ELK Stack SIEM Implementation

A hands-on SOC lab built on Ubuntu — Elasticsearch · Kibana · Logstash · Nginx

Project Overview

I built this project to get hands-on experience with the ELK Stack by setting up a home lab that collects, processes, and visualises real system data. The stack runs on a local Ubuntu VM and includes Elasticsearch for storage and indexing, Kibana for visualisation, Logstash as the data processing pipeline, Auditbeat to collect system events, and Nginx as a reverse proxy with authentication in front of Kibana. The goal was to go through the full setup from scratch and end up with actual data flowing through the pipeline — not just installed services, but something working end to end.

Virtual Machine Setup

I ran everything inside a dedicated VM to keep the setup isolated from my host machine. Elasticsearch is particularly memory-hungry, so I wanted to give it a clean environment with enough resources to actually run without constantly swapping. I used VMware Fusion with the following specs:

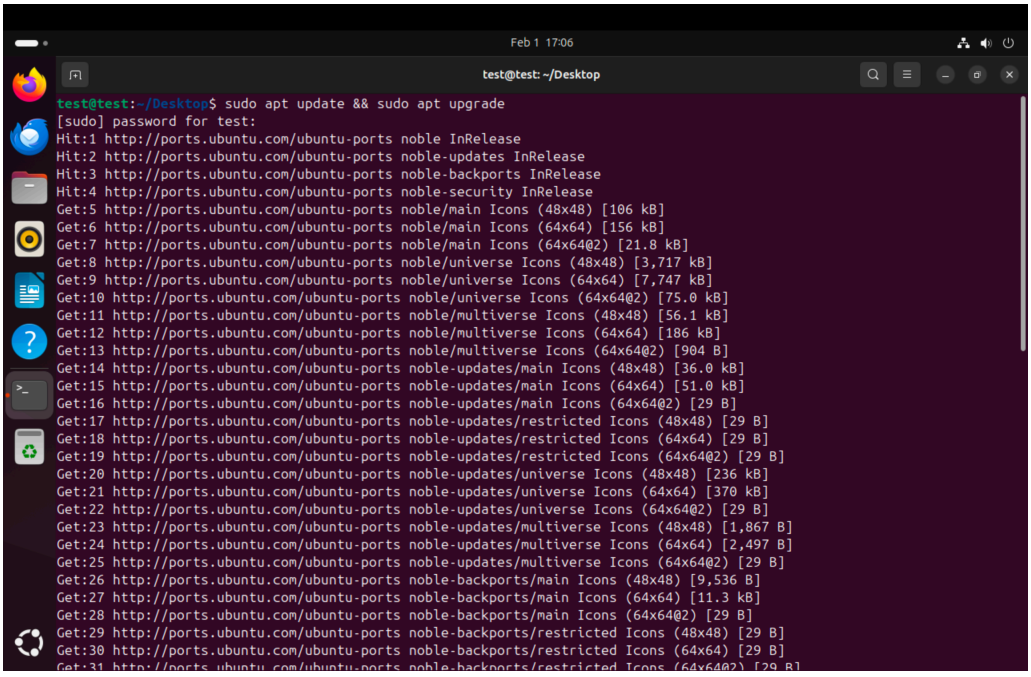
- **Platform:** VMware Fusion
- **OS:** Ubuntu 22.04 LTS
- **Storage:** 50 GB — Elasticsearch index files grow quickly, so you need the space
- **RAM:** 4 GB — the minimum to run Elasticsearch and Kibana together without issues

I went with Ubuntu 22.04 LTS because it's stable, well-supported, and Elastic maintains an official apt repository for it, which makes installing and updating the stack much cleaner than doing it manually.

Step 1 — System Update

Before installing anything, I updated the system to make sure all existing packages were current. This matters because outdated packages can cause dependency conflicts when you start adding third-party repositories like Elastic's, and it also keeps the base system clean before layering anything on top.

```
sudo apt update && sudo apt upgrade
```

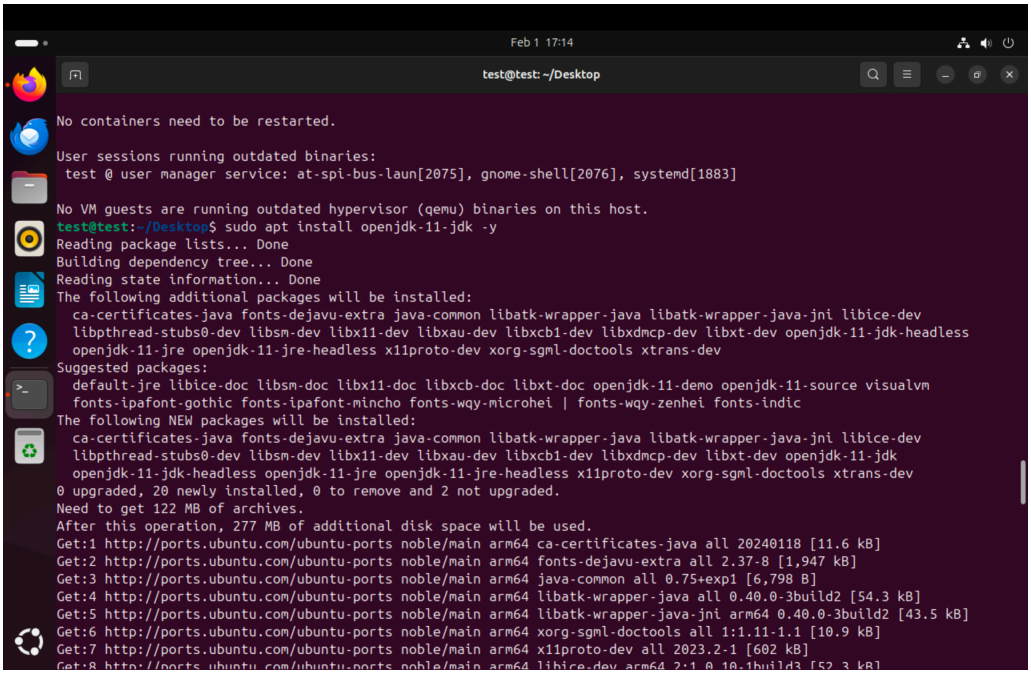


System update running — fetching latest packages

Step 2 — Install Java

Elasticsearch runs on the JVM, so Java needs to be installed first. The version matters here — Elasticsearch 8.x works with OpenJDK 11, so that's what I installed. Using a different version can cause the service to fail on startup without a very obvious error message, so it's worth pinning to the right one from the start.

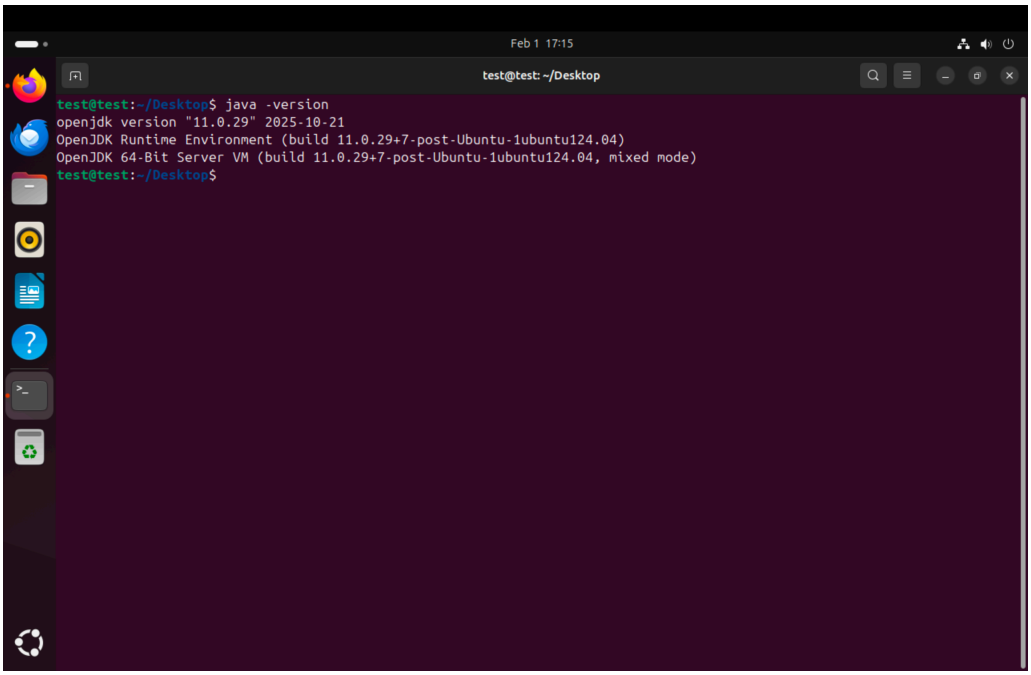
```
sudo apt install openjdk-11-jdk -y
```



