



Monitoring and Telemetry for Production Systems

Kawtar Oukil



Table des matières

I.Introduction:.....	2
<i>Task-1 : Monitoring & Telemetry Fundamentals.</i>	4
<i>Task-2: Linux Command line Monitoring Tools</i>	4
1. Top.....	5
2. Htop.....	5
3. iostat.....	7
4. Lsof	8
5. Vmstat.....	9
6. netstat.....	9
<i>Task-3: Server & Docker Container Monitoring with Netdata & cAdvisor.</i>	12
1. Installing Netdata:	12
2. Accessing Netdata:	14
3. Basic Docker Monitoring.....	17
<i>Task-4: Perform Application Monitoring & Telemetry with Prometheus & Grafana - Part I.</i>	21
1. Prometheus:.....	22
1.1 What is Prometheus?	22
1.2 Why do we need Prometheus?	22
2. Grafana	33
2.1 What is Grafana?	33
2.2 Why Grafana?.....	33
2.3 Installing Grafana:	33
<i>Task-5: Implement Log Monitoring & Analysis with ELK Stack.</i>	42
1. ELK	42
2. What is the ELK Stack?.....	42



I. Introduction:

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:

```
Terminal - rhyme@ip-10-199-127-202:~  
File Edit View Terminal Tabs Help  
rhyme@ip-10-199-127-202:~$ top
```

The screenshot shows a terminal window titled "Terminal - rhyme@ip-10-199-127-202:~". The command "top" is run, and the output is displayed. The output shows system load average (0.00, 0.01, 0.00), task counts (157 total, 1 running, 117 sleeping, 0 stopped, 0 zombie), CPU usage (%CPU(s)), memory usage (Kib Mem: 8064020 total, 6940432 free, 468432 used, 655156 buff/cache, Kib Swap: 0 total, 0 free, 0 used, 7359628 avail Mem), and a detailed list of processes. The process "rhyme" is highlighted with PID 2553, PR 20, NI 0, VIRT 44540, RES 4092, SHR 3432, %CPU 0.3, %MEM 0.1, and TIME+ 0:00.08. Other processes listed include x11vnc, Xorg, dockerd, containerd, and various kernel threads and workers.

this command displays these metrics:

```
Terminal - rhyme@ip-10-199-127-202:~  
File Edit View Terminal Tabs Help  
top - 20:57:38 up 20 min, 1 user, load average: 0.00, 0.01, 0.00  
Tasks: 157 total, 1 running, 117 sleeping, 0 stopped, 0 zombie  
%CPU(s): 0.8 us, 0.8 sy, 0.0 ni, 98.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.3 st  
Kib Mem : 8064020 total, 6940432 free, 468432 used, 655156 buff/cache  
Kib Swap: 0 total, 0 free, 0 used. 7359628 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1751	root	20	0	125044	15300	9540	S	2.0	0.2	0:08.21	x11vnc
1105	root	20	0	497552	60760	33088	S	1.7	0.8	0:22.23	Xorg
947	root	20	0	905088	83176	45456	S	0.7	1.0	0:00.65	dockerd
913	root	20	0	885796	40148	23024	S	0.3	0.5	0:01.10	containerd
1568	rhyme	20	0	266372	37744	16836	S	0.3	0.5	0:00.34	applet.py
2553	rhyme	20	0	44540	4092	3432	R	0.3	0.1	0:00.08	top
1	root	20	0	159784	9028	6696	S	0.0	0.1	0:02.33	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	I	0.0	0.0	0:00.05	kworker/0:0
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H
5	root	20	0	0	0	0	I	0.0	0.0	0:00.06	kworker/u4:0
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
7	root	20	0	0	0	0	S	0.0	0.0	0:00.12	ksoftirqd/0
8	root	20	0	0	0	0	I	0.0	0.0	0:00.34	rcu_sched
9	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_bh
10	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
14	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/1
15	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/1
16	root	20	0	0	0	0	S	0.0	0.0	0:00.10	kssoftirqd/1
18	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/1:0H
19	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
20	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
21	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tasks_kthre
22	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kauditd
24	root	20	0	0	0	0	I	0.0	0.0	0:00.19	kworker/1:1
25	root	20	0	0	0	0	S	0.0	0.0	0:00.00	khungtaskd
26	root	20	0	0	0	0	S	0.0	0.0	0:00.00	oom_reaper
27	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	writelock
28	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kcompactd0
29	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd
30	root	39	19	0	0	0	S	0.0	0.0	0:00.00	khugepaged
31	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	crypto
32	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kintegrityd
33	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kblockd
34	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	ata_sff

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 screenshot shows a terminal window titled "Terminal - rhyme@ip-172-31-234-206: ~". The window displays system statistics at the top: CPU usage (4.0%), Tasks (102, 221 thr; 1 running), Load average (0.00 0.02 0.02), and memory usage (614M/7.69G, Uptime: 00:55:49). Below this is a detailed process list with columns for PID, USER, PRI, NI, VIRT, RES, SHR, %CPU, %MEM, TIME+, and Command. The processes listed include various daemons like avahi, cron, and systemd, along with Xfce4 components like xfce4-panel, xfdesktop, and nm-applet. The bottom of the window shows a series of keyboard shortcuts for navigation.

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

The screenshot shows a terminal window titled "Terminal - rhyme@ip-172-31-234-206: ~". The user runs the command "free" which outputs memory usage statistics. The output shows total memory (Mem: 8064072), used memory (used: 607628), free memory (free: 6062512), shared memory (shared: 20208), buffer/cache (buff/cache: 1393932), and available memory (available: 7144152). The swap memory is also listed as 0.



If we want to check if the website is open or not we can type 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

```
Terminal - rhyme@ip-172-31-234-206:~  
File Edit View Terminal Tabs Help  
rhyme@ip-172-31-234-206:~$ htop  
rhyme@ip-172-31-234-206:~$ free  
total used free shared buff/cache available  
Mem: 8064072 607628 6062512 20208 1393932 7144152  
Swap: 0 0 0  
rhyme@ip-172-31-234-206:~$ free -m  
total used free shared buff/cache available  
Mem: 7875 593 5920 19 1361 6976  
Swap: 0 0 0  
rhyme@ip-172-31-234-206:~$ free -g  
total used free shared buff/cache available  
Mem: 7 0 5 0 1 6  
Swap: 0 0 0  
rhyme@ip-172-31-234-206:~$ telnet google.com 80  
Trying 172.217.15.110...  
Connected to google.com.  
Escape character is '^'..  
^T  
telnet> quit  
Connection closed.  
rhyme@ip-172-31-234-206:~$
```

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

```
Connection closed.  
rhyme@ip-172-31-234-206:~$ ping -c3 google.com  
PING google.com (172.217.15.110) 56(84) bytes of data.  
^C  
--- google.com ping statistics ---  
3 packets transmitted, 0 received, 100% packet loss, time 2049ms
```

So like we see above, In this case, it is not reachable.

Let's try it with IP address

```
rhyme@ip-172-31-234-206:~$ ping -c3 127.0.0.1  
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.044 ms  
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.040 ms  
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.036 ms  
  
--- 127.0.0.1 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2036ms  
rtt min/avg/max/mdev = 0.036/0.040/0.044/0.003 ms
```

So, here like we see that the site of this IP address is 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



bash	2748	rhyme	mem	REG	259,1	131	7802	/usr/lib/locale/C.UTF-8/LC_ADDRESS
bash	2748	rhyme	mem	REG	259,1	47	7811	/usr/lib/locale/C.UTF-8/LC_TELEPHONE
bash	2748	rhyme	mem	REG	259,1	23	7806	/usr/lib/locale/C.UTF-8/LC_MEASUREMENT
bash	2748	rhyme	mem	REG	259,1	26376	5012	/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
bash	2748	rhyme	mem	I	REG	259,1	252	7805 /usr/lib/locale/C.UTF-8/LC_IDENTIFICATION
bash	2748	rhyme	0u	CHR	136,0	0t0	3	/dev/pts/0
bash	2748	rhyme	1u	CHR	136,0	0t0	3	/dev/pts/0
bash	2748	rhyme	2u	CHR	136,0	0t0	3	/dev/pts/0
bash	2748	rhyme	255u	CHR	136,0	0t0	3	/dev/pts/0
lsof	2760	rhyme	cwd	DIR	259,1	4096	522782	/home/rhyme
lsof	2760	rhyme	rtd	DIR	259,1	4096	2	/
lsof	2760	rhyme	txt	REG	259,1	163224	4541	/usr/bin/lsof
lsof	2760	rhyme	mem	REG	259,1	144976	2179	/lib/x86_64-linux-gnu/libpthread-2.27.so
lsof	2760	rhyme	mem	REG	259,1	14560	2087	/lib/x86_64-linux-gnu/libdl-2.27.so
lsof	2760	rhyme	mem	REG	259,1	464824	2159	/lib/x86_64-linux-gnu/libpcre.so.3.13.3
lsof	2760	rhyme	mem	REG	259,1	2030544	2084	/lib/x86_64-linux-gnu/libc-2.27.so
lsof	2760	rhyme	mem	REG	259,1	154832	2162	/lib/x86_64-linux-gnu/libselinux.so.1
lsof	2760	rhyme	mem	REG	259,1	170960	2080	/lib/x86_64-linux-gnu/ld-2.27.so
lsof	2760	rhyme	mem	REG	259,1	199772	7804	/usr/lib/locale/C.UTF-8/LC_CTYPE
lsof	2760	rhyme	mem	REG	259,1	1683056	7798	/usr/lib/locale/locale-archive
lsof	2760	rhyme	mem	REG	259,1	26376	5012	/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
lsof	2760	rhyme	0u	CHR	136,0	0t0	3	/dev/pts/0
lsof	2760	rhyme	1u	CHR	136,0	0t0	3	/dev/pts/0
lsof	2760	rhyme	2u	CHR	136,0	0t0	3	/dev/pts/0
lsof	2760	rhyme	3r	DIR	0,4	0	1	/proc
lsof	2760	rhyme	4r	DIR	0,4	0	30857	/proc/2760/fd
lsof	2760	rhyme	5w	FIFO	0,12	0t0	30862	pipe
lsof	2760	rhyme	6r	FIFO	0,12	0t0	30863	pipe
lsof	2761	rhyme	cwd	DIR	259,1	4096	522782	/home/rhyme
lsof	2761	rhyme	rtd	DIR	259,1	4096	2	/
lsof	2761	rhyme	txt	REG	259,1	163224	4541	/usr/bin/lsof
lsof	2761	rhyme	mem	REG	259,1	144976	2179	/lib/x86_64-linux-gnu/libpthread-2.27.so
lsof	2761	rhyme	mem	REG	259,1	14560	2087	/lib/x86_64-linux-gnu/libdl-2.27.so
lsof	2761	rhyme	mem	REG	259,1	464824	2159	/lib/x86_64-linux-gnu/libpcre.so.3.13.3
lsof	2761	rhyme	mem	REG	259,1	2030544	2084	/lib/x86_64-linux-gnu/libc-2.27.so
lsof	2761	rhyme	mem	REG	259,1	154832	2162	/lib/x86_64-linux-gnu/libselinux.so.1
lsof	2761	rhyme	mem	REG	259,1	170960	2080	/lib/x86_64-linux-gnu/ld-2.27.so
lsof	2761	rhyme	mem	REG	259,1	199772	7804	/usr/lib/locale/C.UTF-8/LC_CTYPE
lsof	2761	rhyme	mem	REG	259,1	1683056	7798	/usr/lib/locale/locale-archive
lsof	2761	rhyme	mem	REG	259,1	26376	5012	/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
lsof	2761	rhyme	4r	FIFO	0,12	0t0	30862	pipe
lsof	2761	rhyme	7w	FIFO	0,12	0t0	30863	pipe

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

5. Vmstat

Another command that allow us to monitor is ‘**vmstat**’

