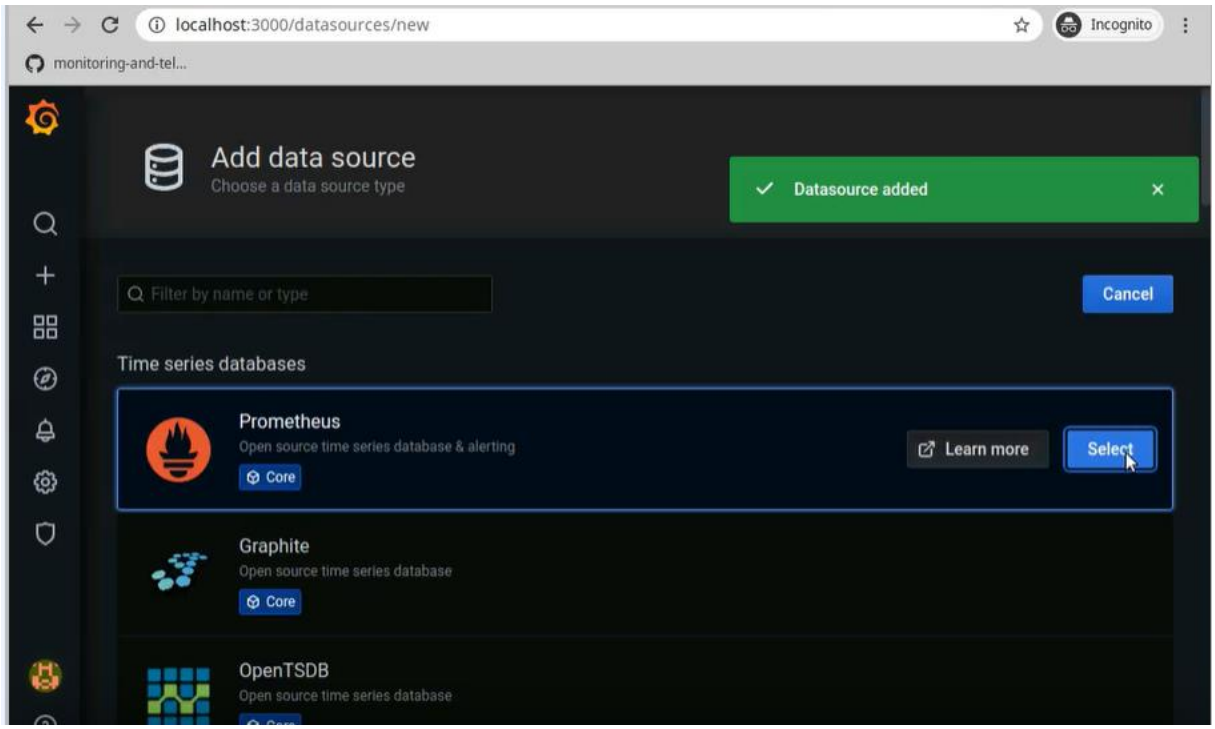




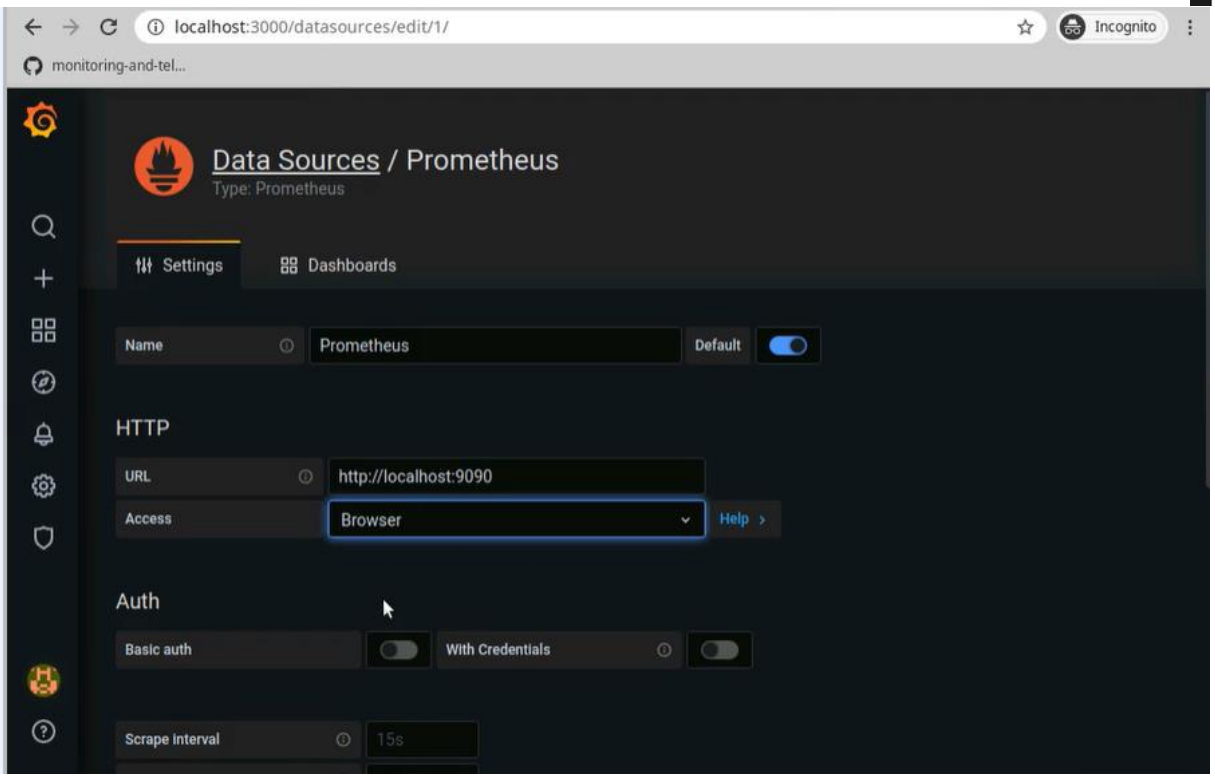
After we arrive at the landing page, we need to set up a data source for Grafana. So, here we have the option of Adding a data source:

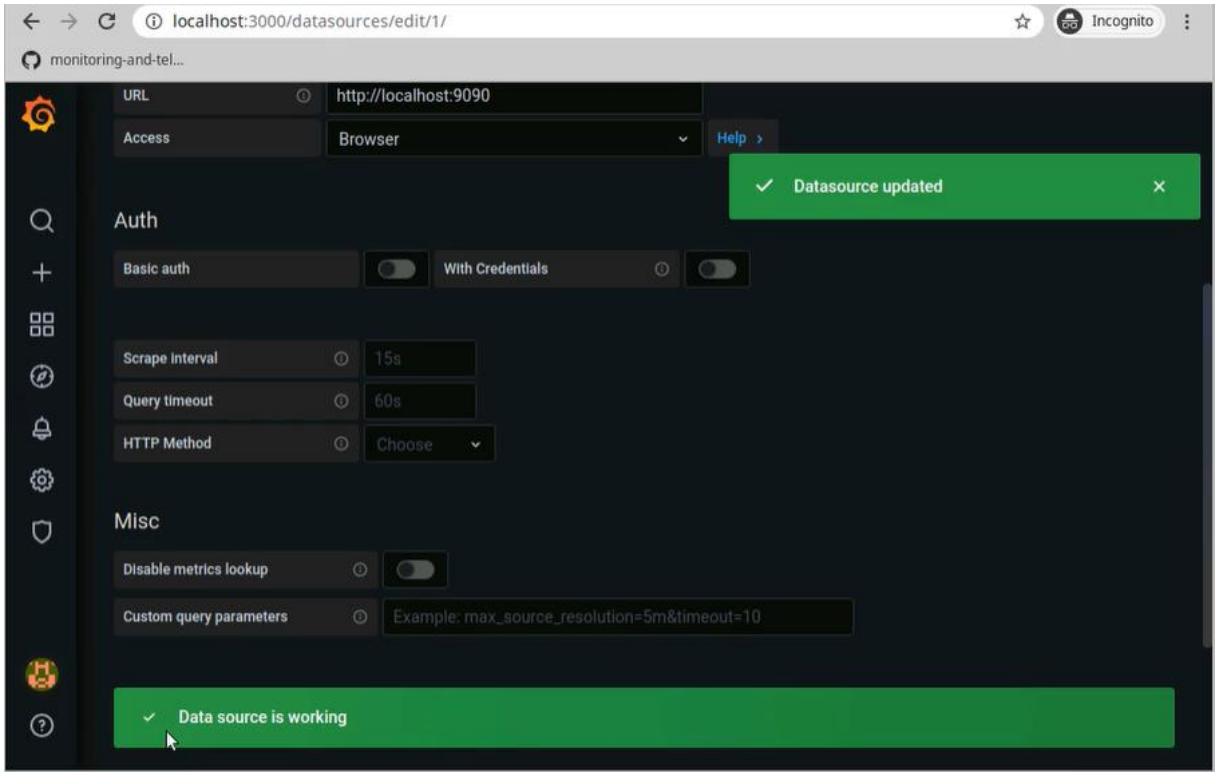


So, we want to Add a Prometheus data sources, that why we gonna select it:

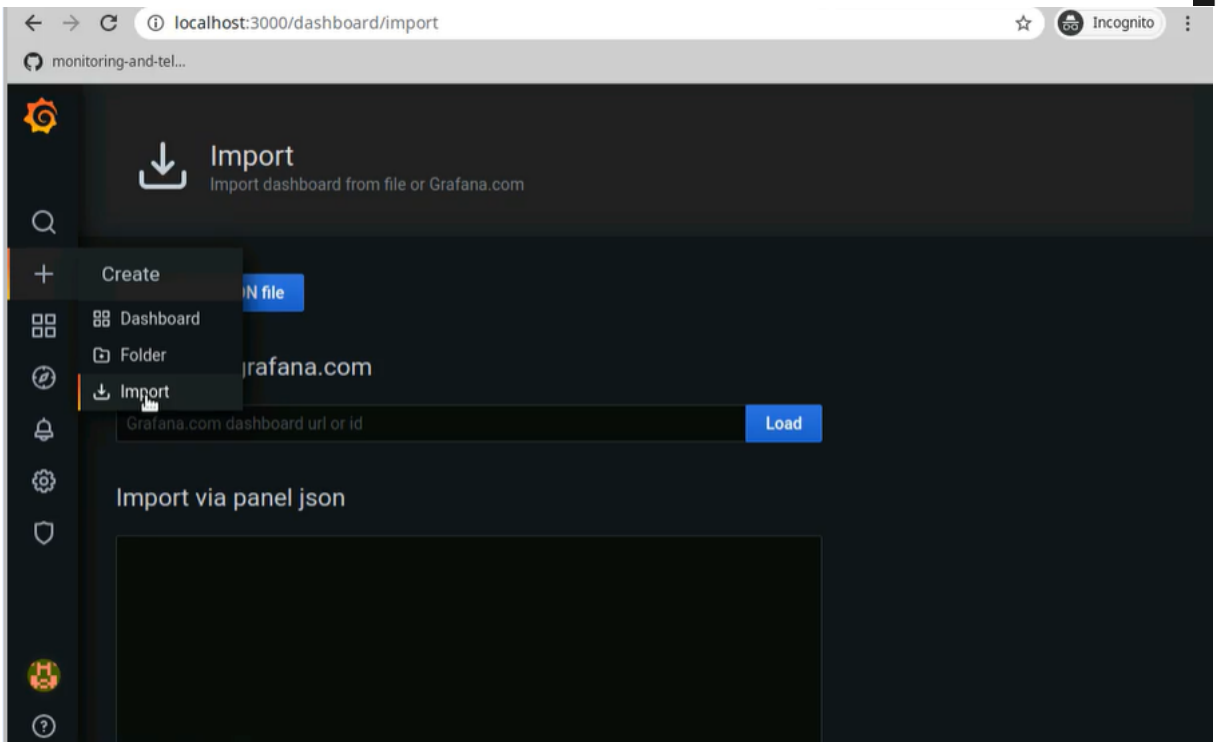


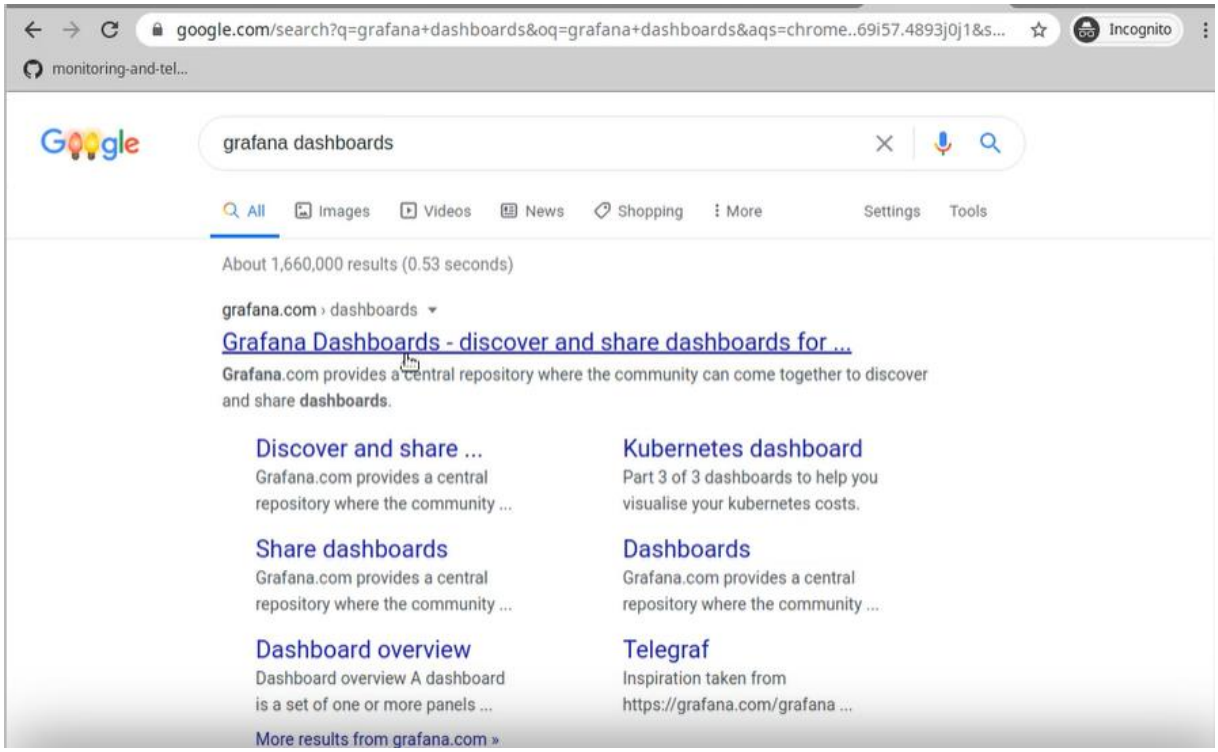
And now we configure the data sources:



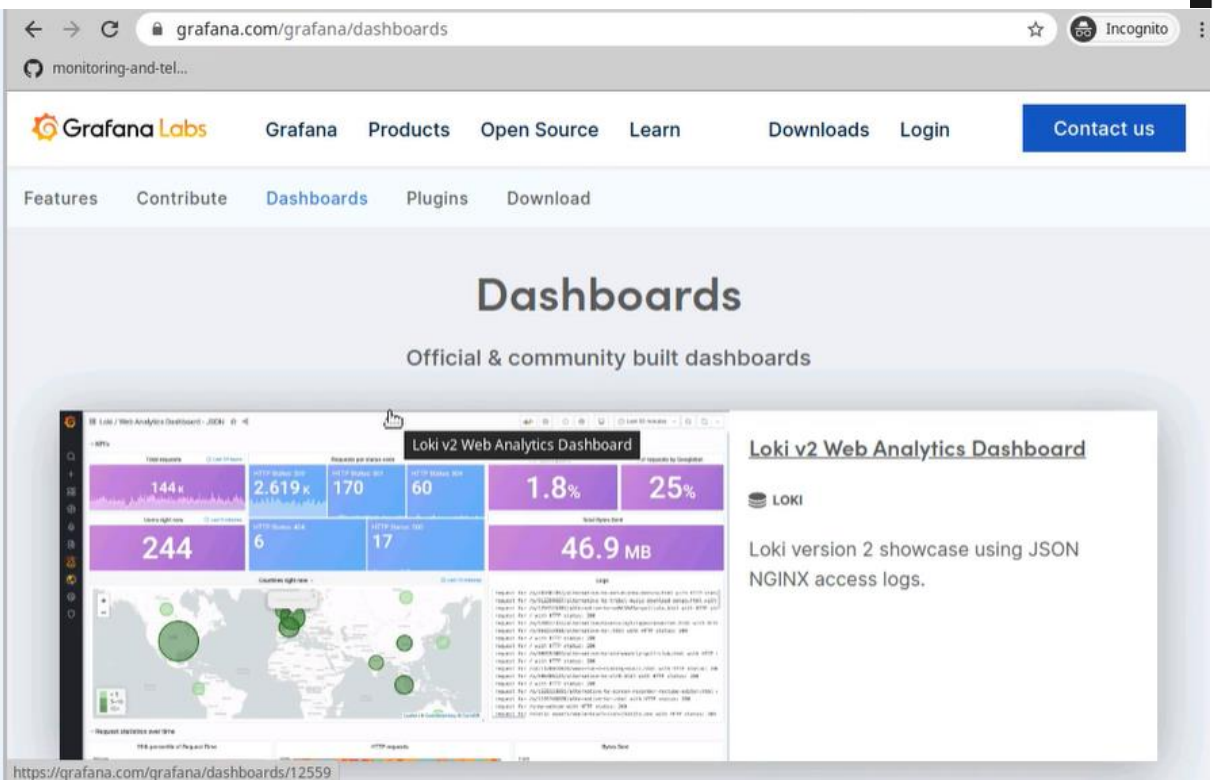


The next step is to create a dashboard to get the metrics from micrometer of SpringBoot application from Prometheus