OpenJDK 11 installing

After the install, I verified it was picked up correctly. If you have multiple Java versions on the system this is especially important, since the wrong one being active will cause Elasticsearch to fail silently.

```
java -version
```



OpenJDK 11.0.29 — correct version confirmed

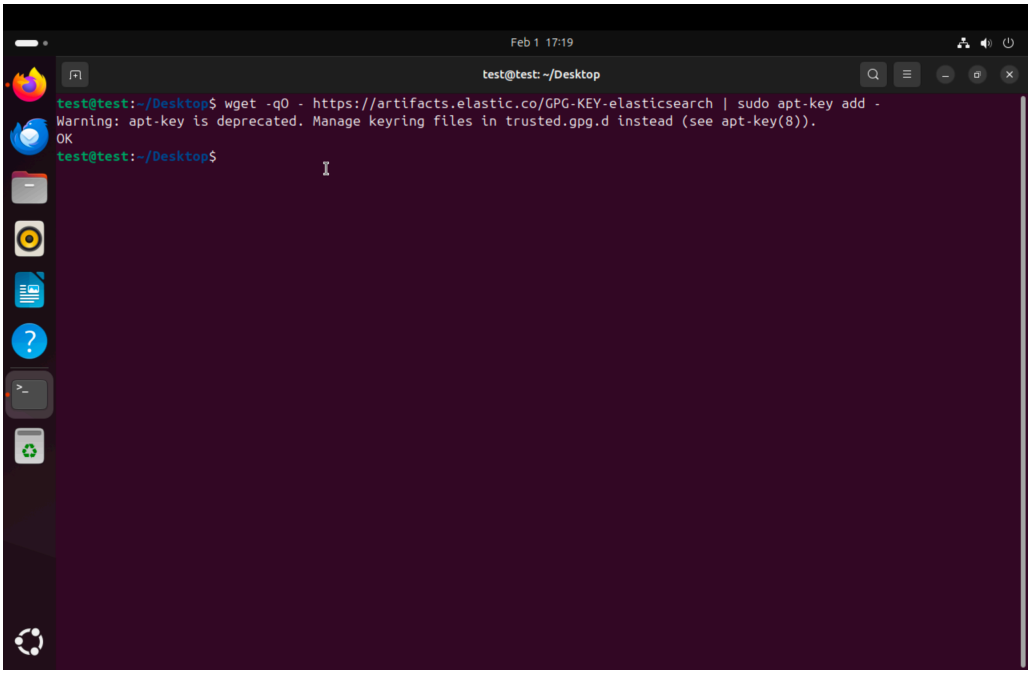
Step 3 — Install Elasticsearch

Elasticsearch is the core of the whole setup, it's where all the data lives. Everything else either pushes data into it or reads data out of it. Rather than downloading a package manually, I added Elastic's official apt repository so I could install it properly and get updates through the normal package manager going forward.

Add the Elastic repository

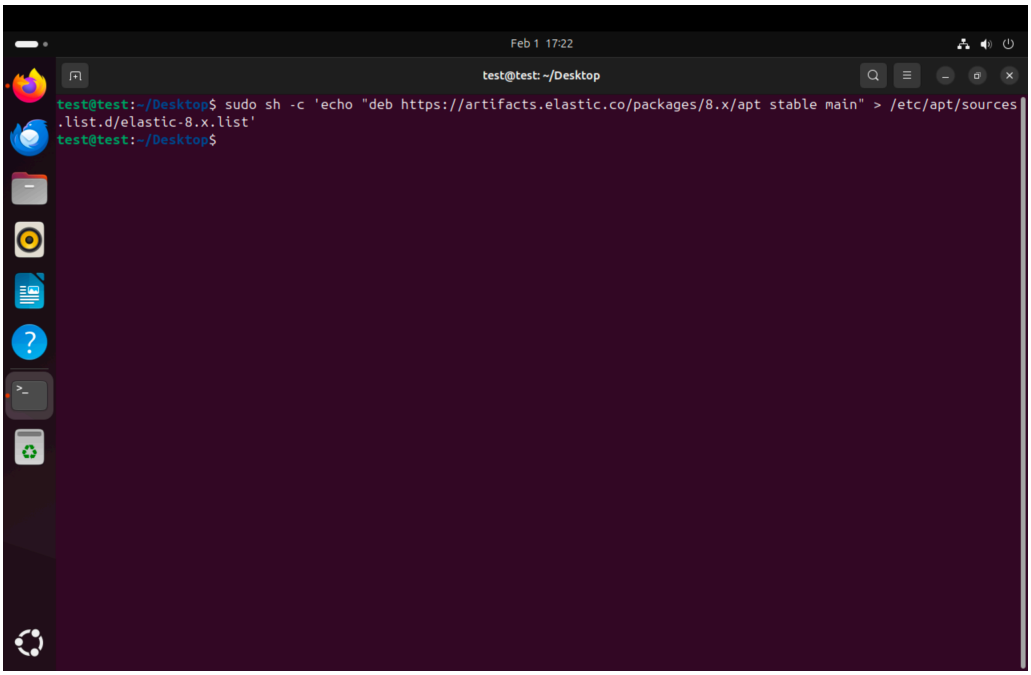
First I added the GPG key so apt can verify the packages actually come from Elastic, then registered the repository itself. Without the GPG key, apt will refuse to install from that source.

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
```



GPG key added — OK

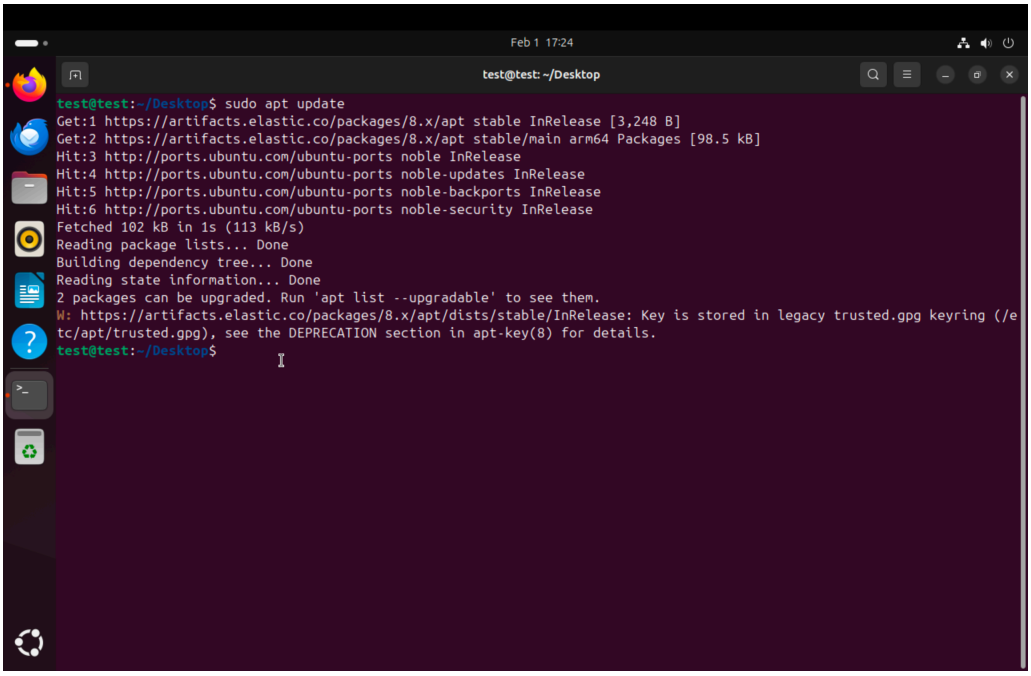
```
sudo sh -c 'echo "deb https://artifacts.elastic.co/packages/8.x/apt stable main" > /etc/apt/
sources.list.d/elastic-8.x.list'
```



Elastic 8.x repository added to apt sources

Then refresh the package list so apt picks up what's available from the new repository:

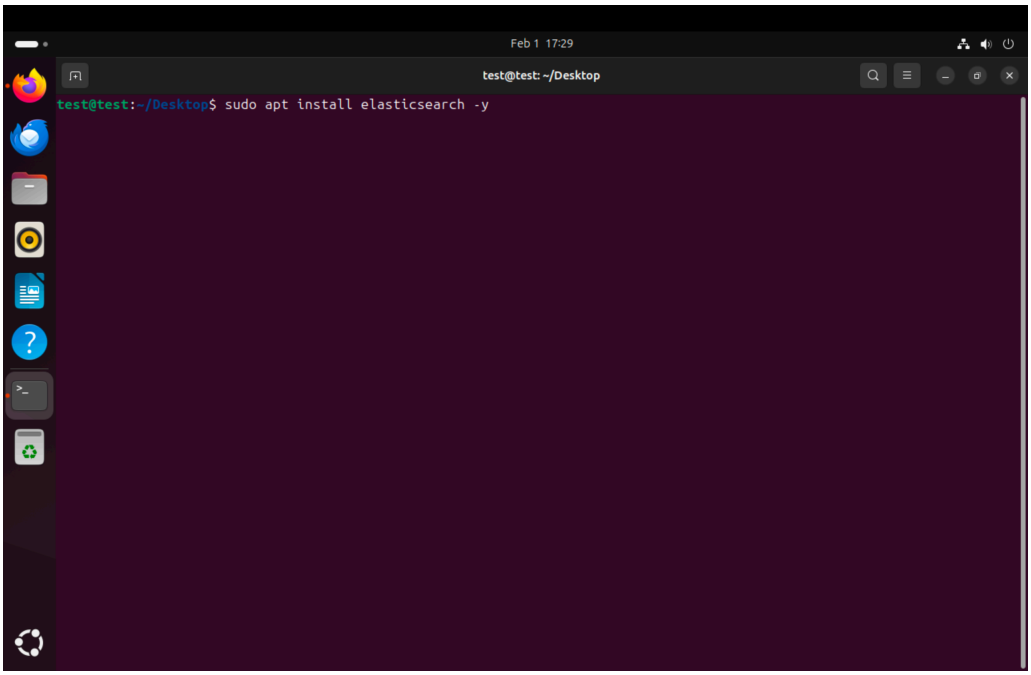
```
sudo apt update
```



apt now sees packages from artifacts.elastic.co

Install