```
rhyme@ip-172-31-234-206:~$ vmstat
procs -----memory-----swap--io---system---cpu-----
 r b swpd free buff cache si so bi bo in cs us sy id wa st
 0 0 0 6047712 232124 1174696 0 0 128 33 496 995 2 1 95 2 1
rhyme@ip-172-31-234-206:~$
```

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:*
tcp      0      0 127.0.0.53:53           0.0.0.0:*
tcp      0      0 0.0.0.0:22              0.0.0.0:*
tcp      0      0 127.0.0.1:631           0.0.0.0:*
tcp      0      0 0.0.0.0:36202          0.0.0.0:*
tcp6     0      0 ::1:22                  ::*:*
tcp6     0      0 ::1:631                 ::*:*
tcp6     0      0 :::36202               ::*:*
tcp6     0      0 :::5900                ::*:*
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:*
udp      0      0 224.0.0.251:5353          0.0.0.0:*
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:



```
-rw-r----- 1 root      adm          16528 Apr  3 2020 apport.log.1
drwxr-xr-x  2 root      root         4096 Dec 11 23:55 apt
-rw-r----- 1 syslog     adm          2558 Dec 16 00:35 auth.log
-rw-r----- 1 syslog     adm          31489 Dec 15 23:26 auth.log.1
-rw-r----- 1 syslog     adm          1657 Dec  9 07:30 auth.log.2.gz
-rw-r----- 1 syslog     adm          3696 Sep 16 09:17 auth.log.3.gz
-rw-r----- 1 syslog     adm          1128 Apr 28 2020 auth.log.4.gz
-rw-rw---- 1 root      utmp         0 Dec  9 07:30 btmp
-rw-rw---- 1 root      utmp         0 Sep 16 09:18 btmp.1
-rw-r--r-- 1 root      root         112671 Dec 15 23:25 cloud-init-output.log
-rw-r--r-- 1 syslog     adm         3852063 Dec 15 23:25 cloud-init.log
drwxr-xr-x  2 root      root         4096 Dec 16 00:07 cups
drwxr-xr-x  2 root      root         4096 Nov  7 2019 dist-upgrade
-rw-r--r-- 1 root      root         27505 Dec 11 23:55 dpkg.log
-rw-r--r-- 1 root      root         5441 Dec 11 22:51 dpkg.log.1
-rw-r--r-- 1 root      root         93817 Apr 28 2020 dpkg.log.2.gz
-rw-r--r-- 1 root      root         6052 Apr  7 2020 fontconfig.log
drwxr-xr-x  2 root      root         4096 Feb 19 2019 gdm3
-rw-r--r-- 1 root      root         1235 Dec 15 23:25 gpu-manager.log
drwxr-xr-x  3 root      root         4096 Apr  3 2020 hp
drwxr-sr-x+ 3 root    systemd-journal 4096 Apr  3 2020 journal
-rw-r----- 1 syslog     adm          0 Dec 15 23:30 kern.log
-rw-r----- 1 syslog     adm          263422 Dec 15 23:25 kern.log.1
-rw-r----- 1 syslog     adm          30703 Dec  9 07:25 kern.log.2.gz
-rw-r----- 1 syslog     adm          41192 Sep 16 09:13 kern.log.3.gz
-rw-r----- 1 syslog     adm          20638 Apr 28 2020 kern.log.4.gz
drwxr-xr-x  2 landscape landscape   4096 Apr  3 2020 landscape
-rw-rw-r-- 1 root      utmp         292584 Sep 16 09:45 lastlog
drwxr-xr-x  2 root      root         4096 Dec 15 23:30 lightdm
drwxr-xr-x  2 root      root         4096 Nov 23 2018 lxd
drwx----- 2 speech-dispatcher root  4096 Apr 23 2018 speech-dispatcher
-rw-r----- 1 syslog     adm          46903 Dec 16 00:36 syslog
-rw-r----- 1 syslog     adm          66639 Dec 16 00:07 syslog.1
-rw-r----- 1 syslog     adm          37284 Dec 15 23:30 syslog.2.gz
-rw-r----- 1 syslog     adm          162232 Dec 11 22:53 syslog.3.gz
-rw-r----- 1 syslog     adm          86679 Dec  9 07:30 syslog.4.gz
-rw-r----- 1 syslog     adm          83610 Sep 16 09:18 syslog.5.gz
-rw-r----- 1 syslog     adm          29911 May  5 2020 syslog.6.gz
-rw-r----- 1 syslog     adm          56710 Apr 28 2020 syslog.7.gz
drwxr-xr-x  2 root      root         4096 Jan 17 2020 sysstat
-rw----- 1 root      root         64128 Apr  7 2020 tallylog
drwxr-x--- 2 root    adm          4096 Dec  9 07:30 unattended-upgrades
-rw-rw-r-- 1 root      utmp         21120 Dec 15 23:25 wtmp
-rw-rw-r-- 1 root      utmp         16896 Dec  9 07:25 wtmp.1
rhyme@ip-172-31-234-206:~$
```

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

```
rhyme@ip-172-31-234-206:~$ sudo cat auth.log
Dec 15 23:35:01 ip-172-31-234-206 CRON[2965]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 15 23:35:01 ip-172-31-234-206 CRON[2965]: pam_unix(cron:session): session closed for user root
Dec 15 23:45:01 ip-172-31-234-206 CRON[3010]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 15 23:45:01 ip-172-31-234-206 CRON[3010]: pam_unix(cron:session): session closed for user root
Dec 15 23:54:31 ip-172-31-234-206 dbus-daemon[816]: [system] Failed to activate service 'org.bluez': timed out (service has no .Service file)
Dec 15 23:55:01 ip-172-31-234-206 CRON[3177]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 15 23:55:01 ip-172-31-234-206 CRON[3177]: pam_unix(cron:session): session closed for user root
Dec 15 23:59:01 ip-172-31-234-206 CRON[3216]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 15 23:59:01 ip-172-31-234-206 CRON[3216]: pam_unix(cron:session): session closed for user root
Dec 16 00:01:51 ip-172-31-234-206 sshd[2290]: pam_unix(sshd:session): session closed for user rhyme
Dec 16 00:01:51 ip-172-31-234-206 systemd-logind[814]: Removed session 2.
Dec 16 00:04:32 ip-172-31-234-206 sshd[3242]: Accepted publickey for rhyme from 172.31.15.81 port 50570 ssh2: RSA SHA256:YXKc/N01liH5bPPU
Dec 16 00:04:32 ip-172-31-234-206 sshd[3242]: pam_unix(sshd:session): session opened for user rhyme by (uid=0)
Dec 16 00:04:32 ip-172-31-234-206 systemd-logind[814]: New session 7 of user rhyme.
Dec 16 00:05:01 ip-172-31-234-206 CRON[3329]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 16 00:05:01 ip-172-31-234-206 CRON[3329]: pam_unix(cron:session): session closed for user root
Dec 16 00:15:01 ip-172-31-234-206 CRON[3728]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 16 00:15:01 ip-172-31-234-206 CRON[3728]: pam_unix(cron:session): session closed for user root
Dec 16 00:17:01 ip-172-31-234-206 CRON[3738]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 16 00:17:01 ip-172-31-234-206 CRON[3738]: pam_unix(cron:session): session closed for user root
Dec 16 00:25:01 ip-172-31-234-206 CRON[3778]: pam_unix(cron:session): session opened for user root by (uid=0)
```

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.

```
rhyme@ip-172-31-234-206:/var/log$ sudo tail -f auth.log
Dec 16 00:17:01 ip-172-31-234-206 CRON[3738]: pam_unix(cron:session): session closed for user root
Dec 16 00:25:01 ip-172-31-234-206 CRON[3778]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 16 00:25:01 ip-172-31-234-206 CRON[3778]: pam_unix(cron:session): session closed for user root
Dec 16 00:35:01 ip-172-31-234-206 CRON[3834]: pam_unix(cron:session): session opened for user root by (uid=0)
Dec 16 00:35:02 ip-172-31-234-206 CRON[3834]: pam_unix(cron:session): session closed for user root
Dec 16 00:37:12 ip-172-31-234-206 sudo: rhyme : TTY=pts/0 ; PWD=/var/log ; USER=root ; COMMAND=/bin/cat auth.log
Dec 16 00:37:12 ip-172-31-234-206 sudo: pam_unix(sudo:session): session opened for user root by (uid=0)
Dec 16 00:37:12 ip-172-31-234-206 sudo: pam_unix(sudo:session): session closed for user root
Dec 16 00:38:20 ip-172-31-234-206 sudo: rhyme : TTY=pts/0 ; PWD=/var/log ; USER=root ; COMMAND=/usr/bin/tail -f au
Dec 16 00:38:20 ip-172-31-234-206 sudo: pam_unix(sudo:session): session opened for user root by (uid=0)
```

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.

Most 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)
```



```
rhymer@ip-172-31-234-206:~$ bash <(curl -sS https://my-netdata.io/kickstart.sh)
System          : Linux
Operating System : GNU/Linux
Machine         : x86_64
BASH major version:
--- Fetching script to detect required packages... ---
[~/tmp/netdata-kickstart-rmfjXrdey9]$ curl -q -sSL --connect-timeout 10 --retry 3 --output /tmp/netdata-kickstart-rmfjXrdey9/install-required-packages.sh https://raw.githubusercontent.com/netdata/netdata/master/packaging/installer/install-required-packages.sh OK

--- Running downloaded script to detect required packages... ---
[~/tmp/netdata-kickstart-rmfjXrdey9]$ sudo /bin/bash /tmp/netdata-kickstart-rmfjXrdey9/install-required-packages.sh netdata Loading /etc/os-release ...
/etc/os-release information:
NAME      : Ubuntu
VERSION   : 18.04.4 LTS (Bionic Beaver)
ID        : ubuntu
ID_LIKE   : debian
VERSION_ID: 18.04

We detected these:
Distribution : ubuntu
Version     : 18.04
Codename    : 18.04.4 LTS (Bionic Beaver)
Package Manager: install_apt_get
Packages Tree : debian
Detection Method: /etc/os-release
Default Python v: 2 (will install python3 too)

Searching for distro_sdk ...
Searching for autoconf archive ...
> Checking if package 'autoconf-archive' is installed...
Searching for autogen ...
> Checking if package 'autogen' is installed...
dpkg-query: no packages found matching autogen
Searching for cmake ...
> Checking if package 'cmake' is installed...
Searching for libbz dev ...
> Checking if package 'zlib1g-dev' is installed...
Searching for libuuid dev ...
> Checking if package 'uuid-dev' is installed...

netdata by default listens on all IPs on port 19999,
so you can access it with:

http://this.machine.ip:19999/

To stop netdata run:

systemctl stop netdata

To start netdata run:

systemctl start netdata

Uninstall script copied to: /usr/libexec/netdata/netdata-uninstaller.sh

--- Installing (but not enabling) the netdata updater tool ---
Failed to disable unit: Unit file netdata-updater.timer does not exist.
Update script is located at /usr/libexec/netdata/netdata-updater.sh

--- Check if we must enable/disable the netdata updater tool ---
Auto-updating has been enabled through cron, updater script linked to /etc/cron.daily/netdata-updater

If the update process fails and you have email notifications set up correctly for cron on this system, you should re-
of the failure.
Successful updates will not send an email.

--- Wrap up environment set up ---
Preparing .environment file
[~/tmp/netdata-kickstart-rmfjXrdey9/netdata-v1.27.0-4-g209c099e]# chmod 0644 /etc/netdata/.environment
OK

Setting netdata.tarball.checksum to 'new_installation'

--- We are done! ---

^
+-----+-----+-----+-----+-----+-----+-----+-----+
| netdata                                netdata
|           is installed and running now! |
+-----+-----+-----+-----+-----+-----+-----+-----+
enjoy real-time performance and health monitoring...

OK
```