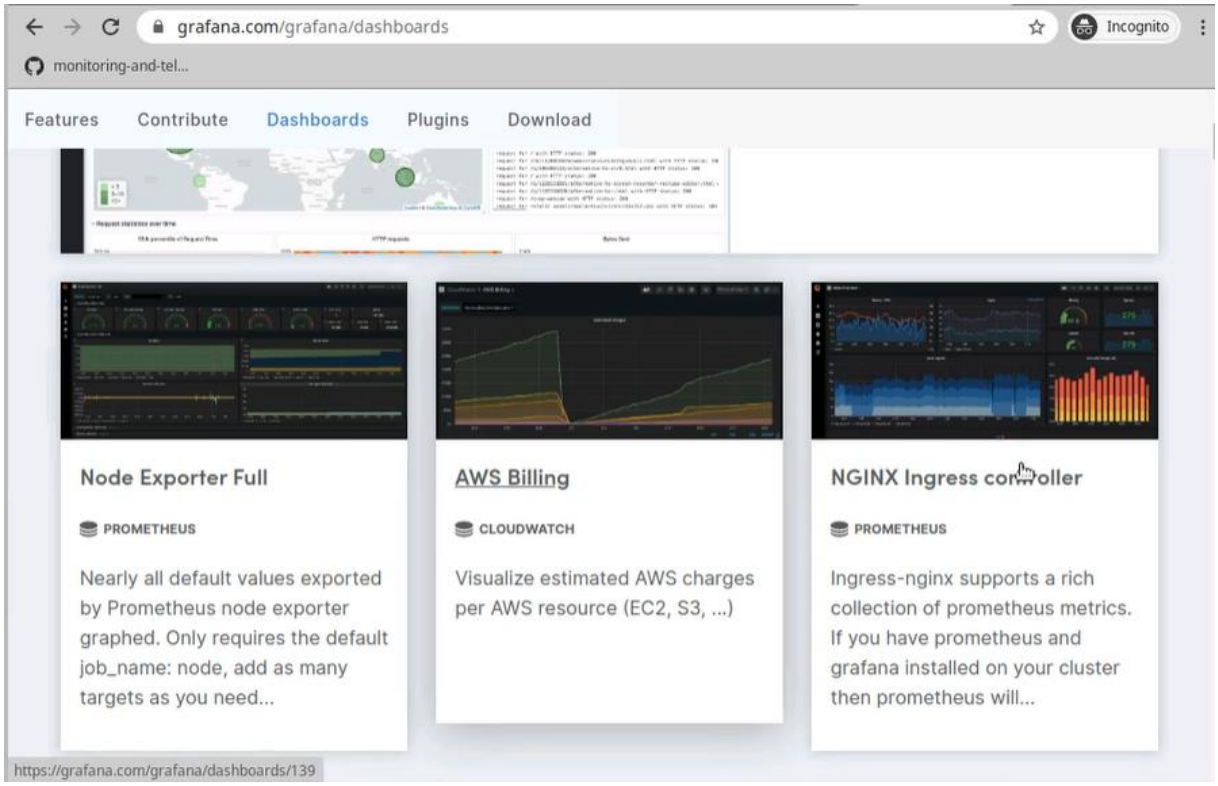




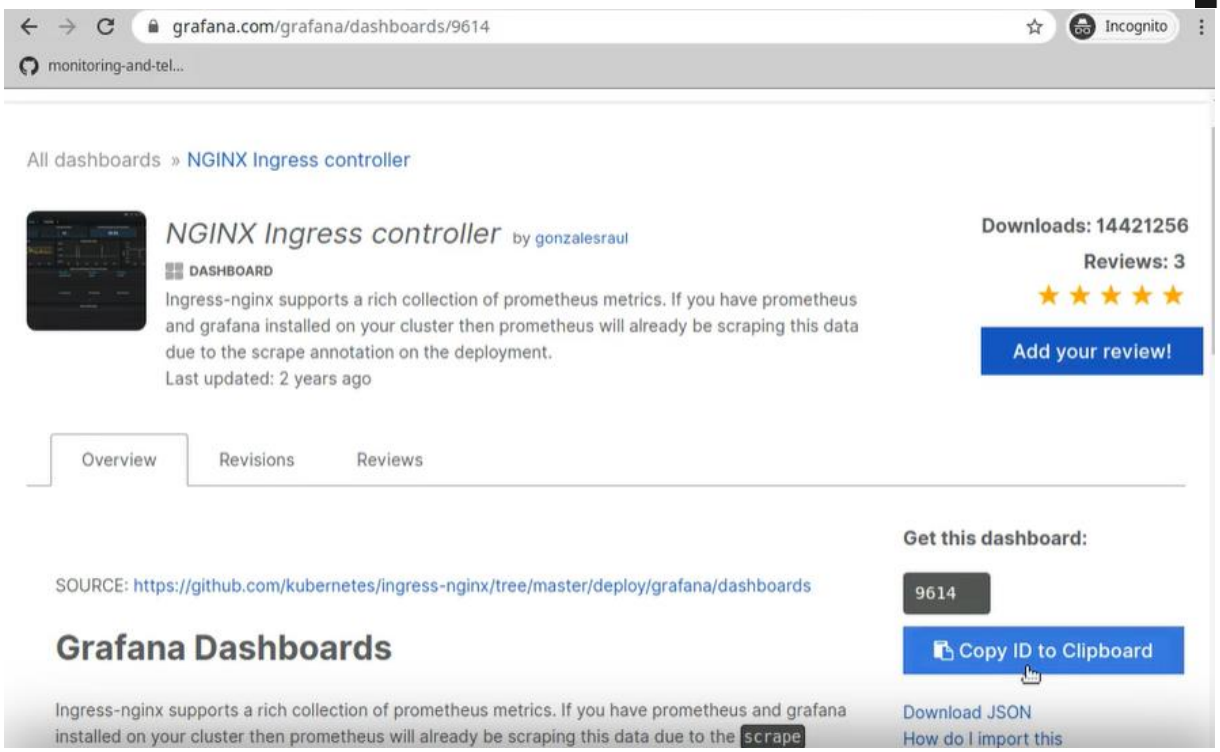
There is a public market place for all the dashboards with the code numbers:



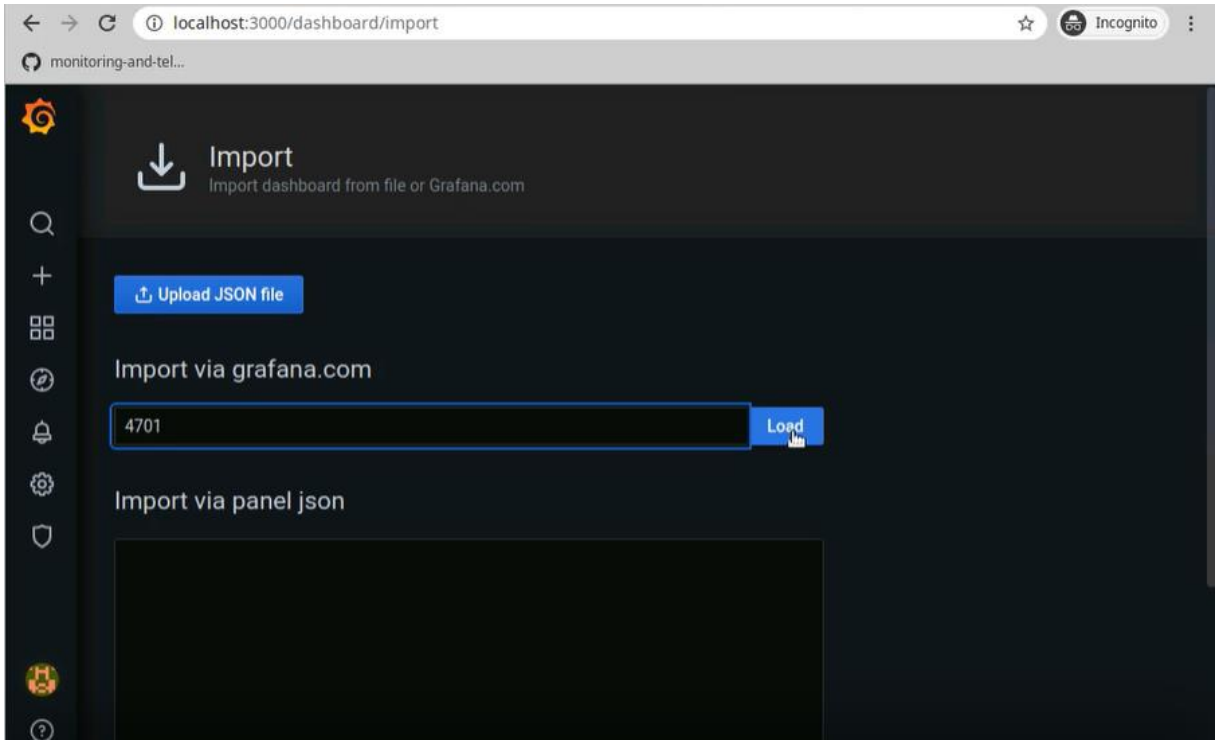
For example,we have here the dashboards for AWS Billing, Note Exporter Full, Ngnix Ingress controller ...



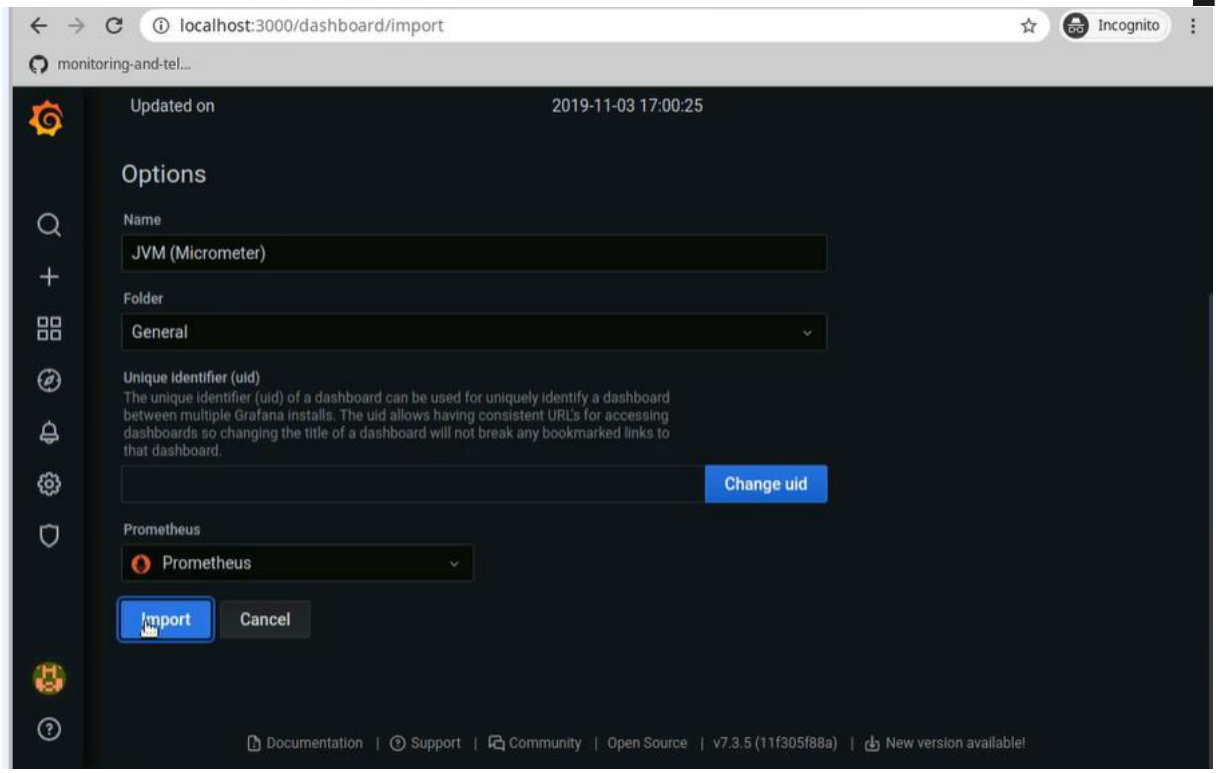
We can just click on one of these dashboards' options, and we can get the code for applying this dashboard:



The dashboard we used to monitor our application is the JVM Micrometer dashboard with import id: 4701. Now, we can paste the code and load it:



We will give our dashboard a custom name and select the prometheus data source:



Now we have a fully pre-configured dashboard, with some important metrics showcased, out of the box.