```
sudo apt install elasticsearch -y
```

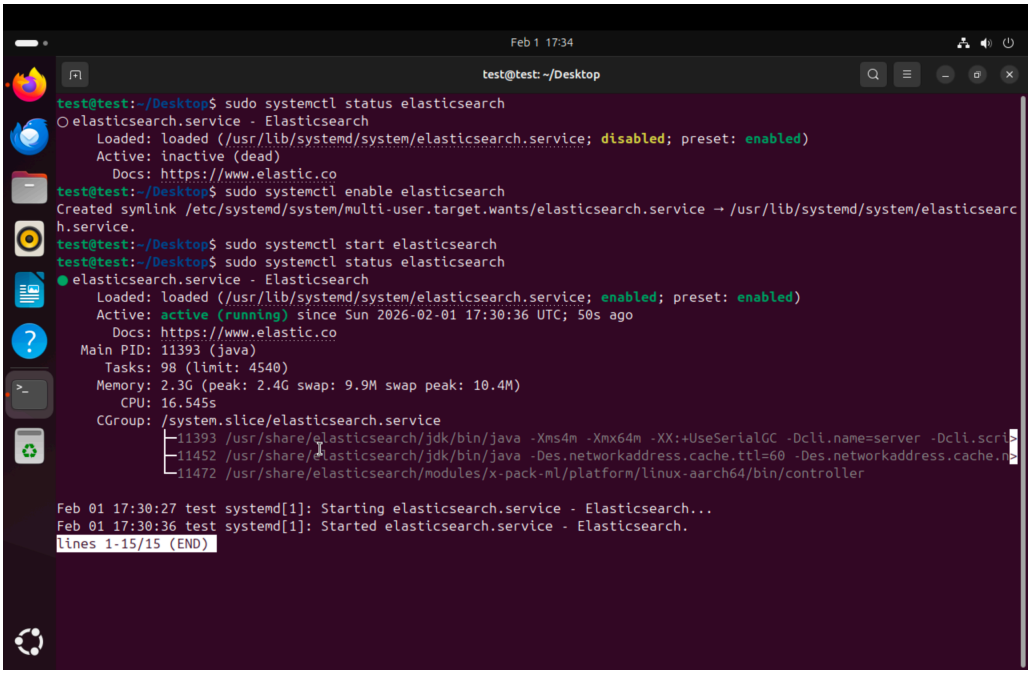


Elasticsearch downloading and installing

Enable and start

Elasticsearch doesn't start automatically after install, which is intentional, you're expected to configure it first. I enabled it so it starts on reboot, then started it manually, and checked the status to make sure it came up cleanly.