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

To start Netdata run: `#systemctl start netdata`



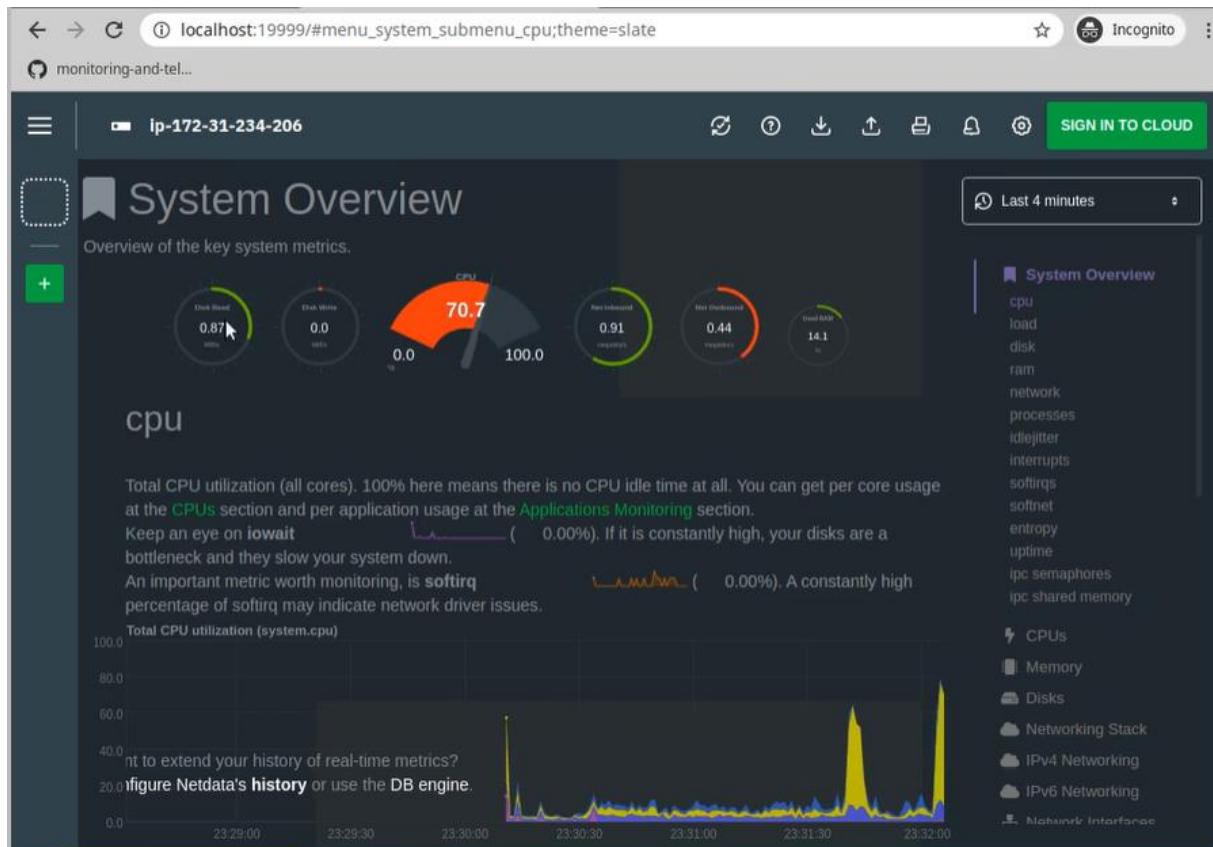
To update netdata execute this script: `/usr/libexec/netdata-updater.sh`

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

2. Accessing Netdata:

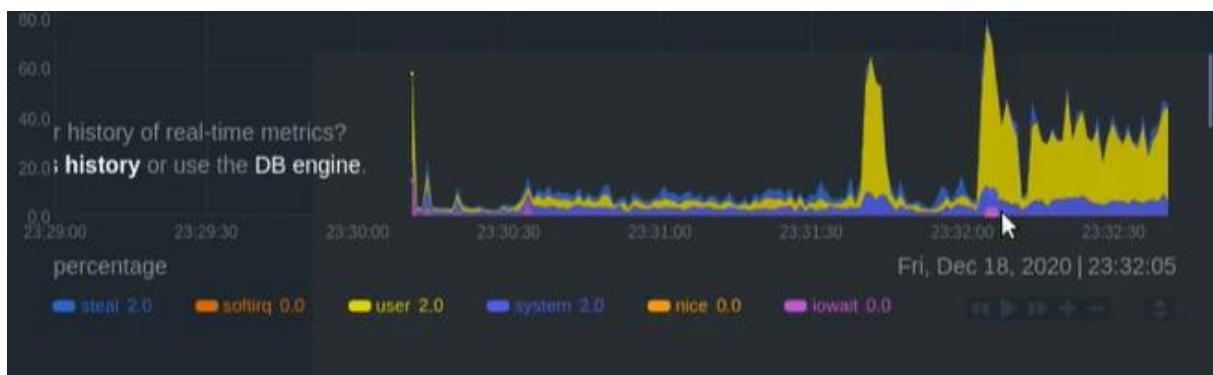
1 . Open a web browser on your local computer, for example Chrome or Firefox.

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.

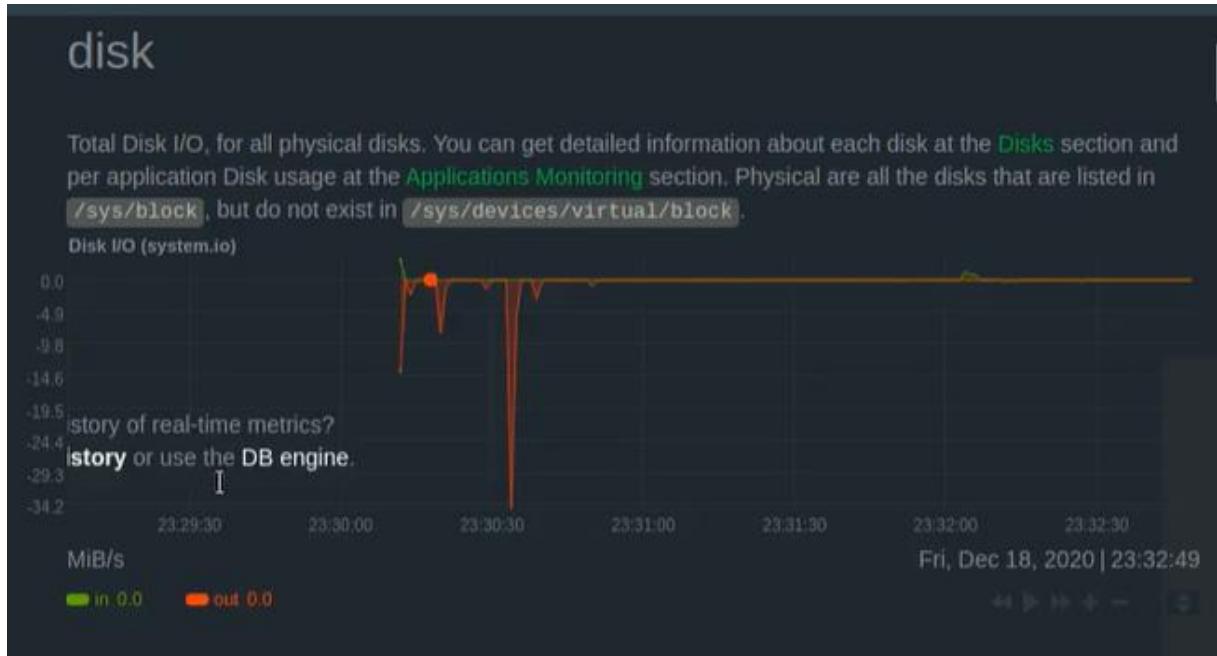
And this the real time graph of the cpu usage:



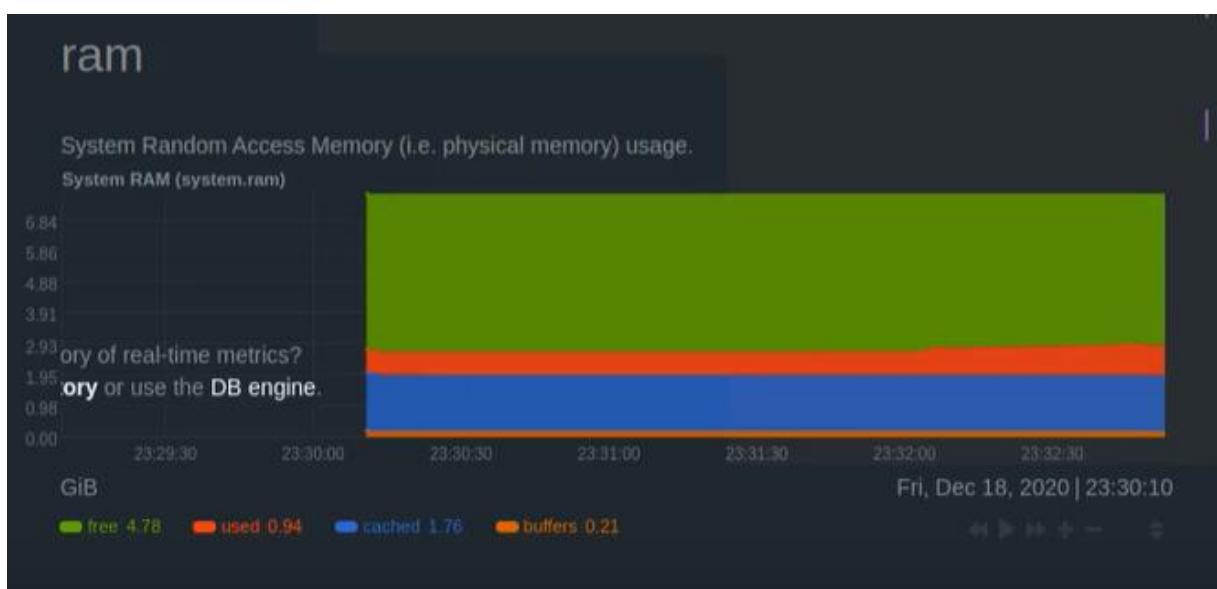
And the real time graph of the load average time:



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 bandwidth being consumed:



network

Total bandwidth of all physical network interfaces. This does not include `lo`, VPNs, network bridges, IFB devices, bond interfaces, etc. Only the bandwidth of physical network interfaces is aggregated. Physical are all the network interfaces that are listed in `/proc/net/dev`, but do not exist in `/sys/devices/virtual/net`.

Physical Network Interfaces Aggregated Bandwidth (system.net)



Total IP traffic in the system.

IP Bandwidth (system.ip)

We have also a presentation of the system processes running:

processes

System processes. **Running** are the processes in the CPU. **Blocked** are processes that are willing to enter the CPU, but they cannot, e.g. because they wait for disk activity.