JVM (Micrometer)
Application
MonitoringSpringDemoProject
Instance
192.168.0.9:8080
JVM Memory Pools Heap
JVM Memory Pools Non-Heap
Restart Detection
Quick Facts
Uptime
2.3 hour
Start time
2020-11-10 11:58:21
Heap used
3.76%
Non-Heap used
4.30%
I/O Overview
Rate
Errors
Duration
Utilisation
No data
JVM Memory
JVM Heap
JVM Non-Heap
JVM Total
JVM Process Memory
JVM Misc
CPU Usage
Load
Threads
Thread States
Log Events
File Descriptors
JVM Memory Pools (Heap)
G1 Eden Space
G1 Old Gen
G1 Survivor Space
JVM Memory Pools (Non-Heap)
Metaspace
Compressed Class Space
CodeHeap 'non-profiled nmethods'
CodeHeap 'non-nmethods'
Garbage Collection
Collections
Pause Durations
Allocated/Promoted
Classloading
Classes loaded
Class delta
Buffer Pools
Direct Buffers
Direct Buffers
Mapped Buffers
Mapped Buffers

Task-5: Implement Log Monitoring & Analysis with ELK Stack.

In this task, we will understand how to monitor the activity for application. So, whether happens to an application is normally logged for written down in a log file. This task will cover how logs are analyzed and how to create a stack that will be a central log aggregator. So, that brings us to ELK.

ELK is an acronym, which stands for Elasticsearch, Log stash and Kibana.

1. ELK

ELK Stack is designed to allow users to take to data from any source, in any format, and to search, analyze, and visualize that data in real time.

ELK provides centralized logging that be useful when attempting to identify problems with servers or applications. It allows you to search all your logs in a single place. It also helps to find issues that occur in multiple servers by connecting their logs during a specific time frame.

2. What is the ELK Stack?

The ELK Stack is a collection of three open-source products—Elasticsearch, Logstash, and Kibana. They are all developed, managed, and maintained by the company Elastic.

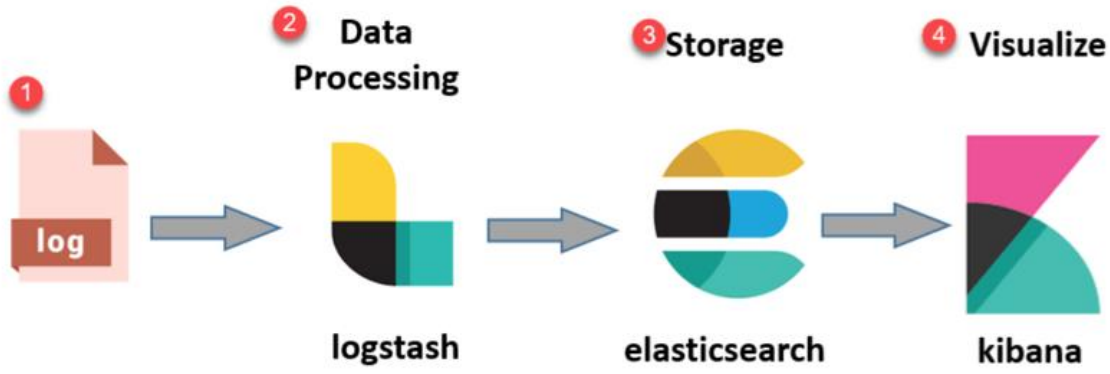
E stands for ElasticSearch: used for storing logs

L stands for LogStash: used for both shipping as well as processing and storing logs

K stands for Kibana: is a visualization tool (a web interface) which is hosted through Nginx or Apache

ELK Stack is designed to allow users to take to data from any source, in any format, and to search, analyze, and visualize that data in real time.

ELK provides centralized logging that be useful when attempting to identify problems with servers or applications. It allows you to search all your logs in a single place. It also helps to find issues that occur in multiple servers by connecting their logs during a specific time frame.



1. Logs: Server logs that need to be analyzed are identified
2. Logstash: Collect logs and events data. It even parses and transforms data
3. ElasticSearch: The transformed data from Logstash is Store, Search, and indexed.
4. Kibana: Kibana uses Elasticsearch DB to Explore, Visualize, and Share

First thing, we gonna clear all the Docker Containers with the command:

#sudo docker rm -f $(sudo docker ps -a -q)

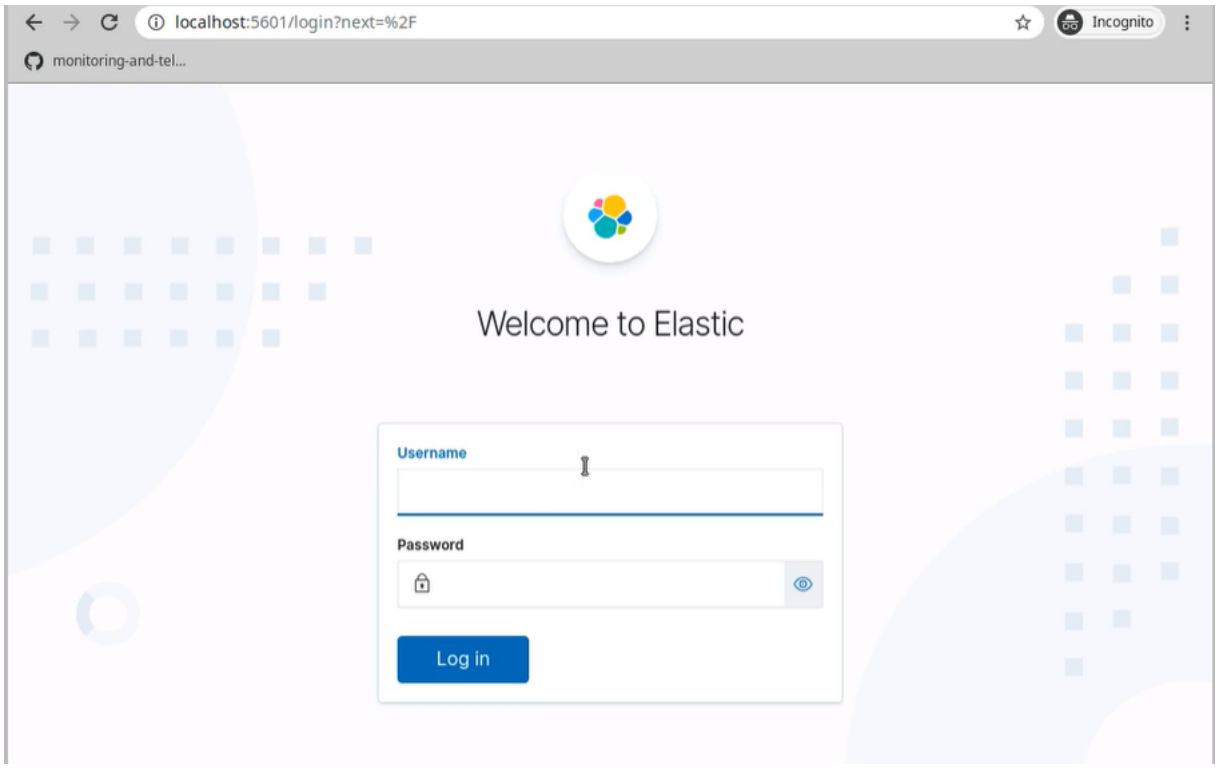
```
rhyme@ip-172-31-234-206:~$ sudo docker rm -f $(sudo docker ps -a -q)
6c00bb8bd546
6a9b4173c736
```

So, Prometheus have been removed. Now, to install ELK we type:

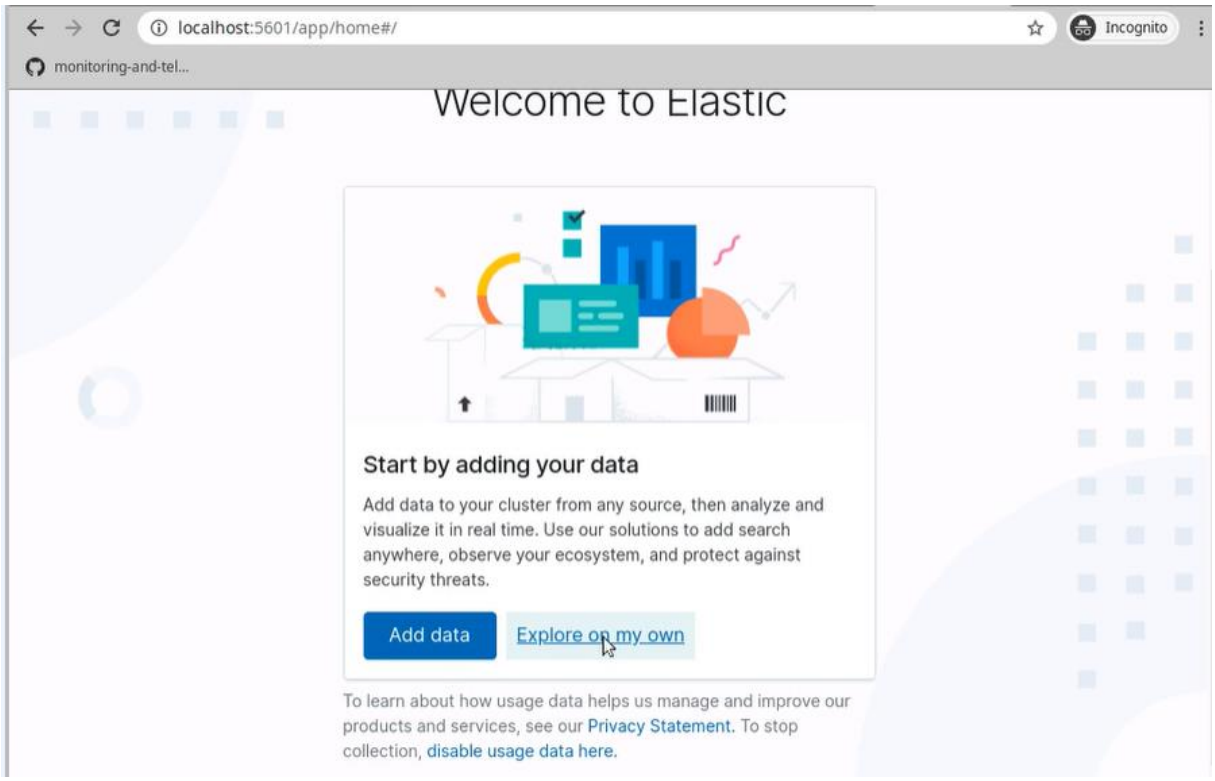
#sudo docker-compose up

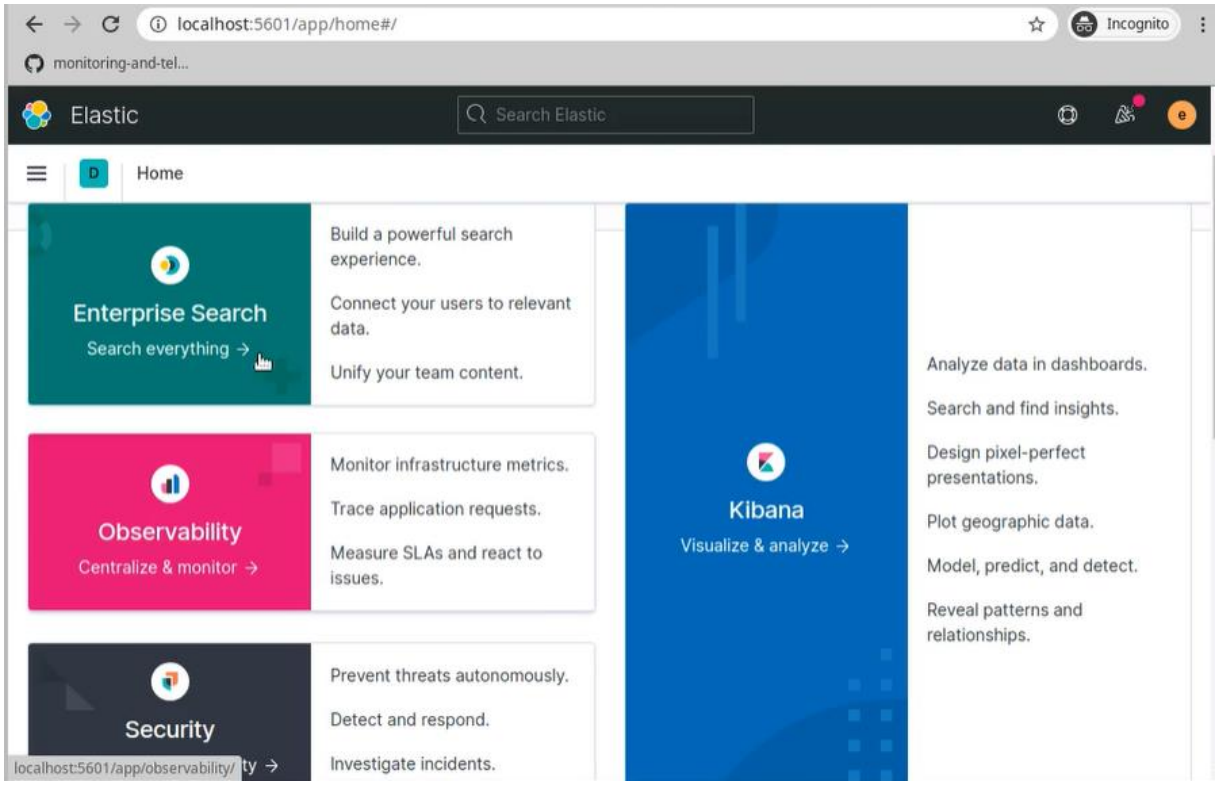
```
rhyme@ip-172-31-234-206:~$ cd docker-elk/
rhyme@ip-172-31-234-206:~/docker-elk$ pwd
/home/rhyme/docker-elk
rhyme@ip-172-31-234-206:~/docker-elk$ ls -ltr
total 48
drwxrwxr-x 4 rhyme rhyme  4096 Dec 11 23:54 logstash
drwxrwxr-x 3 rhyme rhyme  4096 Dec 11 23:54 kibana
drwxrwxr-x 6 rhyme rhyme  4096 Dec 11 23:54 extensions
drwxrwxr-x 3 rhyme rhyme  4096 Dec 11 23:54 elasticsearch
-rw-rw-r-- 1 rhyme rhyme  1782 Dec 11 23:54 docker-stack.yml
-rw-rw-r-- 1 rhyme rhyme  1803 Dec 11 23:54 docker-compose.yml
-rw-rw-r-- 1 rhyme rhyme 19457 Dec 11 23:54 README.md
-rw-rw-r-- 1 rhyme rhyme  1082 Dec 11 23:54 LICENSE
rhyme@ip-172-31-234-206:~/docker-elk$ sudo docker-compose up
Creating network "dockerelk_elk" with driver "bridge"
Creating volume "dockerelk_elasticsearch" with default driver
Creating dockerelk_elasticsearch_1 ...
Creating dockerelk_elasticsearch_1 ... done
Creating dockerelk_kibana_1 ...
Creating dockerelk_logstash_1 ...
Creating dockerelk_logstash_1
Creating dockerelk_kibana_1 ... done
Attaching to dockerelk_elasticsearch_1, dockerelk_logstash_1, dockerelk_kibana_1
logstash_1       | Using bundled JDK: /usr/share/logstash/jdk
logstash_1       | OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a fu
ture release.
elasticsearch_1  | Created elasticsearch keystore in /usr/share/elasticsearch/config/elasticsearch.keystore
```

It's starting Elasticsearch Logstach Kibana. In the browser we can type the port where Kibana listens:

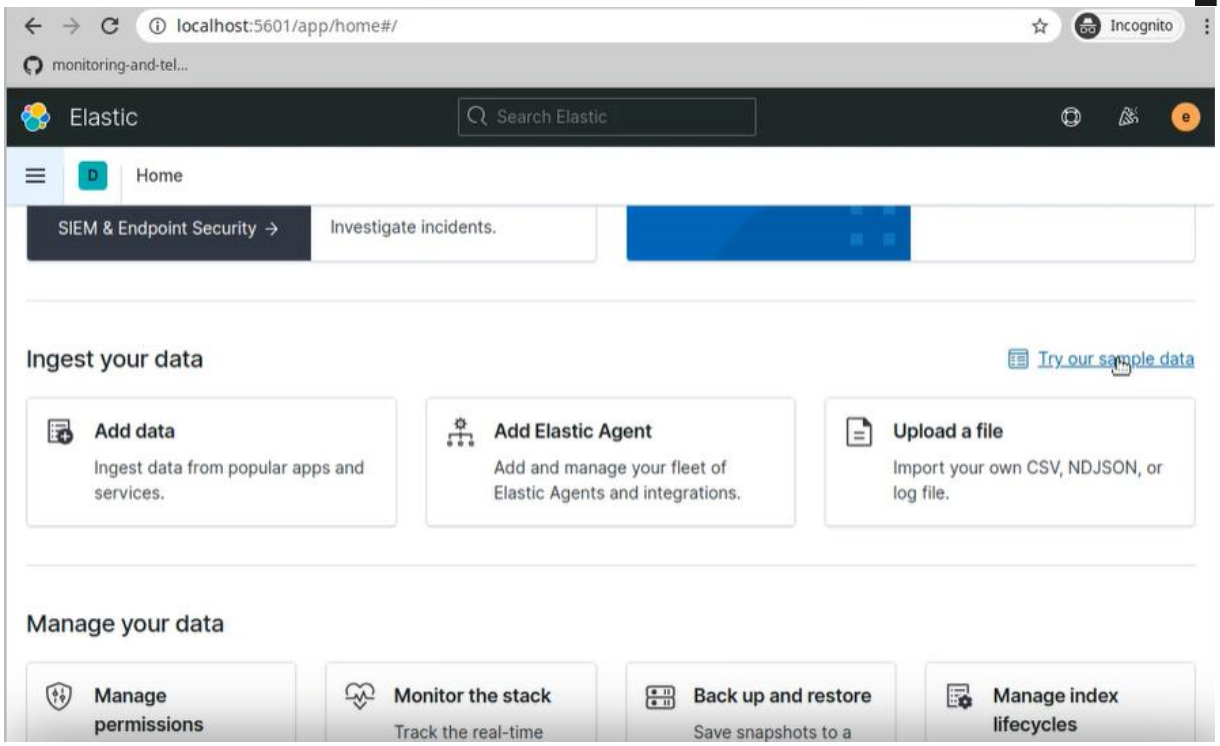


The default user name is 'elastic' and the password is 'changeme', and then we see the Kibana dashboard appears:

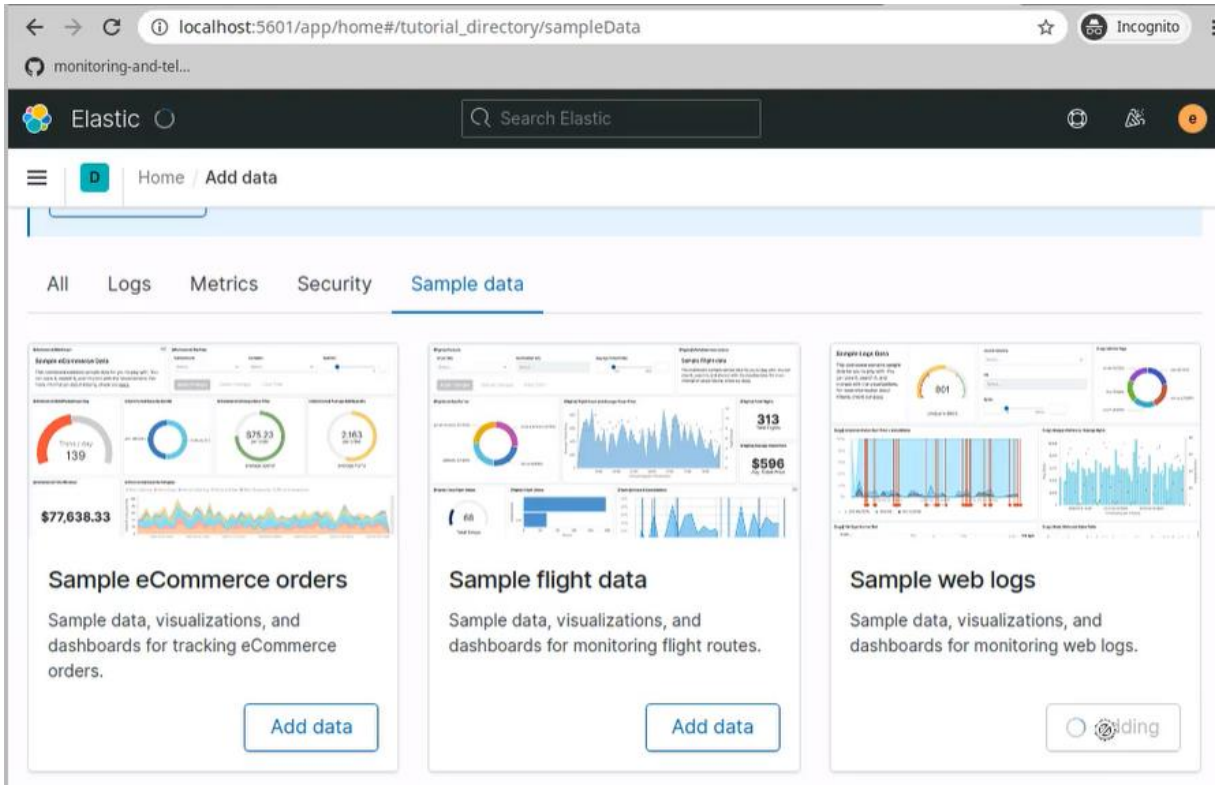




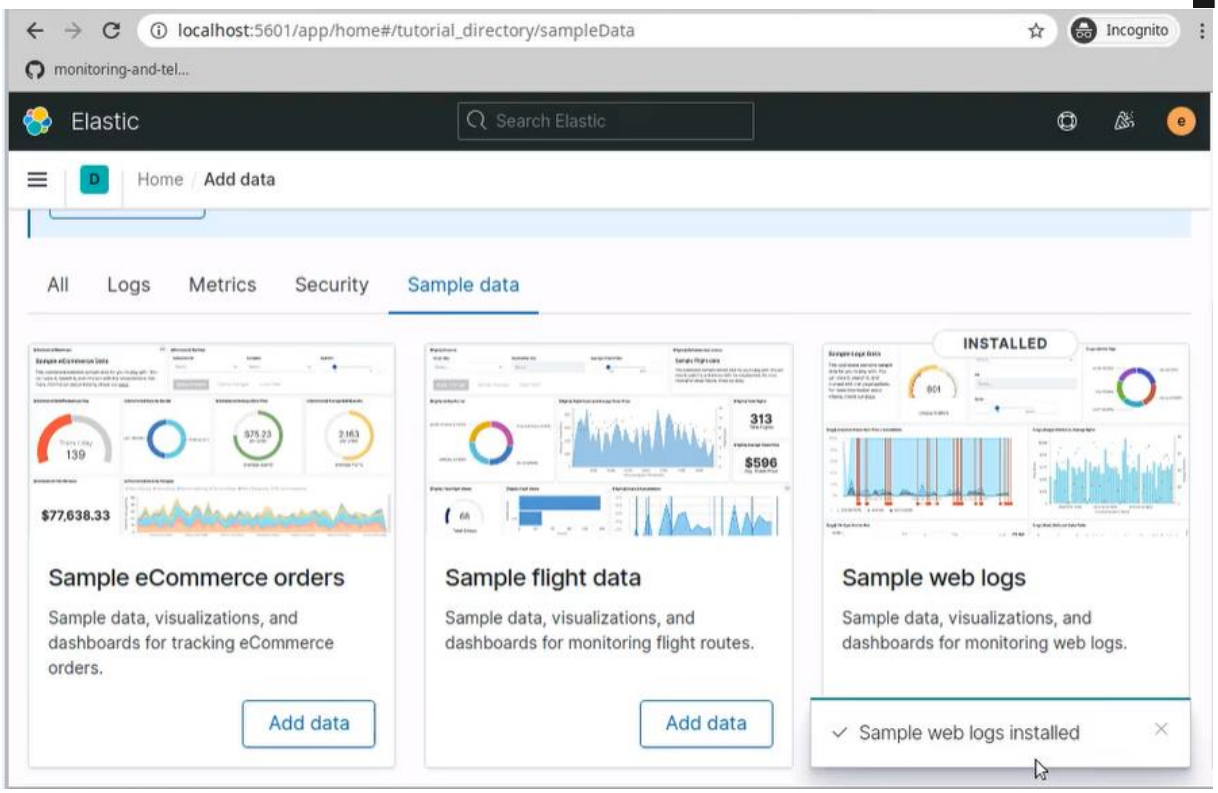
We gonna to try importing the example data:



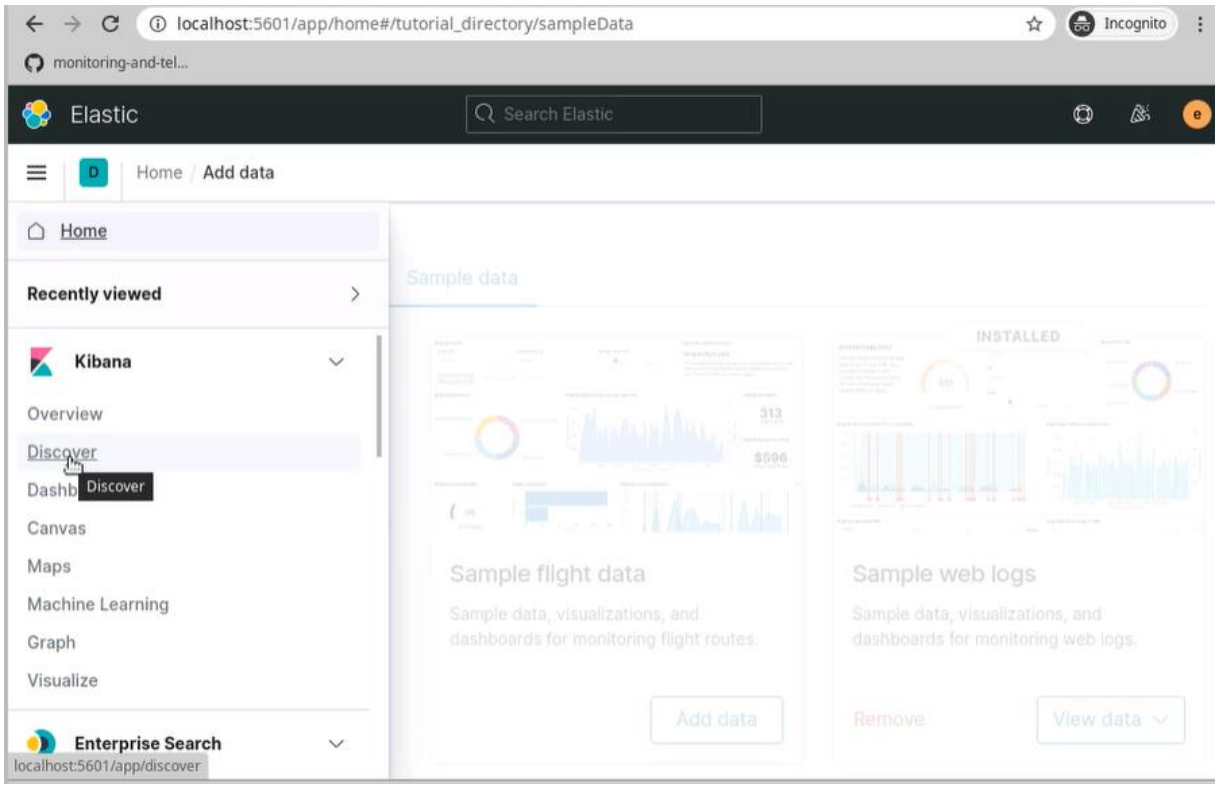
And, then it will import the sample data into our Elasticsearch, and it will be available for us to view in Kibana.



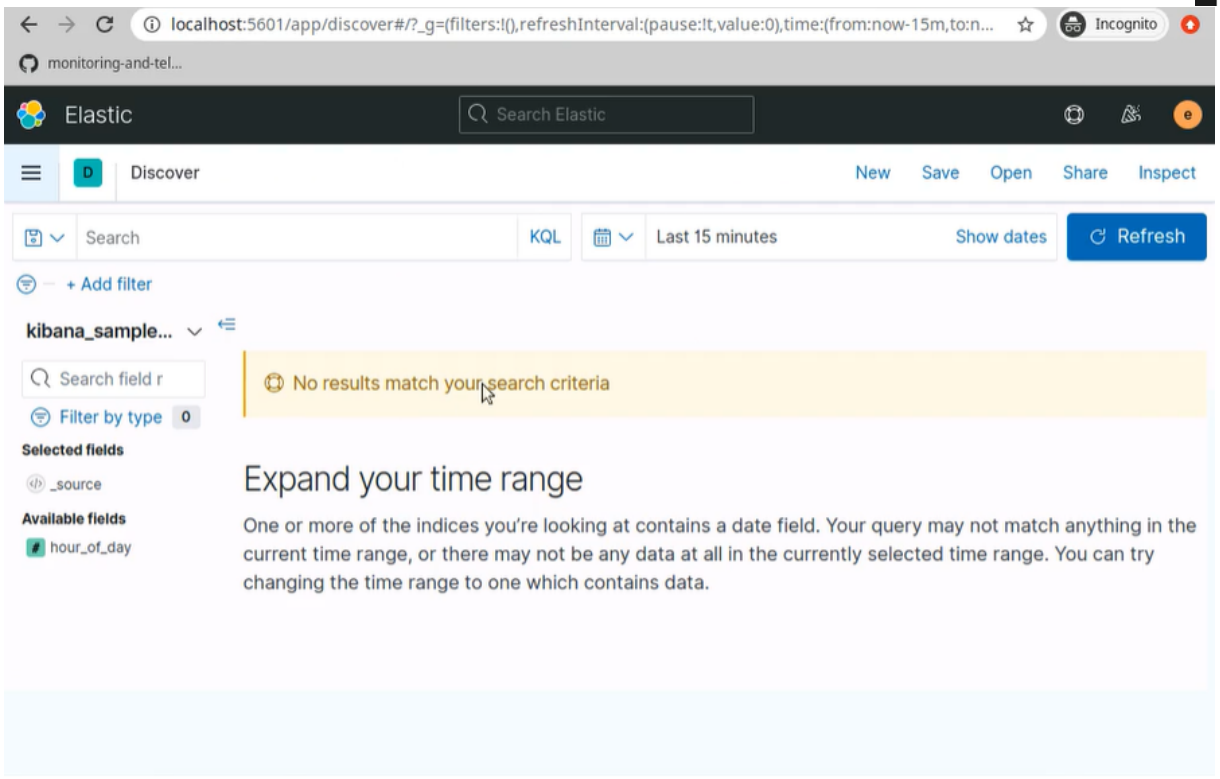
And here we can see the Web log have installed successfully:



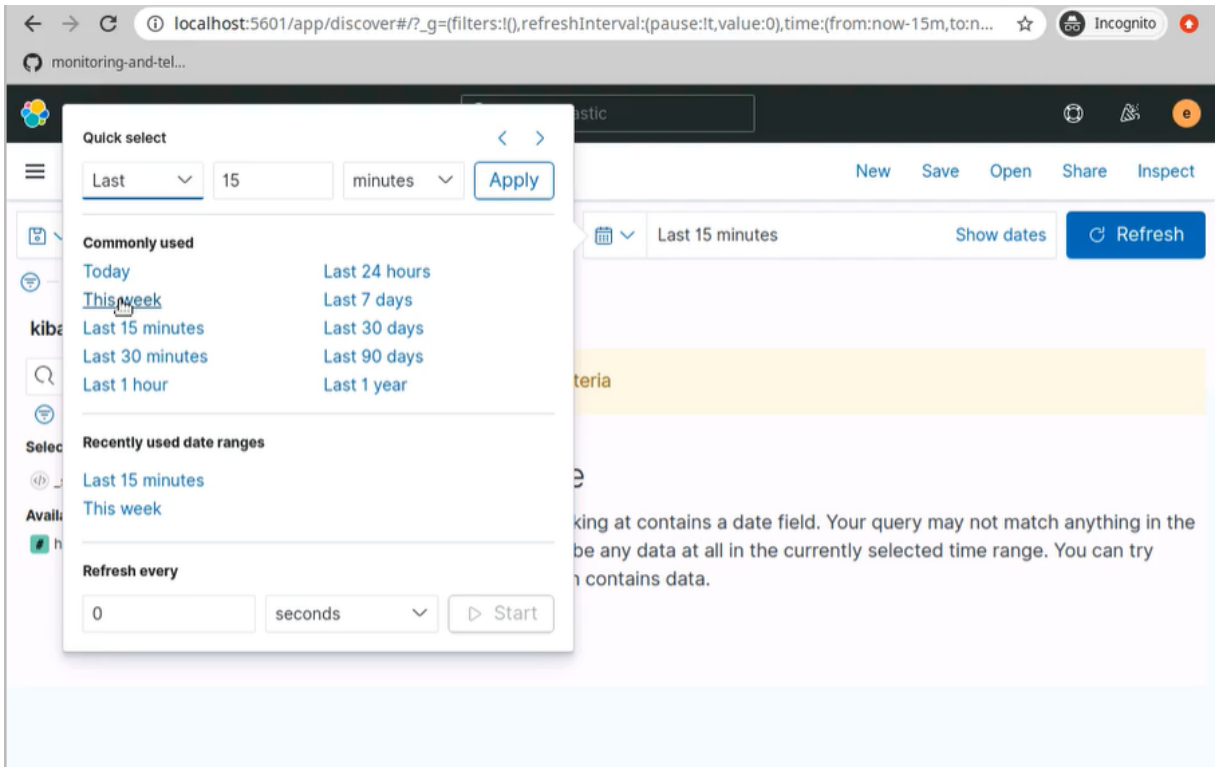
And now let's navigate our logs and see how dashboards are generated out of this sample logs.



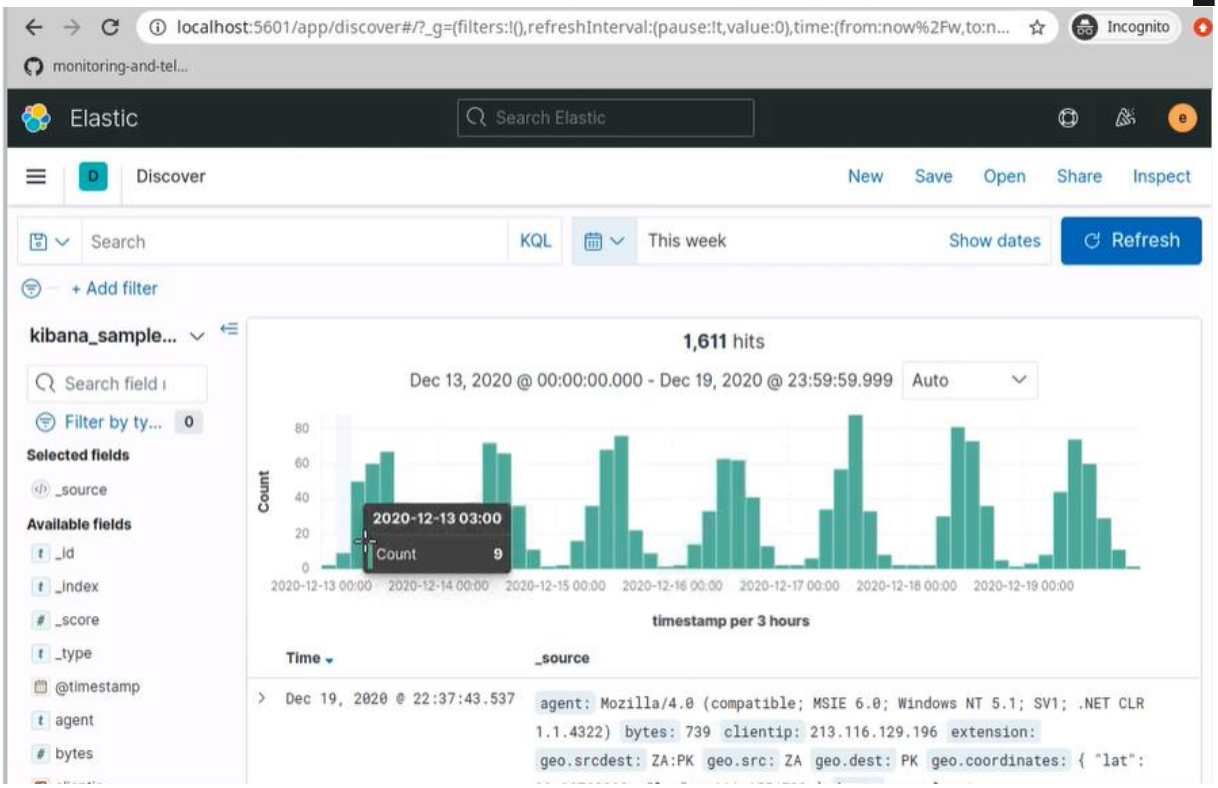
We will be able to see the centralized logs for this sample data that we imported:



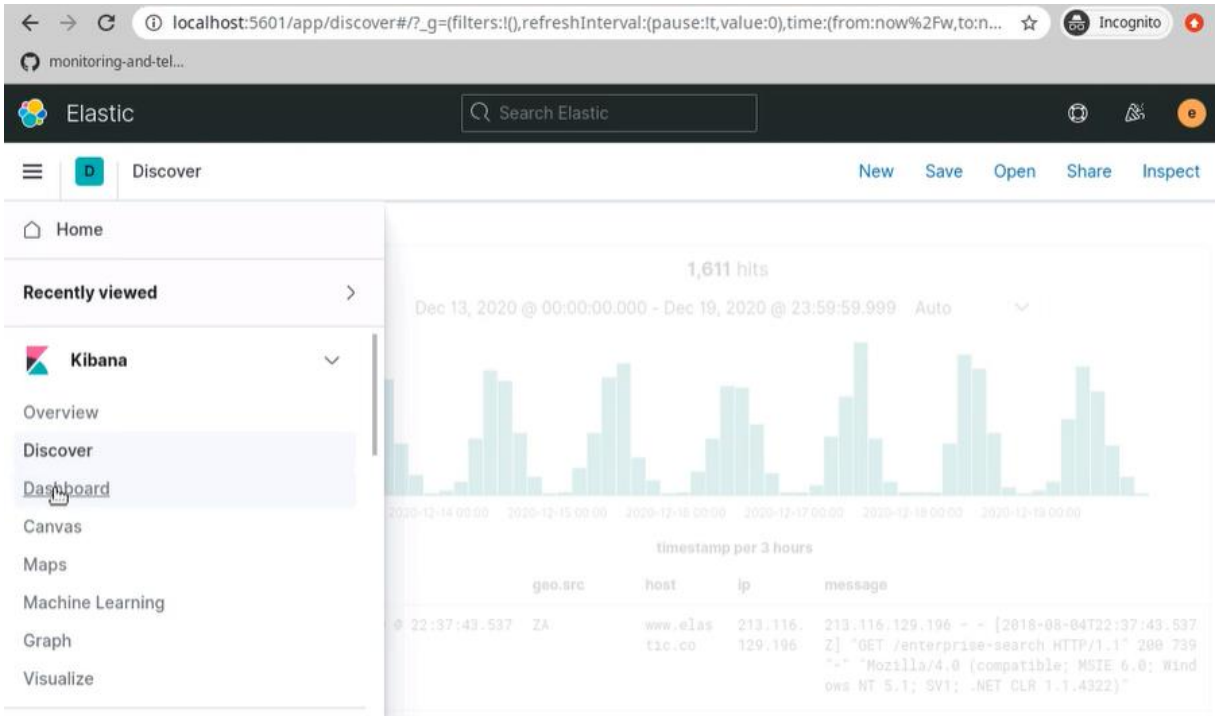
And for the time frame let's select now 'this week':