```
sudo systemctl enable elasticsearch
sudo systemctl start elasticsearch
sudo systemctl status elasticsearch
```

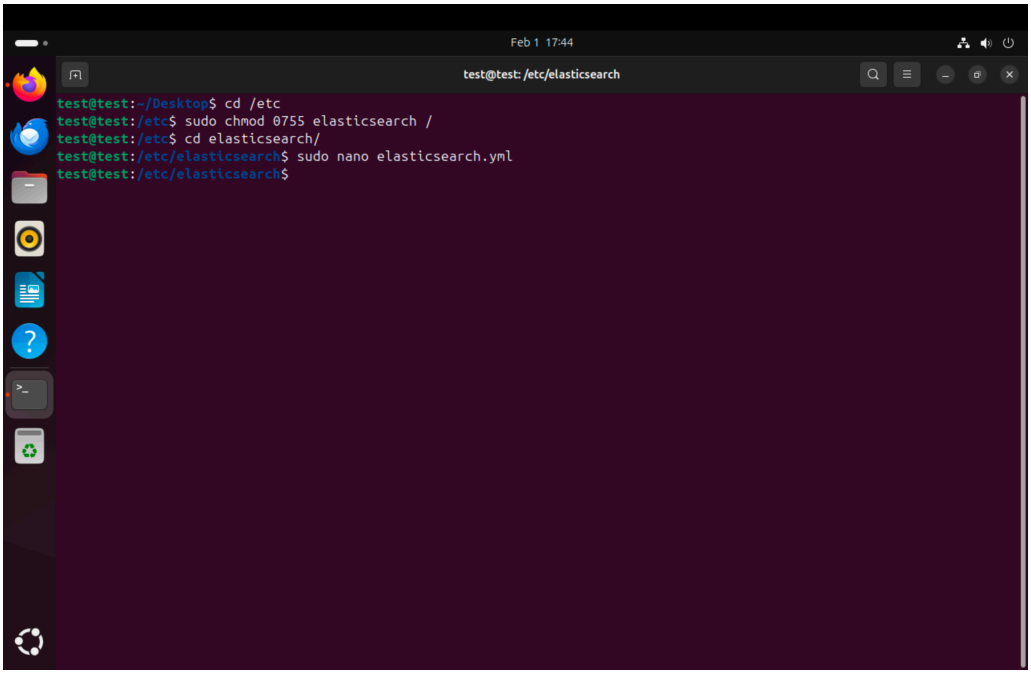


Elasticsearch active and running

Configure — elasticsearch.yml

The default config is bare-bones and won't get the cluster to form properly. I edited the config file to set the cluster name, node name, and the network settings Elasticsearch needs to bootstrap itself as a single-node cluster.

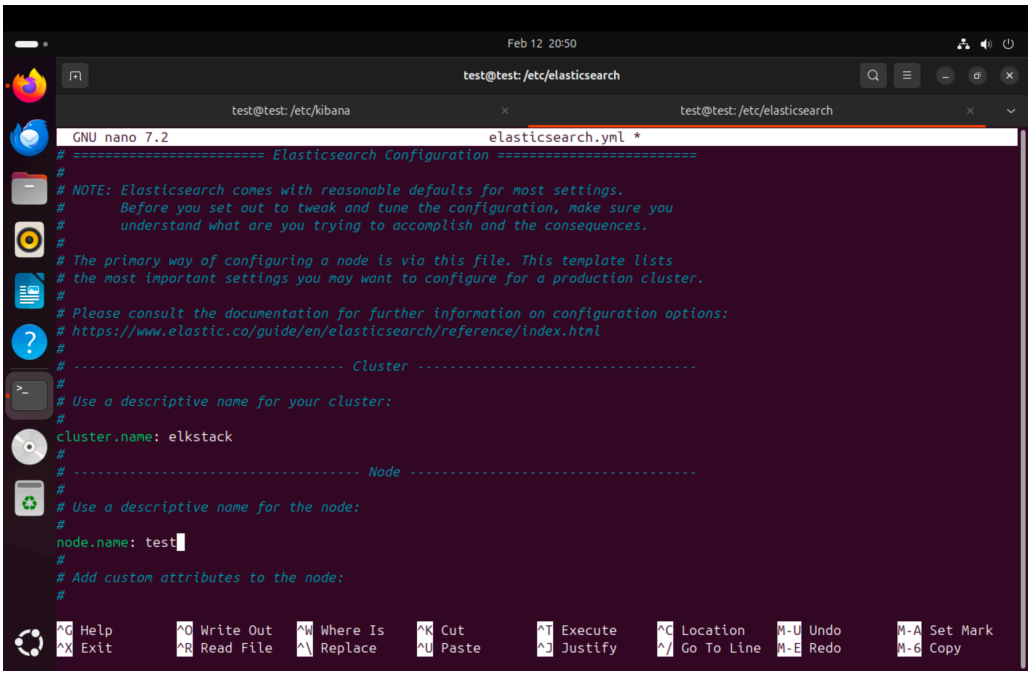
```
cd /etc && sudo chmod 0755 elasticsearch/
cd elasticsearch/
sudo nano elasticsearch.yml
```



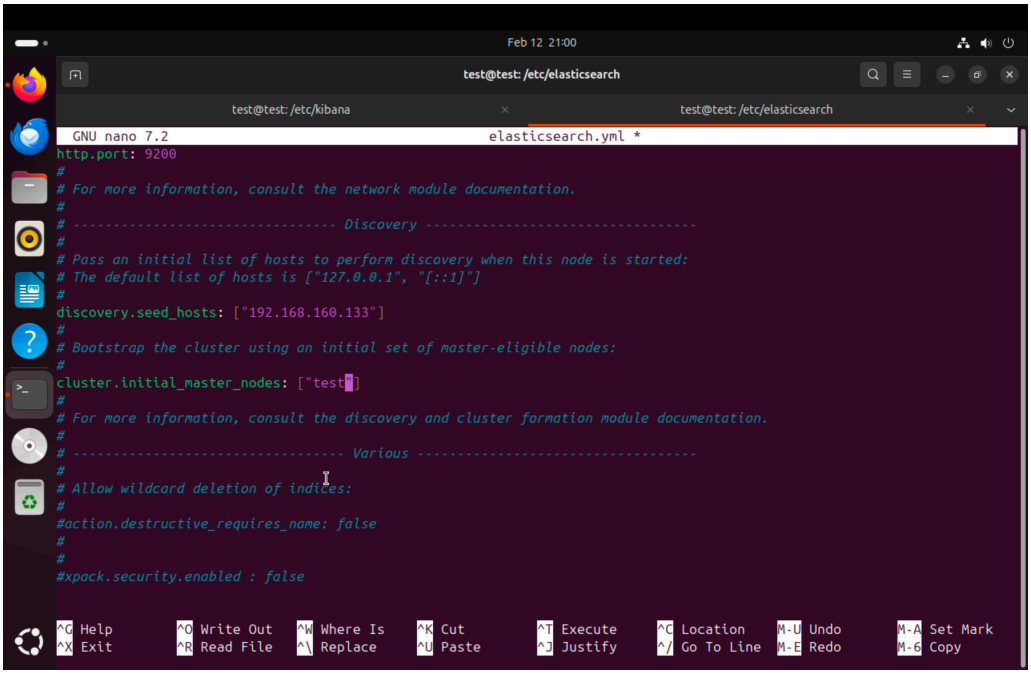
Opening elasticsearch.yml

Here's what I changed and why each setting is needed:

- **cluster.name: elkstack** : a name for the cluster; nodes only join if the cluster name matches
- **node.name: test** : a name for this specific node, which shows up in logs and the API response
- **discovery.seed_hosts: ["192.168.160.133"]** : where to look for other nodes on startup; in a single-node setup this just points to itself
- **cluster.initial_master_nodes: ["test"]** : required for the first bootstrap; without it, Elasticsearch won't elect a master and the cluster stays stuck
- **http.port: 9200** : the port for REST API requests; this is the default and what Kibana will connect to



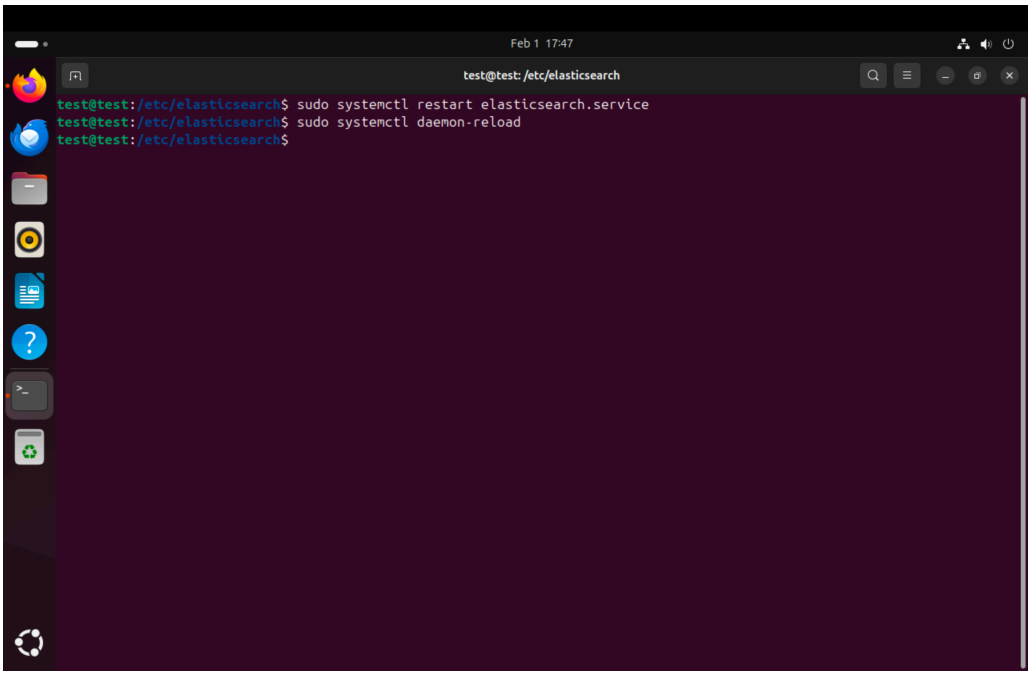
elasticsearch.yml — cluster.name and node.name set



elasticsearch.yml — discovery and master node settings

After saving, I restarted the service to apply the changes:

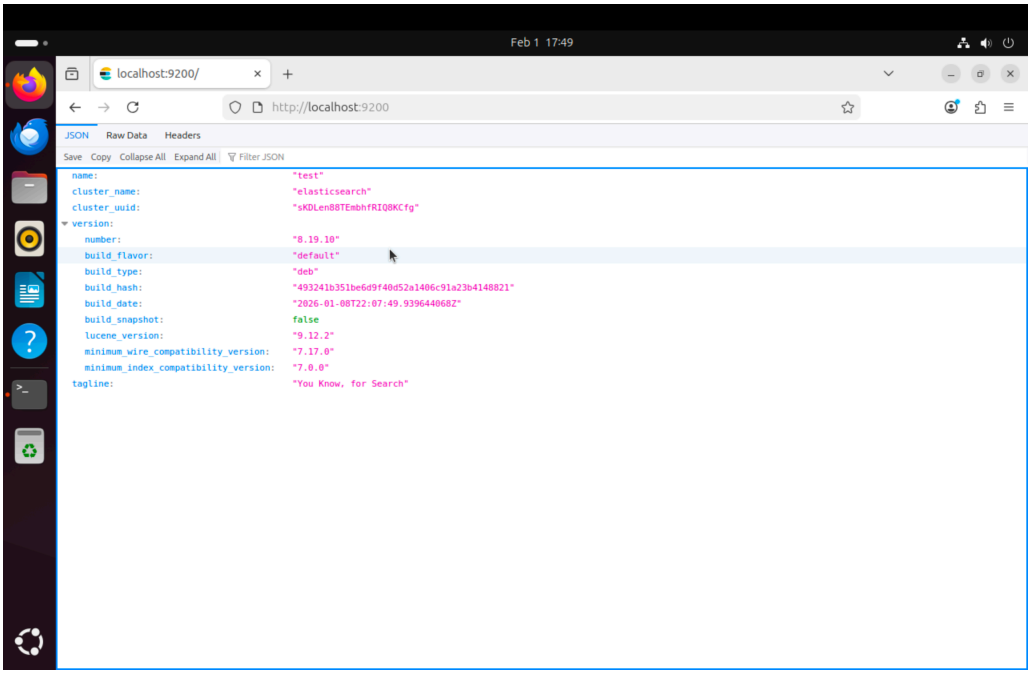
```
sudo systemctl restart elasticsearch.service
sudo systemctl daemon-reload
```



Elasticsearch restarted after config change

Verify

I opened localhost:9200 in the browser to confirm Elasticsearch was up and responding. If you get a JSON response with the cluster name and version, the node formed correctly. If the page doesn't load, something in the config is off — check the service logs with journalctl -u elasticsearch.



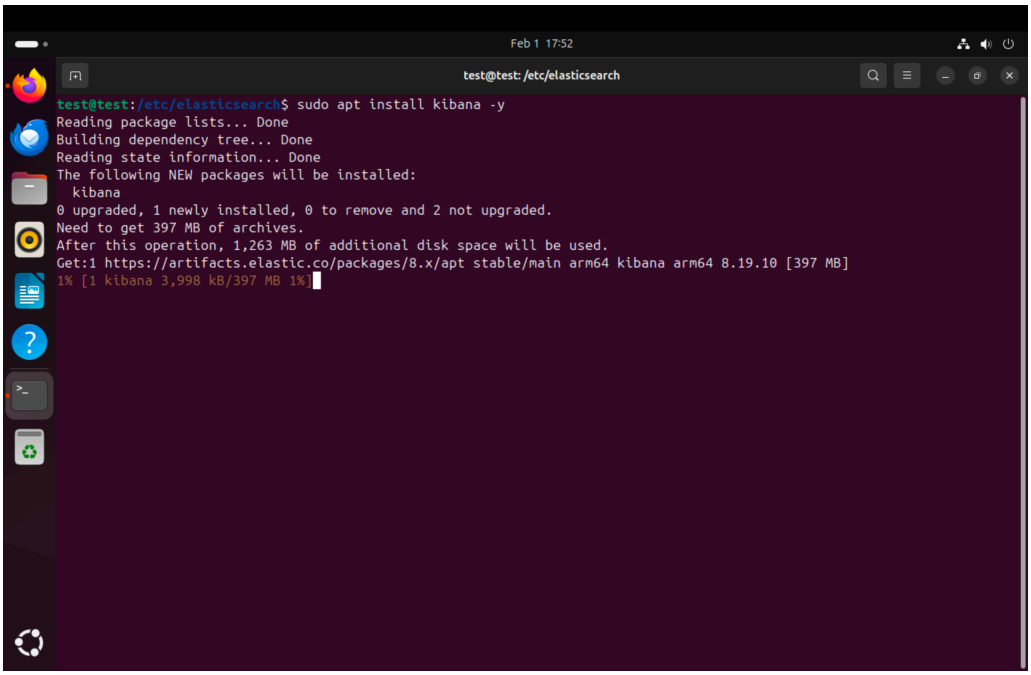
Elasticsearch 8.19.10 responding at localhost:9200

Step 4 — Install Kibana

Kibana is the web interface for the stack, it connects to Elasticsearch and gives you a browser-based UI to search and visualise all the data that's been indexed. Since I already added the Elastic repository, installing Kibana is one command. The important thing is to make sure it's the same version as Elasticsearch — mismatched versions cause connection errors that aren't always obvious.

Install