System Processes (system.processes)



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 nginx container:

```
#sudo docker run -d nginx
```

```
rhymer@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
70acd795e79: Pull complete
Digest: sha256:4cf620a5c81390ee209398ecc18e5fb9dd0f5155cd82adcbae532fec94006fb9
Status: Downloaded newer image for nginx:latest
e915dc6f7cb932ee7180cef93488829f5bef735fc55319e5252e0505032cba33
```



And, for verify we will type: #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:

```
rhymer@ip-172-31-234-206:~$ sudo docker run -d nginx
4a0c85902cb91f15f733b96528b5923bdd5665ac237cc881e16551269754ee25
rhymer@ip-172-31-234-206:~$ sudo docker run -d nginx
ab216012caa05f38929477c123890d45df8f148c4c9c423564f8c40feeee7a419
```

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

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_nightingale
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:

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

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
```



```
rhymer@ip-172-31-234-206:~$ VERSION=v0.36.0 # use the latest release version from https://github.com/google/cadvisor/releases
rhymer@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
24777f8fc15bd79a6a547d2723349657b8af7204ba6bbb0cfef2224b2d7aeb844
```

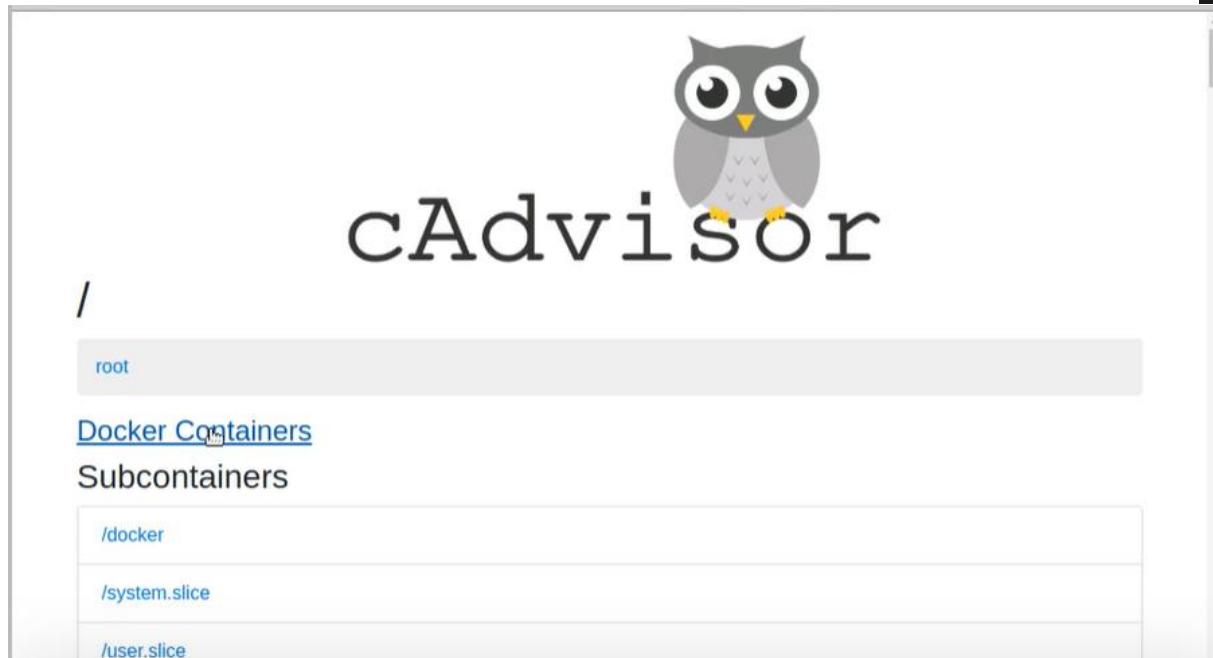
On typing #sudo docker ps we can see the container was created and it is in a starting state (health:starting):

CONTAINER ID	IMAGE NAMES	COMMAND	CREATED	STATUS	PORTS
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	"/docker-entrypoint..."	4 minutes ago	Up 4 minutes	80/tcp
ab216012caa0	nginx	"/docker-entrypoint..."	4 minutes ago	Up 4 minutes	80/tcp
4a0ec85902cb9	gifted_sanderson	"/docker-entrypoint..."	4 minutes ago	Up 4 minutes	80/tcp
e915dc6f7cb9	hopeful_nightingale	"/docker-entrypoint..."	4 minutes ago	Up 4 minutes	80/tcp
	inspiring_dirac	"/docker-entrypoint..."	4 minutes ago	Up 4 minutes	80/tcp

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

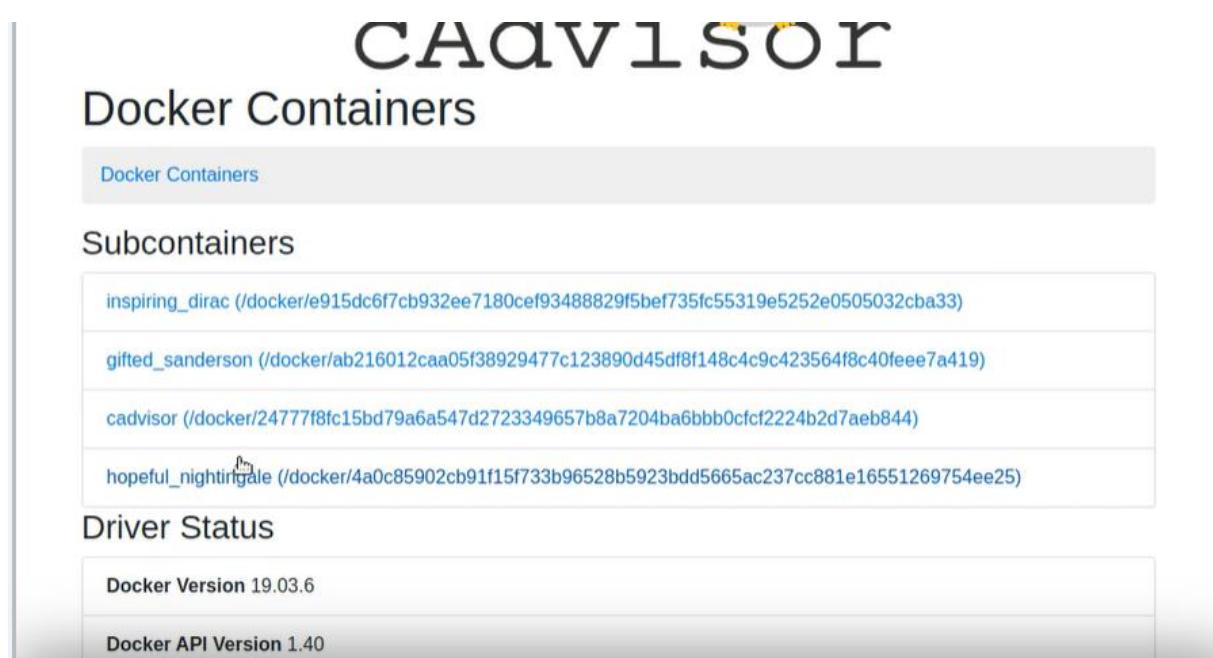
The screenshot shows the cAdvisor interface. At the top, there's a navigation bar with back, forward, and search icons, and an 'Incognito' button. The main content area has a large owl logo and the word "cAdvisor". Below the logo, there's a sidebar with a "root" tab selected. Under "Docker Containers", there are three sub-containers listed: "/docker", "/system.slice", and "/user.slice". At the bottom, there's an "Isolation" section with a progress bar and some status indicators like "18:44 / 21:06". The URL in the address bar is "localhost:8080/containers/".

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



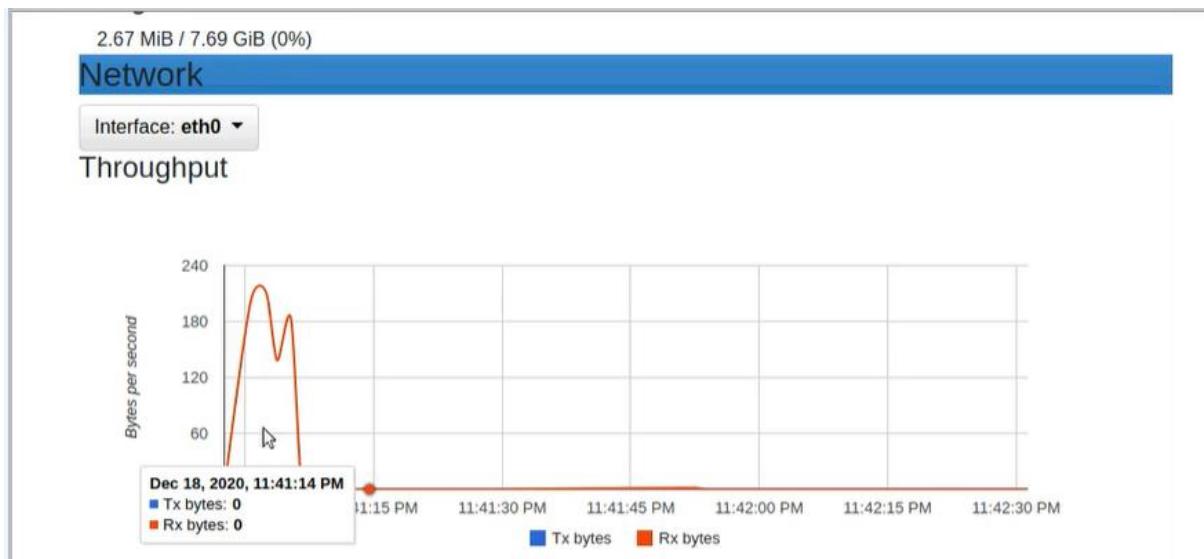
The screenshot shows the cAdvisor interface. At the top is a cartoon owl logo. Below it, the word "cAdvisor" is written in a large, lowercase, sans-serif font. A vertical slash character is positioned to the left of the main content area. The main content area has a light gray header bar with the word "root". Below this, the title "Docker Containers" is underlined in blue. A section titled "Subcontainers" follows, containing a list of paths: "/docker", "/system.slice", and "/user.slice".

We can see the four containers is running:



The screenshot shows the CAAVISOR interface. The title "CAAVISOR" is at the top, followed by "Docker Containers". A header bar contains the link "Docker Containers". Below this is a section titled "Subcontainers" listing four containers: "inspiring_dirac (/docker/e915dc6f7cb932ee7180cef93488829f5bef735fc55319e5252e0505032cba33)", "gifted_sanderson (/docker/ab216012caa05f38929477c123890d45df8f148c4c9c423564f8c40feee7a419)", "cadvisor (/docker/24777f8fc15bd79a6a547d2723349657b8a7204ba6bbb0cfef2224b2d7aeb844)", and "hopeful_nightingale (/docker/4a0c85902cb91f15f733b96528b5923bdd5665ac237cc881e16551269754ee25)". Below this is a section titled "Driver Status" with two entries: "Docker Version 19.03.6" and "Docker API Version 1.40".

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 as applications which are 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 microservice 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 or any sort of other errors. So, that way, checking the health of microservices in 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)`

```
rhymer@ip-172-31-234-206:~$ sudo docker rm -f $(sudo docker ps -a -q)
"docker rm" requires at least 1 argument.
See 'docker rm --help'.