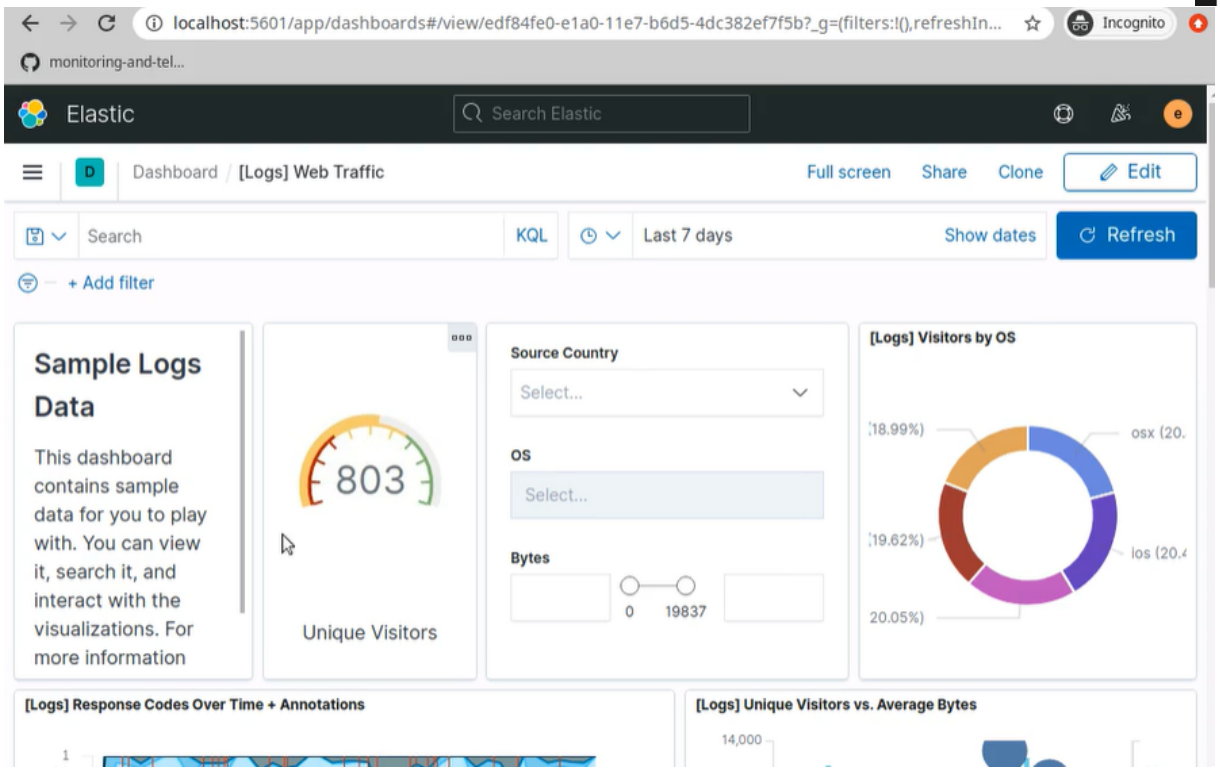
So, we got, the logs for this week:

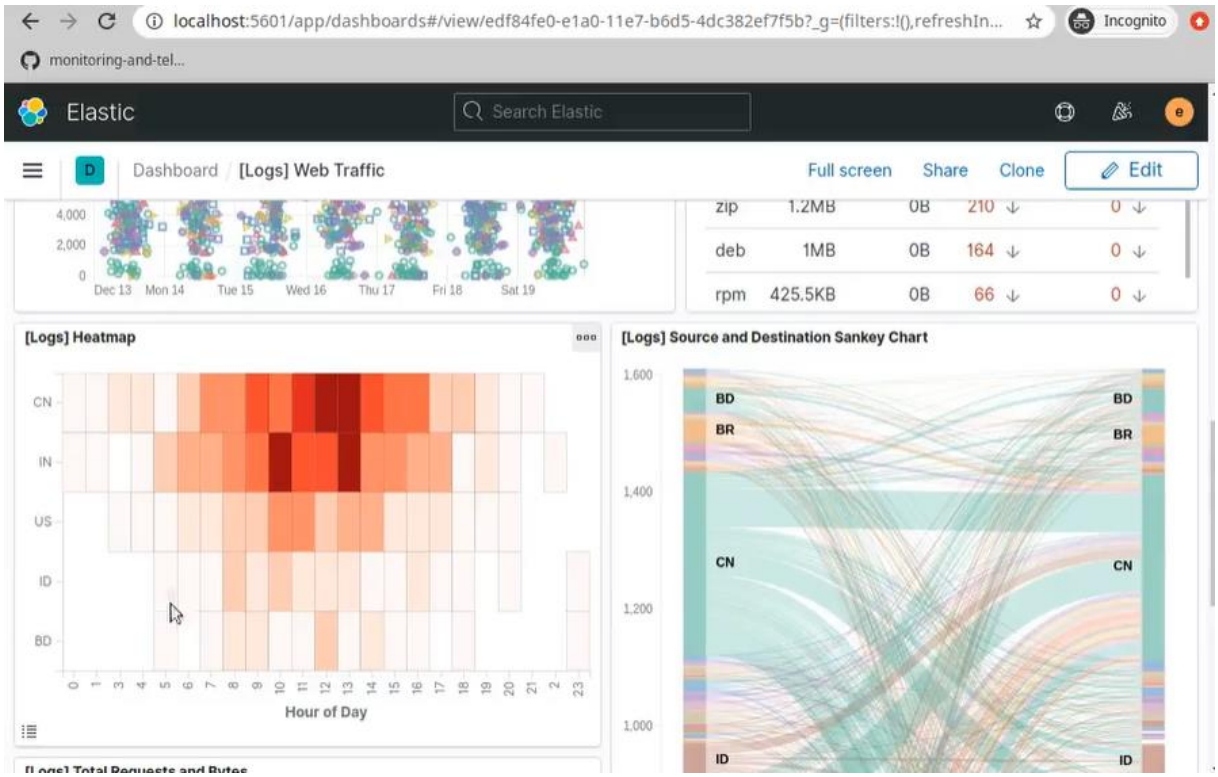


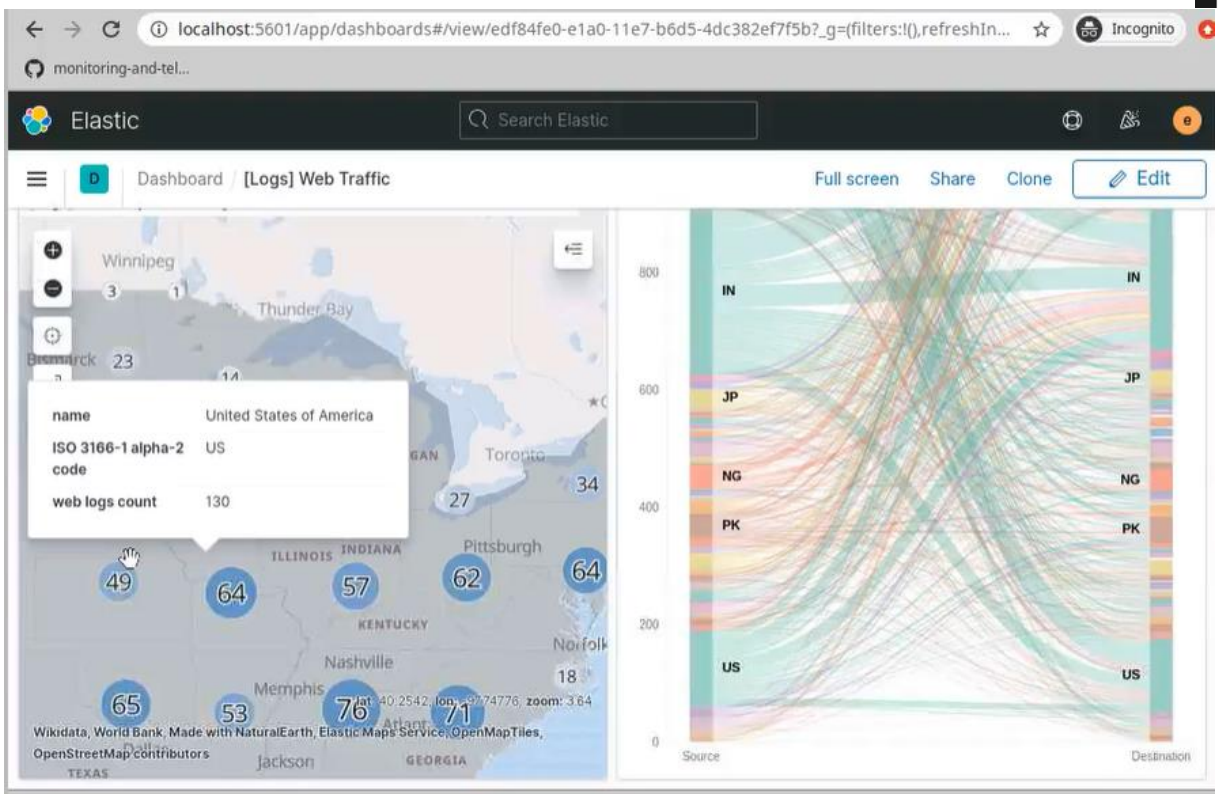
We can also see the dashboard of data by clicking on 'Dashboard':



And then we can see the number of unique visitors, pie charts from the log data from Apache


localhost:5601/app/dashboards#/view/edf84fe0-e1a0-11e7-b6d5-4dc382ef7f5b?_g=(filters:!(),refreshIn...
Incognito
monitoring-and-tel...
Elastic
Search Elastic
Dashboard / [Logs] Web Traffic
Full screen
Share
Clone
Edit
Search
KQL
Last 7 days
Show dates
Refresh
+ Add filter
Sample Logs Data
This dashboard contains sample data for you to play with. You can view it, search it, and interact with the visualizations. For more information
803
Unique Visitors
Source Country
Select...
OS
Select...
Bytes
0
19837
[Logs] Visitors by OS
18.99%)
osx (20.
19.62%)
ios (20.
20.05%)
[Logs] Response Codes Over Time + Annotations
[Logs] Unique Visitors vs. Average Bytes
14,000


localhost:5601/app/dashboards#/view/edf84fe0-e1a0-11e7-b6d5-4dc382ef7f5b?_g=(filters:!(),refreshIn...
Incognito
monitoring-and-tel...
Elastic
Search Elastic
Dashboard / [Logs] Web Traffic
Full screen
Share
Clone
Edit
4,000
2,000
0
Dec 13
Mon 14
Tue 15
Wed 16
Thu 17
Fri 18
Sat 19
zip 1.2MB 0B 210 0
deb 1MB 0B 164 0
rpm 425.5KB 0B 66 0
[Logs] Heatmap
CN
IN
US
ID
BD
Hour of Day
[Logs] Source and Destination Sankey Chart
1,600
1,400
1,200
1,000
BD
BR
CN
ID


localhost:5601/app/dashboards#/view/edf84fe0-e1a0-11e7-b6d5-4dc382ef7f5b?_g=(filters:!(),refreshIn...
Incognito
monitoring-and-tel...
Elastic
Search Elastic
Dashboard
[Logs] Web Traffic
Full screen
Share
Clone
Edit
Winnipeg
Thunder Bay
name
United States of America
ISO 3166-1 alpha-2 code
US
web logs count
130
Toronto
Pittsburgh
ILLINOIS
INDIANA
KENTUCKY
Nashville
Memphis
Jackson
TEXAS
GEORGIA
Wikidata, World Bank, Made with NaturalEarth, Elastic Maps Service, OpenMapTiles, OpenStreetMap contributors
800
600
400
200
0
IN
JP
NG
PK
US
Source
Destination