```
sudo apt install kibana -y
```

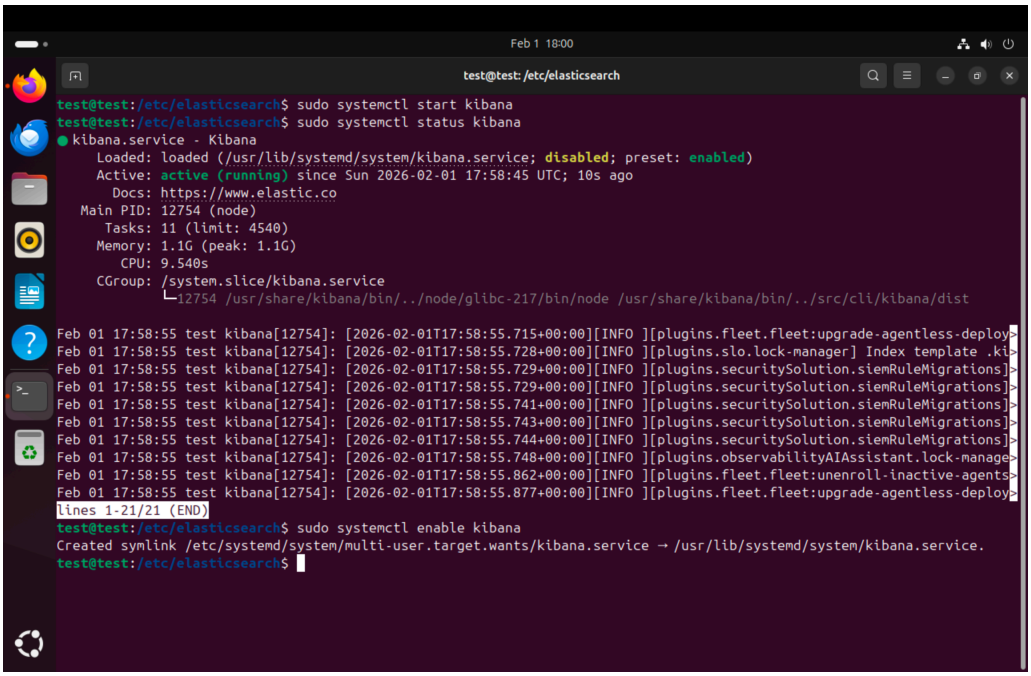


Kibana 8.19.10 downloading — matching Elasticsearch version

Start and enable

Kibana is a Node.js app and takes a bit longer to start than most services because it needs to connect to Elasticsearch and set up its own internal indices. Give it a minute or two before trying to open it in the browser.