Usage: docker rm [OPTIONS] CONTAINER [CONTAINER...]
Remove one or more containers
```



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

And now, Let's install Spring Boot:

The screenshot shows the Spring Initializr web application. It has a sidebar on the left with a menu icon and a search bar. The main area is titled "spring initializr". On the left, there are sections for "Project" (Maven Project selected), "Language" (Java selected), "Spring Boot" (2.4.1 selected), and "Project Metadata" (Group: com.example, Artifact: demo). On the right, there is a "Dependencies" section with a button labeled "ADD ... CTRL + B" and a message "No dependency selected". A small icon in the top right corner shows the sun and moon, indicating a dark mode setting.

And we gonna add some dependencies:

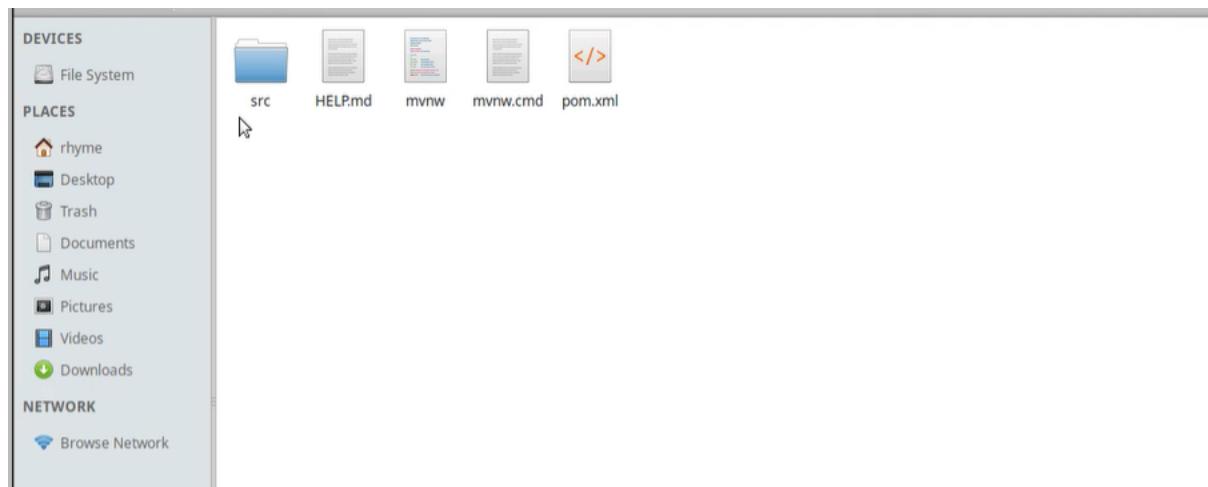
The screenshot shows the Spring Initializr interface after adding dependencies. The "Dependencies" section now includes "Spring Web" (selected) and "WEB". Below it, a description states: "Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container." Further down, the "Spring Boot DevTools" section is shown with a "DEVELOPER TOOLS" button, described as "Provides fast application restarts, LiveReload, and configurations for enhanced development experience."

Now, let's click on GENERATE:



The screenshot shows the IntelliJ IDEA interface for creating a new project. On the left, there's a sidebar with icons for GitHub and Twitter. The main area has sections for 'Project' (Maven Project selected), 'Language' (Java selected), 'Dependencies' (Spring Web selected), and 'Project Metadata' (Group: com.example, Artifact: demo, Name: demo, Description: Demo project for Spring Boot, Package name: com.example.demo). At the bottom are buttons for 'GENERATE' (CTRL + SHIFT + A), 'EXPLORE' (CTRL + SPACE), and 'SHARE...'. The 'Dependencies' section also includes a 'WEB' button and a 'Spring Boot DevTools' section.

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**:

```
① application.properties ②
home > rhyme > Downloads > demo > src > main > resources > ③ 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**

```
rhymer@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 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-builtins-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-validatenpm-package-license node-which node-wrapify 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**



```
rhyme@ip-172-31-234-206:~/Downloads/demo$ mvn clean install
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.inject.internal.cglib.core.$ReflectUtils$1 (file:/usr/share/maven/lib/guice.jar) to method java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int,java.security.ProtectionDomain)
WARNING: Please consider reporting this to the maintainers of com.google.inject.internal.cglib.core.$ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Scanning for projects...
[INFO]
[INFO] -----> com.example:demo <-----
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] -----*[ jar ]-----*
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ demo ---
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO] The encoding used to copy filtered properties files have not been set. This means that the same encoding will be used to copy filtered properties files as when copying other filtered resources. This might not be what you want! Run your build with --debug to see which files might be affected. Read more at https://maven.apache.org/plugins/maven-resources-plugin/examples/filtering-properties-files.html
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ demo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /home/rhyme/Downloads/demo/target/classes
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory /home/rhyme/Downloads/demo/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ demo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /home/rhyme/Downloads/demo/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ demo ---
[INFO]
[INFO] -----*
[INFO] T E S T S
[INFO] -----*
```

```
^__^
(oo)\______)
  (__)\       )\/\
   (__)\    )\/\
    (__)\_ )\/\
     (__)\_\_(
      :: Spring Boot ::           (v2.4.1)

2020-12-19 21:14:19.533  INFO 5987 --- [           main] com.example.demo.DemoApplicationTests : Starting DemoApplicationTests using Java 11
8.7 on ip-172-31-234-206 with PID 5987 (started by rhyme in /home/rhyme/Downloads/demo)
2020-12-19 21:14:19.538  INFO 5987 --- [           main] com.example.demo.DemoApplicationTests : No active profile set, falling back to default profiles: default
2020-12-19 21:14:22.089  INFO 5987 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-12-19 21:14:22.747  INFO 5987 --- [           main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 0 endpoint(s) beneath base path '/actuator'
2020-12-19 21:14:22.882  INFO 5987 --- [           main] com.example.demo.DemoApplicationTests : Started DemoApplicationTests in 3.903 seconds (JVM running for 5.502)
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.152 s - in com.example.demo.DemoApplicationTests
2020-12-19 21:14:23.445  INFO 5987 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor : Shutting down ExecutorService 'applicationTaskExecutor'
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ demo ---
[INFO] Building jar: /home/rhyme/Downloads/demo/target/demo-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.1:repackage (repackage) @ demo ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ demo ---
[INFO] Installing /home/rhyme/Downloads/demo/target/demo-0.0.1-SNAPSHOT.jar to /home/rhyme/.m2/repository/com/example/demo/0.0.1-SNAPSHOT/demo-0.0.1-SNAPSHOT.jar
[INFO] Installing /home/rhyme/Downloads/demo/pom.xml to /home/rhyme/.m2/repository/com/example/demo/0.0.1-SNAPSHOT/demo-0.0.1-SNAPSHOT.pom
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 11.912 s
[INFO] Finished at: 2020-12-19T21:14:24Z
[INFO]
```

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.



```
rhyme@ip-172-31-234-206:~/Downloads/demo$ ls -l
total 36
-rw-r--r-- 1 rhyme rhyme 1005 Dec 19 21:01 HELP.md
-rwxr-xr-x 1 rhyme rhyme 10070 Dec 19 21:01 mvnw
-rw-r--r-- 1 rhyme rhyme 6608 Dec 19 21:01 mvnw.cmd
-rw-r--r-- 1 rhyme rhyme 1737 Dec 19 21:04 pom.xml
drwxr-xr-x 4 rhyme rhyme 4096 Dec 19 21:01 src
drwxrwxr-x 9 rhyme rhyme 4096 Dec 19 21:14 target
rhyme@ip-172-31-234-206:~/Downloads/demo$ cd target/
rhyme@ip-172-31-234-206:~/Downloads/demo/target$ ls -l
total 18608
drwxrwxr-x 3 rhyme rhyme 4096 Dec 19 21:14 classes
-rw-rw-r-- 1 rhyme rhyme 19019116 Dec 19 21:14 demo-0.0.1-SNAPSHOT.jar
-rw-rw-r-- 1 rhyme rhyme 2795 Dec 19 21:14 demo-0.0.1-SNAPSHOT.jar.original
drwxrwxr-x 3 rhyme rhyme 4096 Dec 19 21:14 generated-sources
drwxrwxr-x 3 rhyme rhyme 4096 Dec 19 21:14 generated-test-sources
drwxrwxr-x 2 rhyme rhyme 4096 Dec 19 21:14 maven-archiver
drwxrwxr-x 3 rhyme rhyme 4096 Dec 19 21:14 maven-status
drwxrwxr-x 2 rhyme rhyme 4096 Dec 19 21:14 surefire-reports
drwxrwxr-x 3 rhyme rhyme 4096 Dec 19 21:14 test-classes
```

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

```
rhyme@ip-172-31-234-206:~/Downloads/demo/target$ java -jar demo-0.0.1-SNAPSHOT.jar
. . . . .
:: Spring Boot :: (v2.4.1)
2020-12-19 21:15:32.686 INFO 6122 --- [           main] com.example.demo.DemoApplication      : Starting DemoApplication v0.0.1-SNAPSHOT using Java 11.0.7 on ip-172-31-234-206 with PID 6122 (/home/rhyme/Downloads/demo/target/demo-0.0.1-SNAPSHOT.jar started by rhyme in /home/rhyme/Downloads/demo/target)
2020-12-19 21:15:32.698 INFO 6122 --- [           main] com.example.demo.DemoApplication      : No active profile set, falling back to default profiles: default
2020-12-19 21:15:35.134 INFO 6122 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-12-19 21:15:35.153 INFO 6122 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-12-19 21:15:35.154 INFO 6122 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.41]
2020-12-19 21:15:35.267 INFO 6122 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]      : Initializing Spring embedded WebApplication Context
2020-12-19 21:15:35.267 INFO 6122 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2408 ms
2020-12-19 21:15:35.988 INFO 6122 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-12-19 21:15:36.473 INFO 6122 --- [           main] o.s.b.a.e.web.EndpointLinksResolver    : Exposing 1 endpoint(s) beneath base path '/actuator'
2020-12-19 21:15:36.630 INFO 6122 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
```

Like 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:



A screenshot of a web browser window. The address bar shows 'localhost:8080'. Below it, a grey bar contains a circular icon with a 'G' and the text 'monitoring-and-tel...'. The main content area has a dark grey header with the text 'Whitelabel Error Page' in white. Below this, a message in black text says 'This application has no explicit mapping for /error, so you are seeing this as a fallback.' Underneath, it shows the timestamp 'Sat Dec 19 21:16:11 UTC 2020' and the error message 'There was an unexpected error (type=Not Found, status=404)'.

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

A screenshot of a web browser window. The address bar shows 'localhost:8080/actuator'. Below it, a grey bar contains a circular icon with a 'G' and the text 'monitoring-and-tel...'. The main content area displays a JSON object: {"links": [{"self": {"href": "http://localhost:8080/actuator", "templated": false}, "prometheus": {"href": "http://localhost:8080/actuator/prometheus", "templated": false}}]}

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

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



```
localhost:8080/actuator/prometheus
monitoring-andtel...

# HELP logback_events_total Number of error level events that made it to the logs
# TYPE logback_events_total counter
logback_events_total{application="MonitoringSpringDemoProject",level="debug",} 0.0
logback_events_total{application="MonitoringSpringDemoProject",level="warn",} 0.0
logback_events_total{application="MonitoringSpringDemoProject",level="info",} 7.0
logback_events_total{application="MonitoringSpringDemoProject",level="error",} 0.0
logback_events_total{application="MonitoringSpringDemoProject",level="trace",} 0.0
# HELP jvm_classes_loaded_classes The number of classes that are currently loaded in the Java virtual machine
# TYPE jvm_classes_loaded_classes gauge
jvm_classes_loaded_classes{application="MonitoringSpringDemoProject",} 7185.0
# HELP jvm_threads_states_threads The current number of threads having NEW state
# TYPE jvm_threads_states_threads gauge
jvm_threads_states_threads{application="MonitoringSpringDemoProject",state="waiting",} 11.0
jvm_threads_states_threads{application="MonitoringSpringDemoProject",state="new",} 0.0
jvm_threads_states_threads{application="MonitoringSpringDemoProject",state="timed-waiting",} 3.0
jvm_threads_states_threads{application="MonitoringSpringDemoProject",state="blocked",} 0.0
jvm_threads_states_threads{application="MonitoringSpringDemoProject",state="terminated",} 0.0
jvm_threads_states_threads{application="MonitoringSpringDemoProject",state="runnable",} 7.0
# HELP tomcat_sessions_expired_sessions_total
# TYPE tomcat_sessions_expired_sessions_total counter
tomcat_sessions_expired_sessions_total{application="MonitoringSpringDemoProject",} 0.0
# HELP system_load_average_1m The sum of the number of runnable entities queued to available processors and the number of
runnable entities running on the available processors averaged over a period of time
# TYPE system_load_average_1m gauge
system_load_average_1m{application="MonitoringSpringDemoProject",} 0.22
# HELP tomcat_sessions_alive_max_seconds
# TYPE tomcat_sessions_alive_max_seconds gauge
tomcat_sessions_alive_max_seconds{application="MonitoringSpringDemoProject",} 0.0
# HELP jvm_gc_pause_seconds Time spent in GC pause
# TYPE jvm_gc_pause_seconds summary
jvm_gc_pause_seconds_count{action="end of minor GC",application="MonitoringSpringDemoProject",cause="G1 Evacuation Pause",} 1.0
jvm_gc_pause_seconds_sum{action="end of minor GC",application="MonitoringSpringDemoProject",cause="G1 Evacuation Pause",} 0.033
# HELP jvm_gc_pause_seconds_max Time spent in GC pause
# TYPE jvm_gc_pause_seconds_max gauge
jvm_gc_pause_seconds_max{action="end of minor GC",application="MonitoringSpringDemoProject",cause="G1 Evacuation Pause",} 0.033
```

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.