```
sudo systemctl start kibana
sudo systemctl status kibana
sudo systemctl enable kibana
```

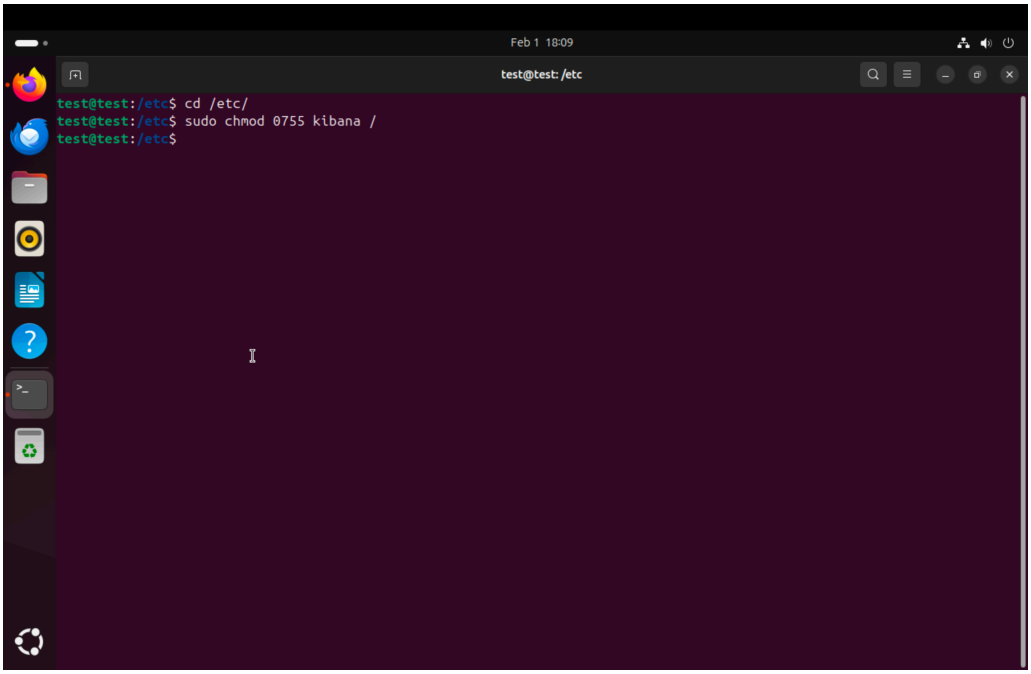


Kibana active and running

Configure — kibana.yml

By default Kibana only listens on localhost, which means the browser on my host machine can't reach it. I edited the config to bind it to all interfaces and set the public URL, so links and redirects generated by Kibana point to the right address.

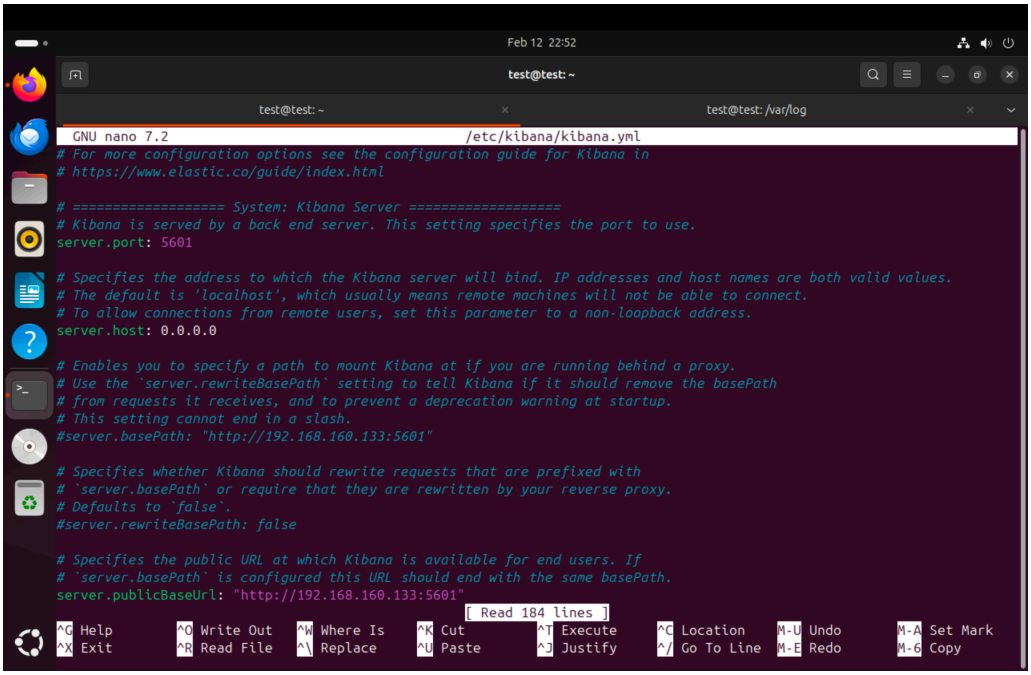
```
cd /etc/ && sudo chmod 0755 kibana/
```



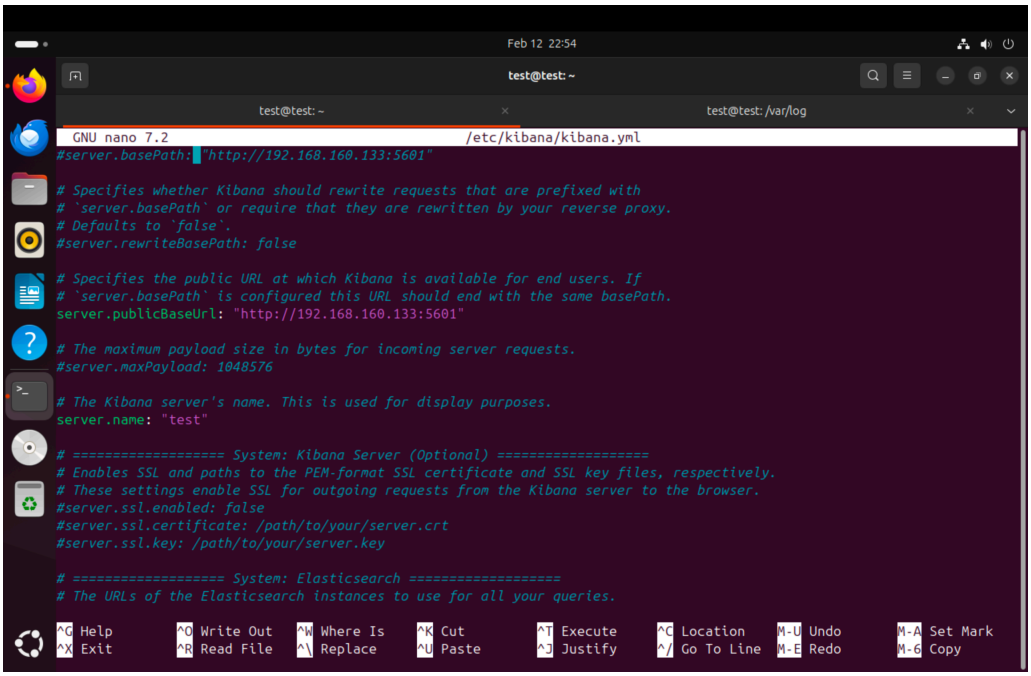
Setting permissions on /etc/kibana

The settings I changed:

- **server.port: 5601** : default Kibana port
- **server.host: 0.0.0.0** : without this, Kibana only listens on localhost and you can't reach it from another machine
- **server.name: test** : display name for this instance
- **server.publicBaseUrl: http://192.168.160.133:5601** : the external URL; needed so Kibana generates correct links once Nginx is in front



kibana.yml — port and host configured

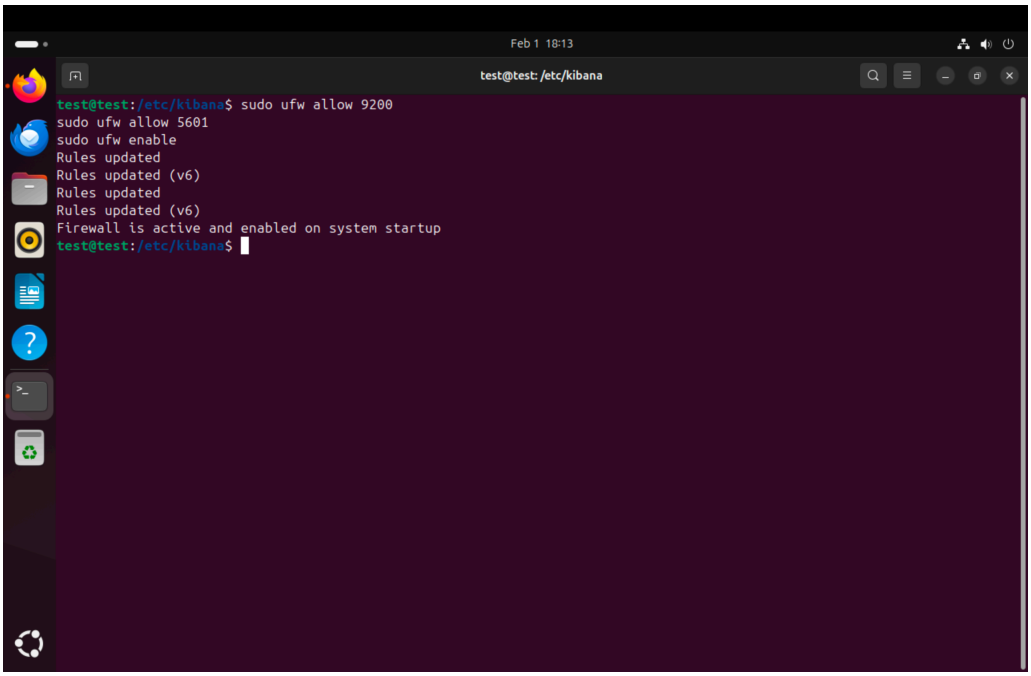


kibana.yml — server name and publicBaseUrl set

Open firewall ports

Ubuntu's firewall (UFW) blocks inbound traffic by default. I opened ports 9200 and 5601 so I could reach Elasticsearch and Kibana from the host machine. If you skip this step and wonder why nothing connects — this is usually why.

```
sudo ufw allow 9200
sudo ufw allow 5601
sudo ufw enable
```

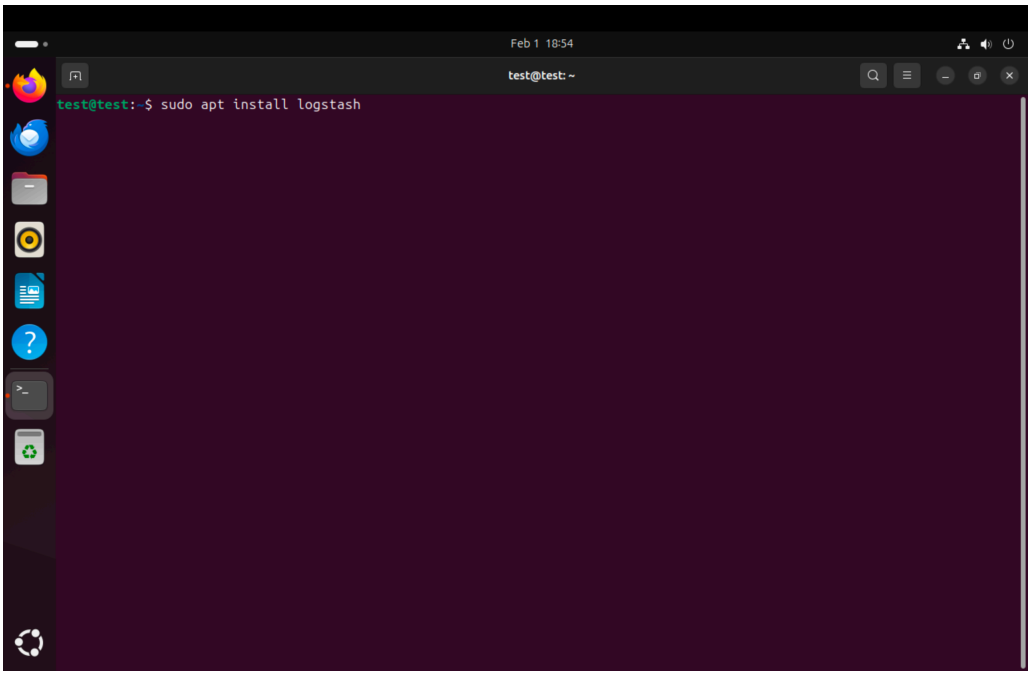


UFW rules added and firewall enabled

Step 5 — Install and Configure Logstash

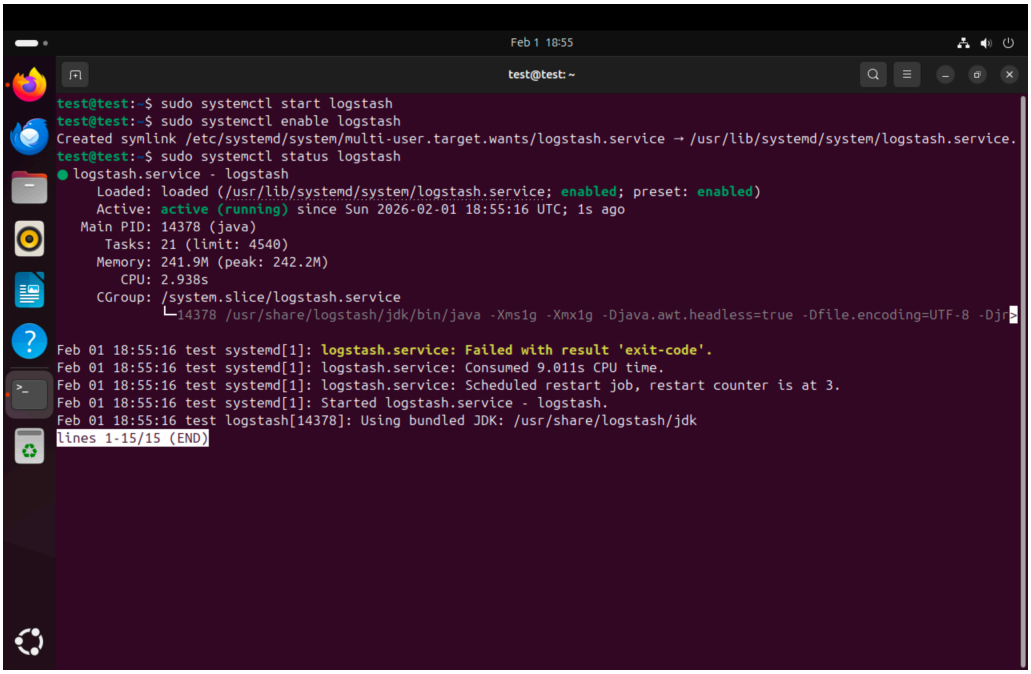
Logstash is the data pipeline, it receives logs from various sources, processes and transforms them, and forwards them to Elasticsearch. I installed it and then created two config files: one that tells Logstash where to listen for incoming data, and one that tells it where to send the output.