```
rhymer@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`

```
[root@ip-172-31-234-206 ~]# code /tmp/prometheus.yml
```

```
! prometheus.yml ●
! prometheus.yml
1   global:
2     |   scrape_interval:      15s # By default, scrape targets every 15 seconds. ①
3
4   rule_files:
5     |   # - "first.rules"
6     |   # - "second.rules" ②
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'] ③
```

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
```

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

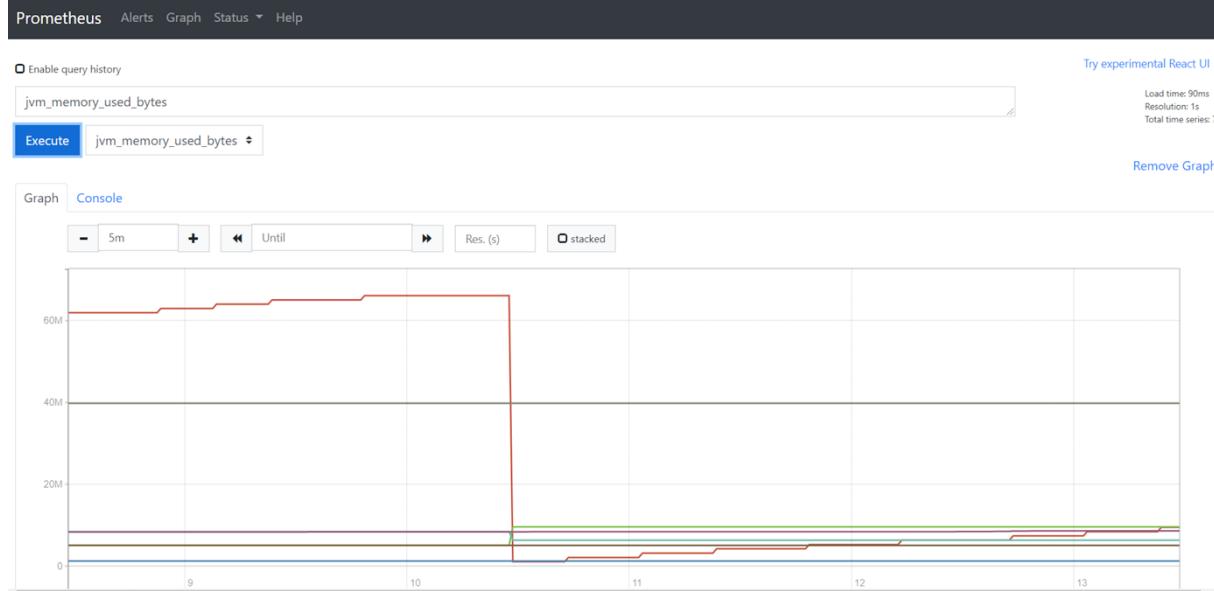
On running the command: **#sudo docker ps**, we got that the container is running on 9090 port

```
rhymer@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.

The screenshot shows the Prometheus WebUI interface. At the top, there's a navigation bar with links for Prometheus, Alerts, Graph, Status, Help, and Classic UI. Below the navigation bar, there are three checkboxes: 'Enable query history' (unchecked), 'Use local time' (unchecked), and 'Enable autocomplete' (checked). A search bar contains the placeholder 'Expression (press Shift+Enter for newlines)' with a blue 'Execute' button to its right. Below the search bar, there are two tabs: 'Table' (disabled) and 'Graph' (selected). Under the 'Graph' tab, there's a 'Evaluation time' input field with arrows for adjusting the time range. The main area displays the message 'No data queried yet'. At the bottom left, there's a blue 'Add Panel' button, and at the bottom right, a blue 'Remove Panel' link.



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**

← → ⓘ localhost:9090/targets ☆ Incognito :

monitoring-and-tel...

Prometheus Alerts Graph Status ▾ Help Classic UI

Targets

All Unhealthy

prometheus (1/1 up) show less

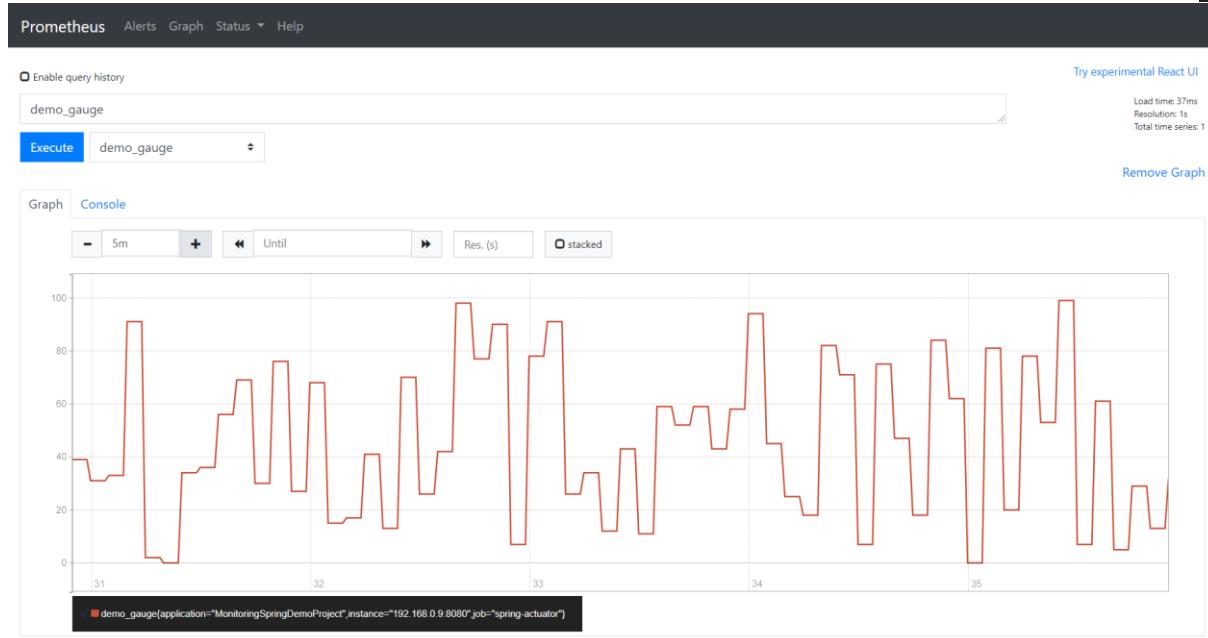
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	3.603s	4.291ms	

spring-boot-starter (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://172.31.234.206:8080/actuator/prometheus	UP	instance="172.31.234.206:8080" job="spring-boot-starter"	4.952s	5.827ms	

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`

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

So here the grafana container was launched.



```
rhymer@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.

localhost:3000/login

Welcome to Grafana

Don't get in the way of the data

Email or username

Password

Log in

Forgot your password?

Documentation | Support | Community | Open Source | v7.3.5 (11f305f88a) | New version available

localhost:3000/datasources

monitoring-and-tel...

Home

Basic

The steps below will guide you to quickly up your llation.

TUTORIAL

DATA SOURCES AND DASHBOARDS

Grafana fundamentals

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

Remove this panel

DATA SOURCES

Add your first data source

Learn how in the docs

Latest from the blog



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:

A screenshot of the Grafana configuration interface. The URL in the address bar is 'localhost:3000/datasources'. The main title is 'Configuration' with 'Organization: Main Org.' below it. A sidebar on the left contains icons for Data Sources, Users, Teams, Plugins, Preferences, and API Keys. The main content area shows a message 'There are no data sources defined yet' and a prominent blue button labeled 'Add data source'. Below the button is a pro tip: 'ProTip: You can also define data sources through configuration files. [Learn more](#)'. At the bottom of the screen, there are links for Documentation, Support, Community, Open Source, and a note about a new version available.

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

A screenshot of the 'Add data source' dialog. The URL in the address bar is 'localhost:3000/datasources/new'. The title is 'Add data source' with the sub-instruction 'Choose a data source type'. A green success message box says '✓ Datasource added'. Below it is a search bar with the placeholder 'Filter by name or type' and a 'Cancel' button. The main section is titled 'Time series databases' and lists three options: 'Prometheus' (selected), 'Graphite', and 'OpenTSDB'. Each option has a small icon, a brief description, and a 'Select' button. The 'Prometheus' option is highlighted with a blue border.

And now we configure the data sources:



localhost:3000/datasources/edit/1/ monitoring-and-tel...

Data Sources / Prometheus

Type: Prometheus

Settings Dashboards

Name: Prometheus Default: On

HTTP

URL: http://localhost:9090 Access: Browser

Auth

Basic auth: Off With Credentials: Off

Scrape interval: 15s

Incognito

localhost:3000/datasources/edit/1/ monitoring-and-tel...

URL: http://localhost:9090 Access: Browser ✓ Datasource updated

Auth

Basic auth: Off With Credentials: Off

Scrape interval: 15s Query timeout: 60s HTTP Method: Choose

Misc

Disable metrics lookup: Off Custom query parameters: Example: max_source_resolution=5m&timeout=10

Data source is working

Incognito

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



localhost:3000/dashboard/import

monitoring-and-tel...

Import

Import dashboard from file or Grafana.com

Create

ON file

Dashboard

Folder

Grafana.com

Import

Grafana.com dashboard url or id

Load

Import via panel json

?

google.com/search?q=grafana+dashboards&oq=grafana+dashboards&aqs=chrome..69i57.4893j0j1&s...

monitoring-and-tel...

grafana dashboards

All Images Videos News Shopping More Settings Tools

About 1,660,000 results (0.53 seconds)

grafana.com › dashboards

[Grafana Dashboards - discover and share dashboards for ...](#)

Grafana.com provides a central repository where the community can come together to discover and share dashboards.

Discover and share ...
Grafana.com provides a central repository where the community ...

Share dashboards
Grafana.com provides a central repository where the community ...

Dashboard overview
Dashboard overview A dashboard is a set of one or more panels ...

Kubernetes dashboard
Part 3 of 3 dashboards to help you visualise your kubernetes costs.

Dashboards
Grafana.com provides a central repository where the community ...

Telegraf
Inspiration taken from <https://grafana.com/grafana ...>

[More results from grafana.com »](#)

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



grafana.com/grafana/dashboards

monitoring-and-tel...

Grafana Labs Grafana Products Open Source Learn Downloads Login Contact us

Features Contribute Dashboards Plugins Download

Dashboards

Official & community built dashboards

The dashboard displays various metrics such as 144k requests per hour, 2.619k unique visitors, and a map showing request distribution across continents. It also includes a log viewer showing NGINX access logs.

<https://grafana.com/grafana/dashboards/12559>

Loki v2 Web Analytics Dashboard
LOKI
Loki version 2 showcase using JSON NGINX access logs.

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

grafana.com/grafana/dashboards

monitoring-and-tel...

Features Contribute Dashboards Plugins Download

Node Exporter Full
PROMETHEUS
Nearly all default values exported by Prometheus node exporter graphed. Only requires the default job_name: node, add as many targets as you need...

AWS Billing
CLOUDWATCH
Visualize estimated AWS charges per AWS resource (EC2, S3, ...)

NGINX Ingress controller
PROMETHEUS
Ingress-nginx supports a rich collection of prometheus metrics. If you have prometheus and grafana installed on your cluster then prometheus will...

<https://grafana.com/grafana/dashboards/139>

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



grafana.com/grafana/dashboards/9614

monitoring-and-tel...

All dashboards » NGINX Ingress controller

NGINX Ingress controller by gonzalesraul

DASHBOARD

Ingress-nginx supports a rich collection of prometheus metrics. If you have prometheus and grafana installed on your cluster then prometheus will already be scraping this data due to the scrape annotation on the deployment.

Last updated: 2 years ago

Downloads: 14421256

Reviews: 3

★ ★ ★ ★ ★

Add your review!

Overview Revisions Reviews

Get this dashboard:

SOURCE: <https://github.com/kubernetes/ingress-nginx/tree/master/deploy/grafana/dashboards>

Grafana Dashboards

Ingress-nginx supports a rich collection of prometheus metrics. If you have prometheus and grafana installed on your cluster then prometheus will already be scraping this data due to the `scrape`

9614

Copy ID to Clipboard

Download JSON

How do I import this

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:

localhost:3000/dashboard/import

monitoring-and-tel...

Import

Import dashboard from file or Grafana.com

Upload JSON file

Import via grafana.com

4701 **Load**

Import via panel json

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



localhost:3000/dashboard/import

monitoring-and-tel...

Updated on 2019-11-03 17:00:25

Options

Name: JVM (Micrometer)

Folder: General

Unique identifier (uid)
The unique identifier (uid) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The uid allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

Change uid

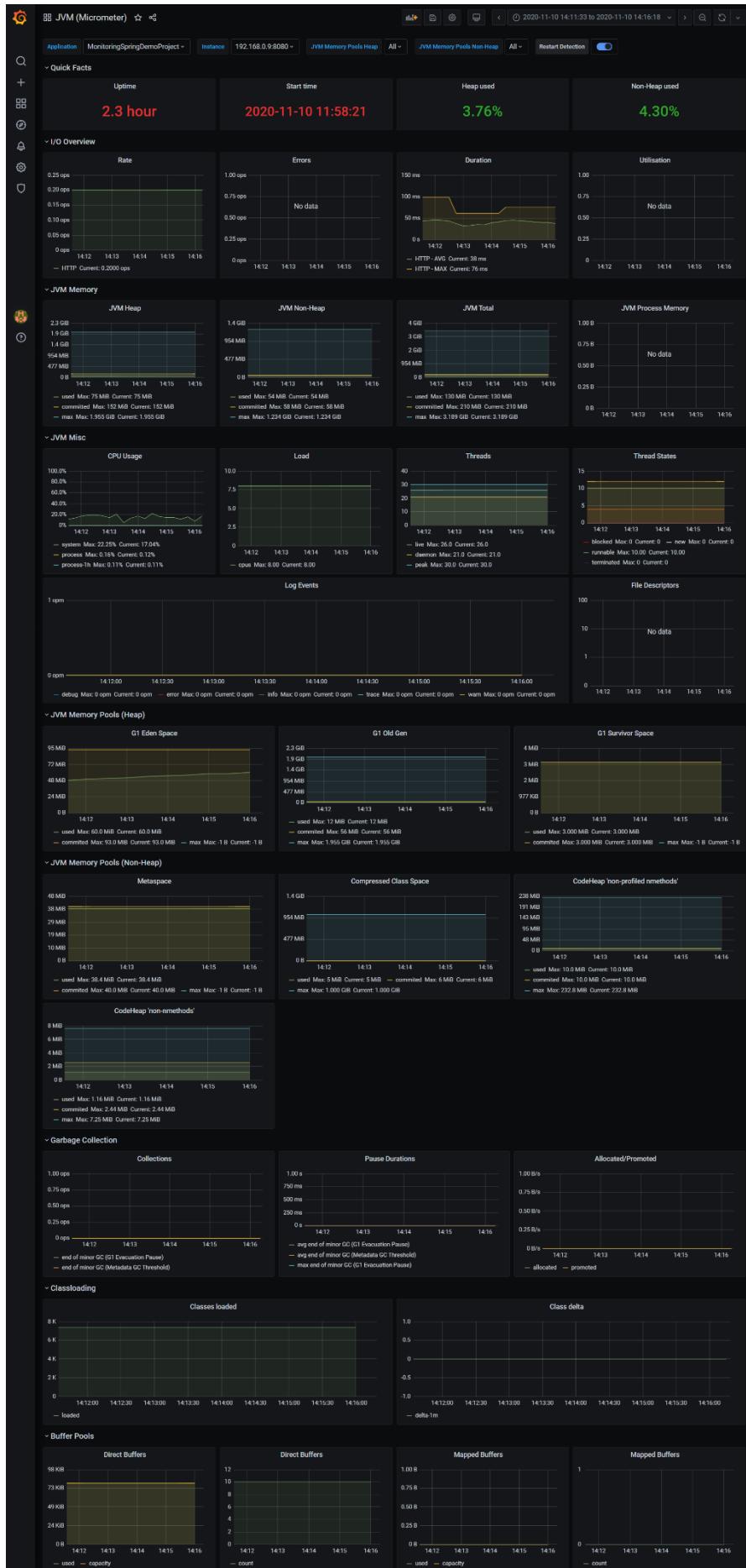
Prometheus: Prometheus

Import Cancel

Documentation | Support | Community | Open Source | v7.3.5 (11f305f88a) | New version available!

This screenshot shows the 'Import' dialog in Grafana. It includes fields for 'Name' (set to 'JVM (Micrometer)'), 'Folder' (set to 'General'), and a 'Unique identifier (uid)' section with a 'Change uid' button. Below these are Prometheus and Grafana settings. At the bottom are 'Import' and 'Cancel' buttons, and a footer with links to documentation, support, community, open source, and a note about a new version available.

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





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, Logstash 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.

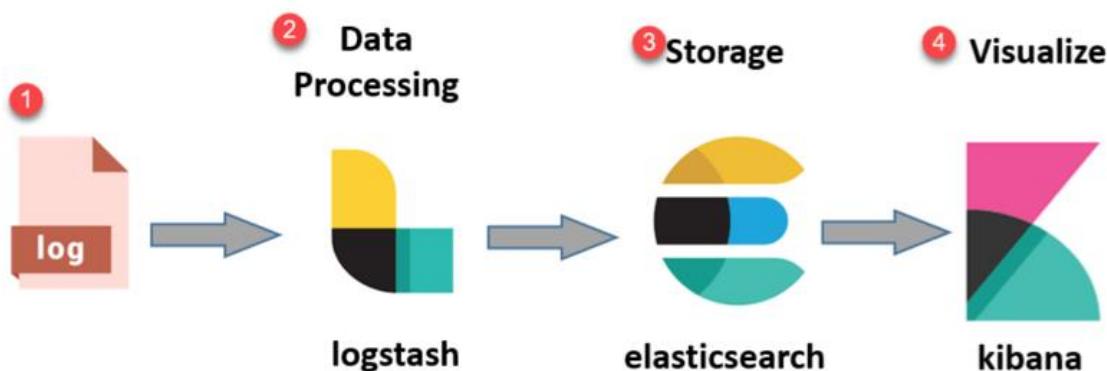
E stands for ElasticSearch: used for storing logs

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)
```

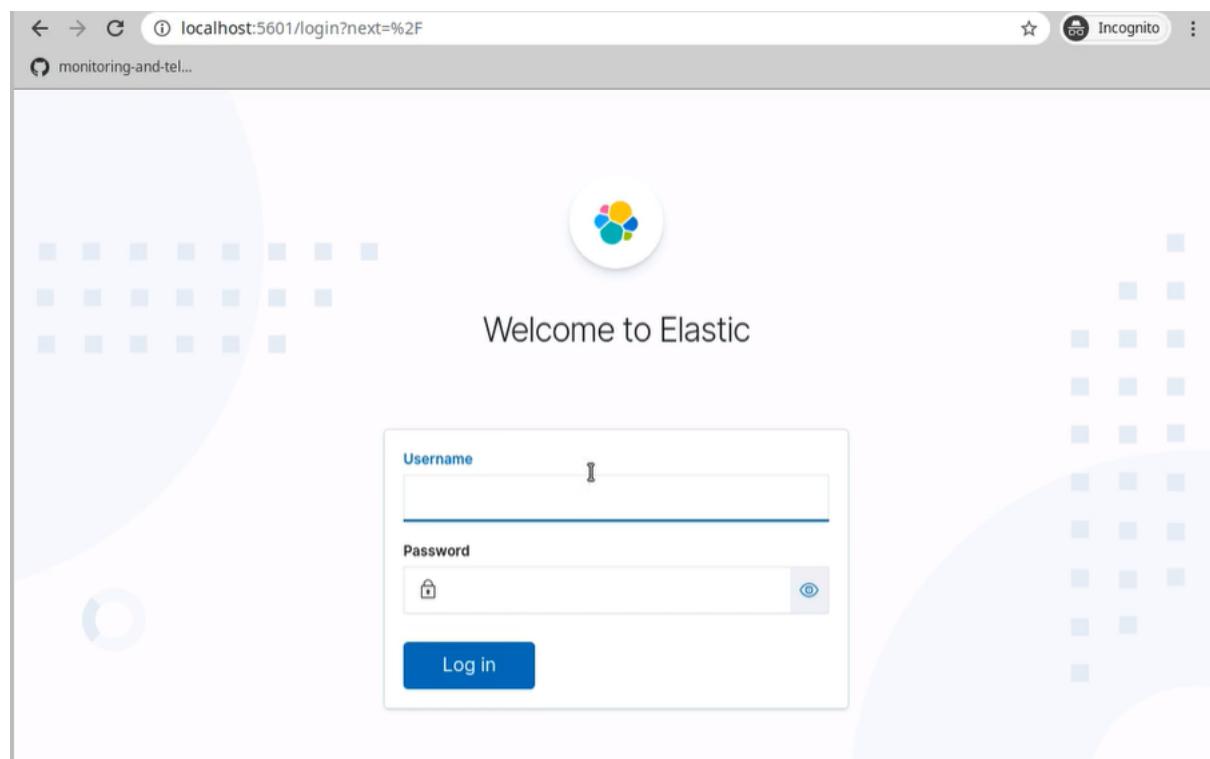
```
rhyme@ip-172-31-234-206:~$ sudo docker rm -f $(sudo docker ps -a -q)
6c00bb8bd546
6a9b4173c736
```

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_logstash_1 ...
Creating dockerelk_kibana_1 ...
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 future 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:



localhost:5601/app/home#/ monitoring-and-tel... Incognito

Welcome to Elastic

Start by adding your data

Add data to your cluster from any source, then analyze and visualize it in real time. Use our solutions to add search anywhere, observe your ecosystem, and protect against security threats.

[Add data](#) [Explore on my own](#)

To learn about how usage data helps us manage and improve our products and services, see our [Privacy Statement](#). To stop collection, [disable usage data here](#).

localhost:5601/app/home#/ monitoring-and-tel... Incognito

Elastic

Search Elastic

Home

Enterprise Search
Search everything →

Build a powerful search experience.
Connect your users to relevant data.
Unify your team content.

Observability
Centralize & monitor →

Monitor infrastructure metrics.
Trace application requests.
Measure SLAs and react to issues.

Kibana
Visualize & analyze →

Analyze data in dashboards.
Search and find insights.
Design pixel-perfect presentations.
Plot geographic data.
Model, predict, and detect.
Reveal patterns and relationships.

Security
localhost:5601/app/observability/ try →

Prevent threats autonomously.
Detect and respond.
Investigate incidents.

We gonna to try importing the example data:



localhost:5601/app/home#/

monitoring-and-tel...

Elastic

Search Elastic

Home

SIEM & Endpoint Security → Investigate incidents.

Ingest your data

Add data: Ingest data from popular apps and services.

Add Elastic Agent: Add and manage your fleet of Elastic Agents and integrations.

Upload a file: Import your own CSV, NDJSON, or log file.