```
sudo apt install logstash
```



Logstash installing

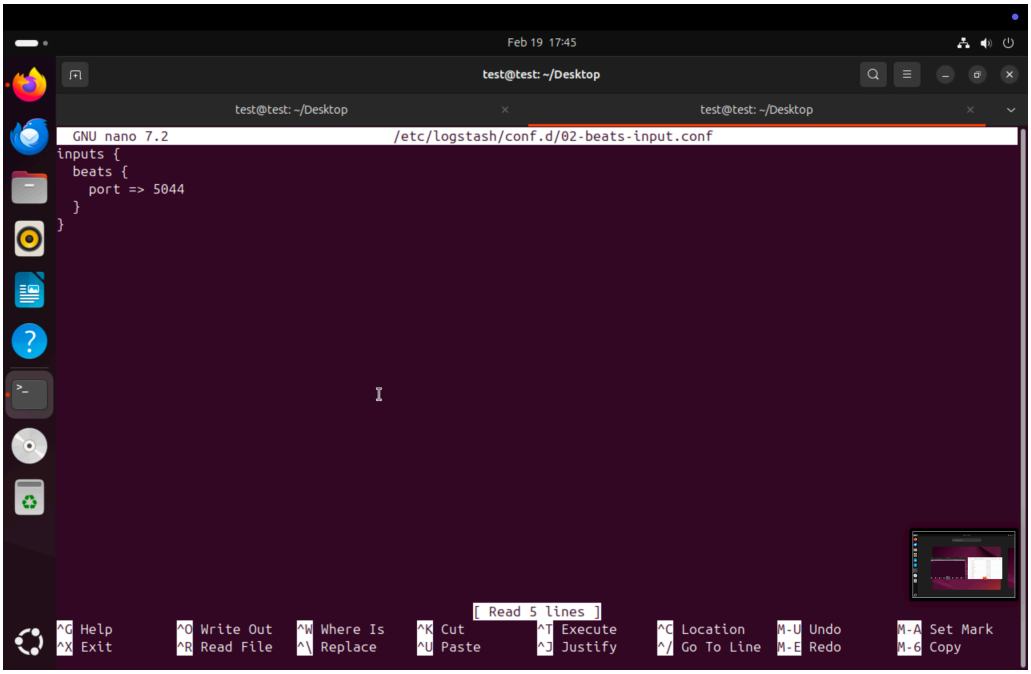
```
sudo systemctl start logstash
sudo systemctl enable logstash
sudo systemctl status logstash
```



Logstash active and running

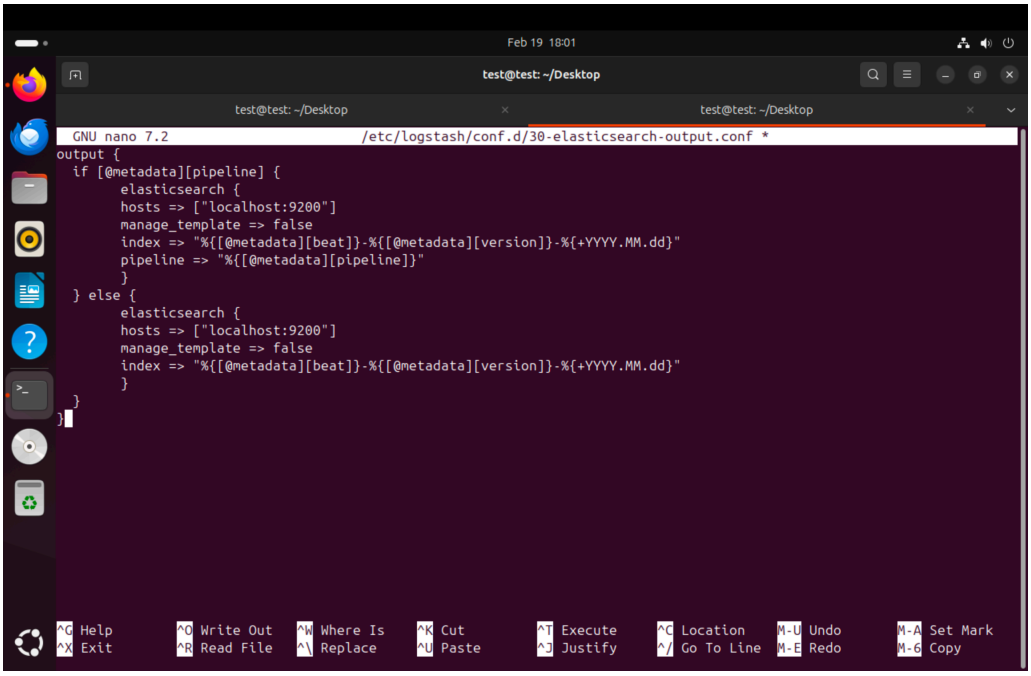
Configure the pipeline

Logstash reads pipeline config from .conf files placed in /etc/logstash/conf.d/. I created two files to define the input and output.
The input file tells Logstash to listen on port 5044 for connections from Beat agents (Auditbeat, Filebeat, etc.). Port 5044 is the default for the Beats protocol, so any Beat you configure will try to send there by default.



02-beats-input.conf — listening on port 5044 for Beat agents

The output file routes the processed events to Elasticsearch and dynamically names the index based on the Beat that sent the data, so Auditbeat events go into auditbeat-* indices, Filebeat events into filebeat-* and so on. This keeps the data organised and makes it easy to target the right index pattern in Kibana later.

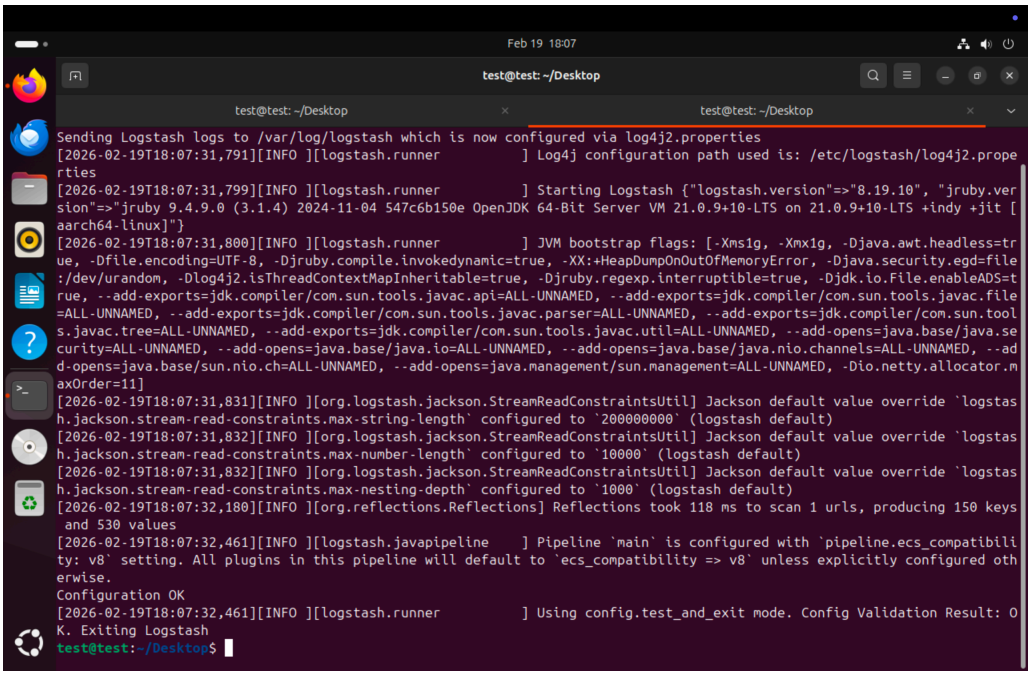


30-elasticsearch-output.conf — forwarding to Elasticsearch with dynamic index names

Test the pipeline config

I always test new Logstash configs before restarting the service. A syntax error in a pipeline file will crash Logstash on restart, so this validation step saves a lot of headache.