Try our sample data

Manage your data

Manage permissions: Monitor the stack

Monitor the stack: Track the real-time

Back up and restore: Save snapshots to a

Manage index lifecycles

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

localhost:5601/app/home#/tutorial_directory/sampleData

monitoring-and-tel...

Elastic

Search Elastic

Home / Add data

All Logs Metrics Security Sample data

Sample eCommerce orders
Sample data, visualizations, and dashboards for tracking eCommerce orders.
[Add data](#)

Sample flight data
Sample data, visualizations, and dashboards for monitoring flight routes.
[Add data](#)

Sample web logs
Sample data, visualizations, and dashboards for monitoring web logs.
[Add data](#)

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



localhost:5601/app/home#/tutorial_directory/sampleData

monitoring-and-tel...

Elastic

Home / Add data

All Logs Metrics Security Sample data

Sample eCommerce orders

Sample data, visualizations, and dashboards for tracking eCommerce orders.

Add data

Sample flight data

Sample data, visualizations, and dashboards for monitoring flight routes.

Add data

Sample web logs

Sample data, visualizations, and dashboards for monitoring web logs.

✓ Sample web logs installed

This screenshot shows the Kibana home page with three main sample data sections: 'Sample eCommerce orders', 'Sample flight data', and 'Sample web logs'. Each section contains a brief description, a visualization, and an 'Add data' button. A message at the bottom right indicates that 'Sample web logs' have been successfully installed.

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

localhost:5601/app/home#/tutorial_directory/sampleData

monitoring-and-tel...

Elastic

Home / Add data

Home

Recently viewed

Kibana

- Overview
- Discover**
- Dashboard
- Canvas
- Maps
- Machine Learning
- Graph
- Visualize

Enterprise Search

localhost:5601/app/discover

Sample data

Sample flight data

Sample data, visualizations, and dashboards for monitoring flight routes.

Add data

Sample web logs

Sample data, visualizations, and dashboards for monitoring web logs.

Remove View data

This screenshot shows the Kibana Discover interface. The left sidebar has 'Discover' selected. The main area displays the 'Sample flight data' and 'Sample web logs' sections, each with its own description and visualization. Buttons for 'Add data', 'Remove', and 'View data' are visible at the bottom of each section.

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



localhost:5601/app/discover#/?_g=(filters:!(),refreshInterval:(pause:1t,value:0),time:(from:now-15m,to:n...)

monitoring-and-tel...

Elastic

Discover

Search KQL Last 15 minutes Show dates Refresh

+ Add filter

kibana_sample... ▾

Search field r Filter by type 0

Selected fields _source

Available fields hour_of_day

No results match your search criteria

Expand your time range

One or more of the indices you're looking at contains a date field. Your query may not match anything in the current time range, or there may not be any data at all in the currently selected time range. You can try changing the time range to one which contains data.

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

localhost:5601/app/discover#/?_g=(filters:!(),refreshInterval:(pause:1t,value:0),time:(from:now-15m,to:n...)

monitoring-and-tel...

Quick select

Last 15 minutes Show dates Refresh

Commonly used

Today	Last 24 hours
This week	Last 7 days
Last 15 minutes	Last 30 days
Last 30 minutes	Last 90 days
Last 1 hour	Last 1 year

Recently used date ranges

Last 15 minutes This week

Refresh every

0 seconds Start

No results match your search criteria

One or more of the indices you're looking at contains a date field. Your query may not match anything in the current time range, or there may not be any data at all in the currently selected time range. You can try changing the time range to one which contains data.

So, we got, the logs for this week:



localhost:5601/app/discover#/?_g=(filters:!(),refreshInterval:(pause:lt,value:0),time:(from:now%2Fw,to:n...)

monitoring-and-telemetry

Elastic

Discover

Search KQL This week Show dates Refresh

kibana_sample... + Add filter

Selected fields: _source

Available fields: _id, _index, _score, _type, @timestamp, agent, bytes

Count Dec 13, 2020 @ 00:00:00.000 - Dec 19, 2020 @ 23:59:59.999 Auto

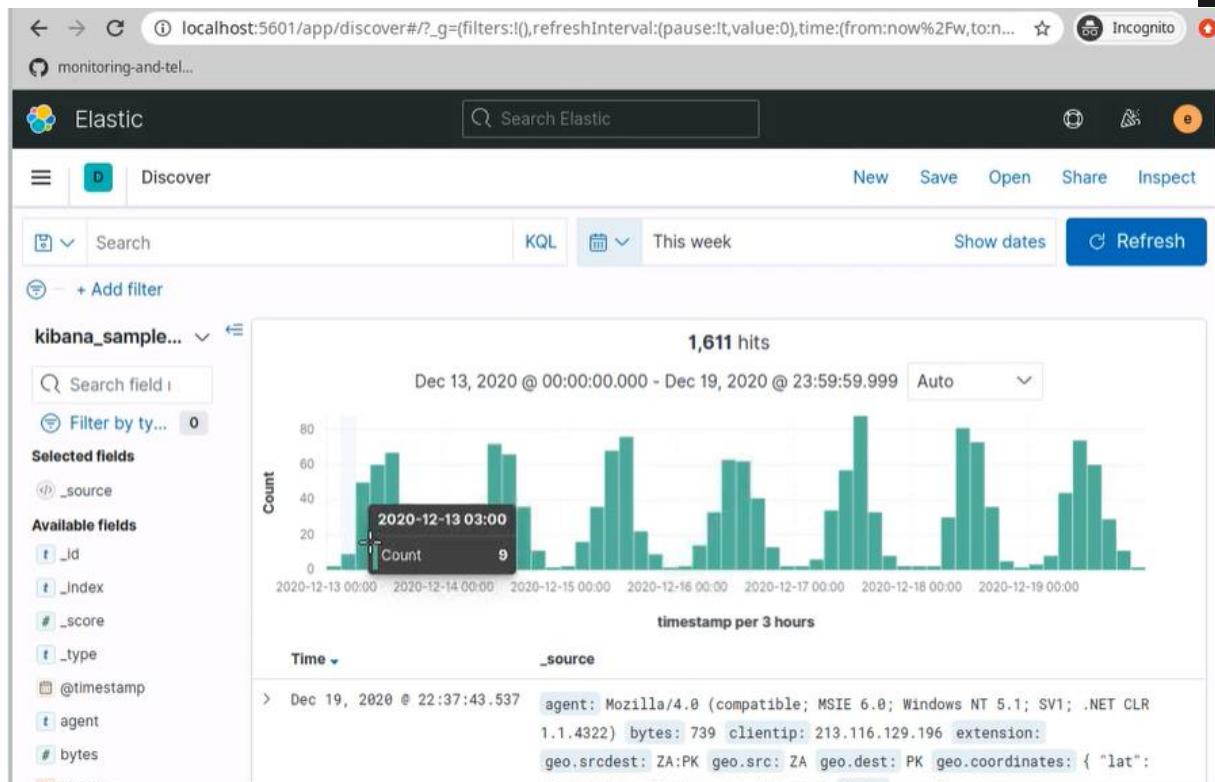
1,611 hits

Dec 13, 2020 @ 00:00:00.000 - Dec 19, 2020 @ 23:59:59.999 Auto

Count timestamp per 3 hours

Time _source

> Dec 19, 2020 @ 22:37:43.537 agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322) bytes: 739 clientip: 213.116.129.196 extension: geo.srcdest: ZA:PK geo.src: ZA geo.dest: PK geo.coordinates: { "lat":



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

localhost:5601/app/discover#/?_g=(filters:!(),refreshInterval:(pause:lt,value:0),time:(from:now%2Fw,to:n...)

monitoring-and-telemetry

Elastic

Discover

Home Recently viewed

Kibana Overview Discover Dashboard Canvas Maps Machine Learning Graph Visualize

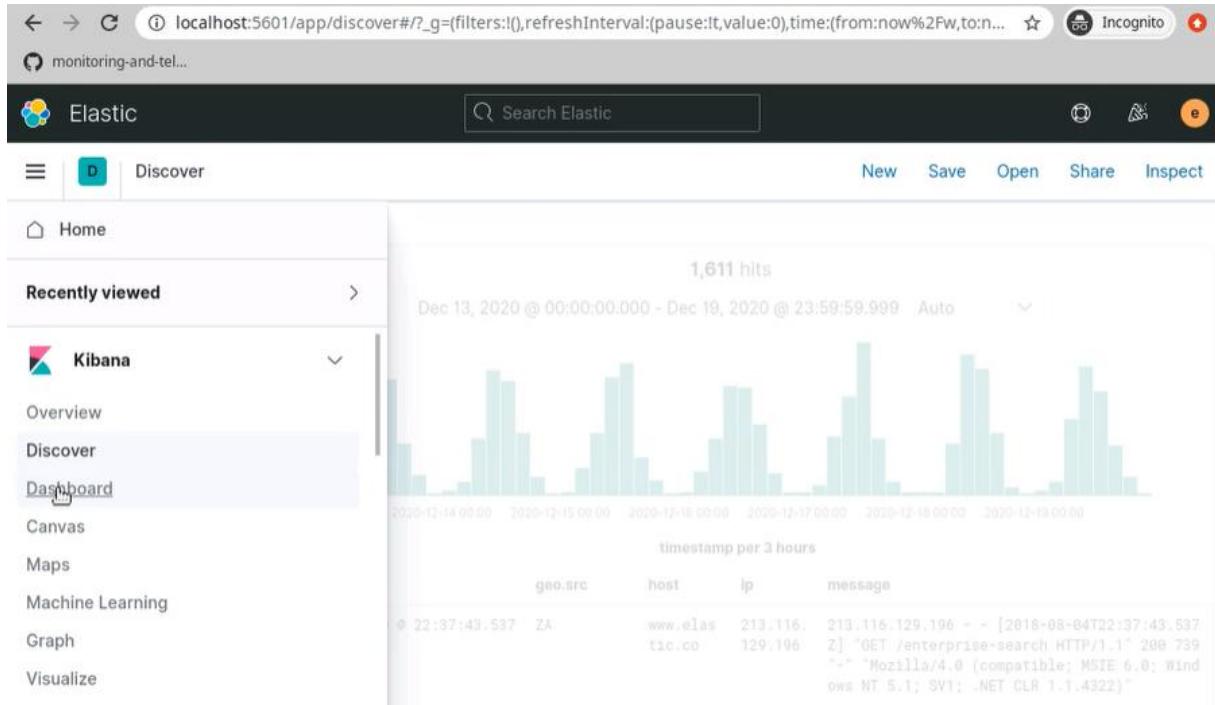
1,611 hits

Dec 13, 2020 @ 00:00:00.000 - Dec 19, 2020 @ 23:59:59.999 Auto

Count timestamp per 3 hours

geo.src host ip message

> 22:37:43.537 ZA www.elastic.co 213.116.129.196 - [2018-08-04T22:37:43.537Z] "GET /enterprise-search HTTP/1.1" 200 739 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)"



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:(),refreshInterval:5s,searchSourceId:1,sort:[])&_t=1544811600000

monitoring-and-telemetry

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

Unique Visitors

803

Source Country

Select... OS Select... Bytes

[Logs] Visitors by OS

OS	Percentage
osx	20.4%
ios	20.4%
Windows	59.2%

[Logs] Response Codes Over Time + Annotations

[Logs] Unique Visitors vs. Average Bytes

localhost:5601/app/dashboards#/view/edf84fe0-e1a0-11e7-b6d5-4dc382ef7f5b?_g=(filters:(),refreshInterval:5s,searchSourceId:1,sort:[])&_t=1544811600000

monitoring-and-telemetry

Elastic

Dashboard / [Logs] Web Traffic

Full screen Share Clone Edit

Logstash Metrics

Metric	Value	Unit	Count	Size
zip	1.2MB	0B	210	0
deb	1MB	0B	164	0
rpm	425.5KB	0B	66	0

[Logs] Heatmap

[Logs] Source and Destination Sankey Chart

Total Requests and Bytes