```
sudo -u logstash /usr/share/logstash/bin/logstash --path.settings /etc/logstash -t
```



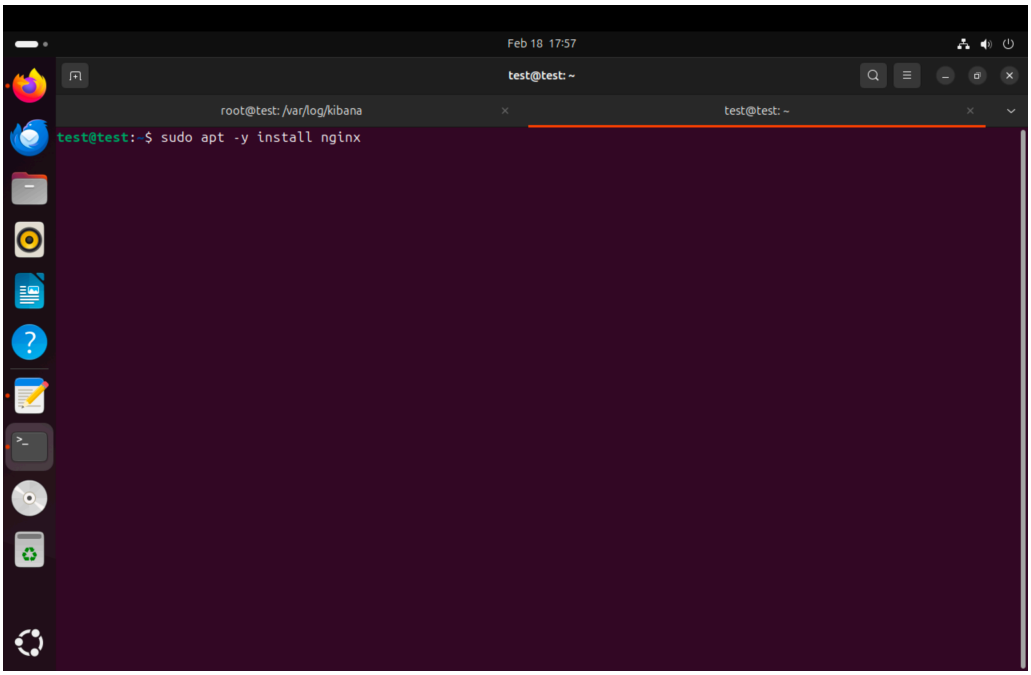
Logstash config test — "Configuration OK"

Step 6 — Set Up Nginx as Reverse Proxy

Kibana was accessible on port 5601 at this point, but I wanted two things: a cleaner URL on port 80, and a login prompt so the dashboard isn't just open to anyone who can reach the VM. I set up Nginx as a reverse proxy that sits in front of Kibana and handles both.

Install Nginx

```
sudo apt -y install nginx
```

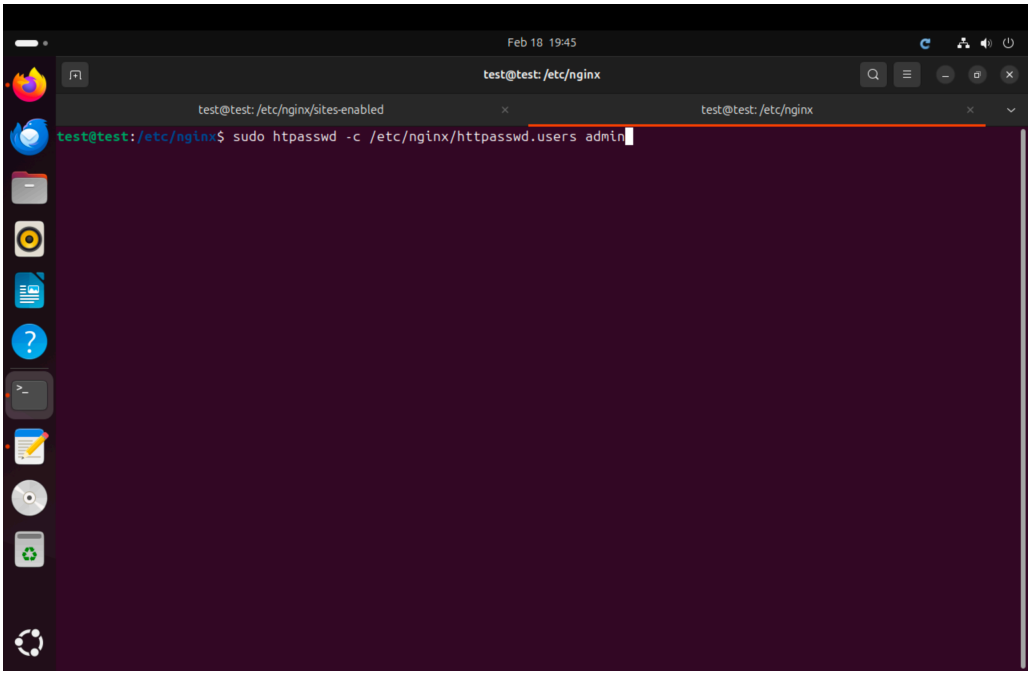


Nginx installing

Create a password file

Nginx's Basic Auth reads from an htpasswd file, a list of usernames with their hashed passwords. The -c flag creates the file. If you run it again with -c later it will wipe the existing one, so only use -c the first time.

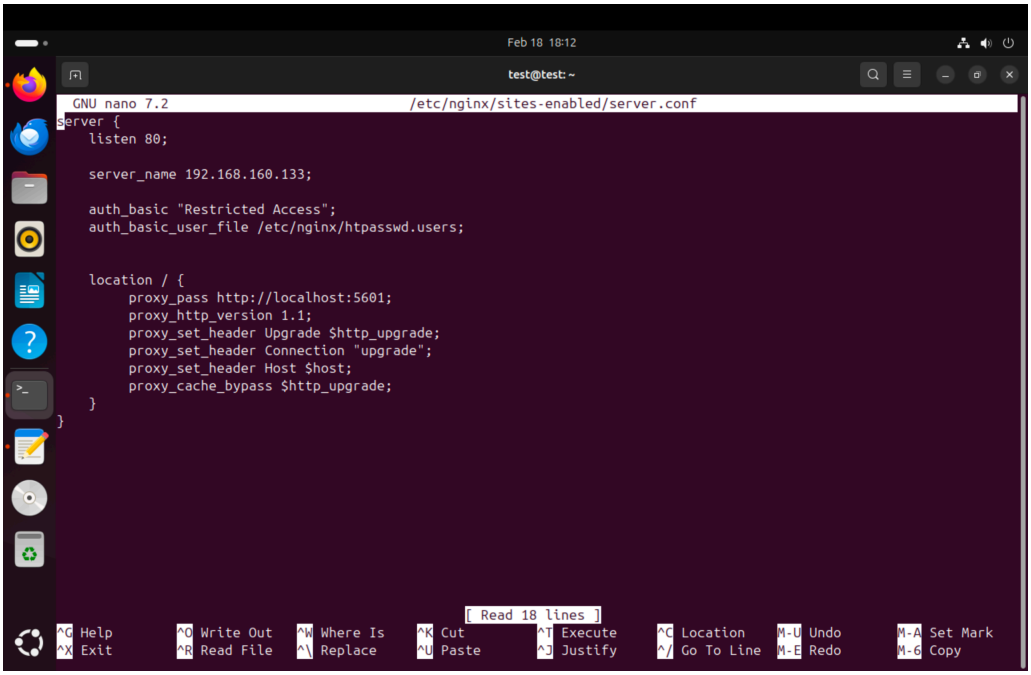
```
sudo htpasswd -c /etc/nginx/httpasswd.users admin
```



htpasswd file created with admin user

Configure the virtual host

I created a server block that listens on port 80, checks credentials against the htpasswd file, and proxies authenticated requests to Kibana on localhost:5601. The proxy headers for Upgrade and Connection are important — Kibana uses WebSockets, and without those headers the connection gets dropped by the proxy.

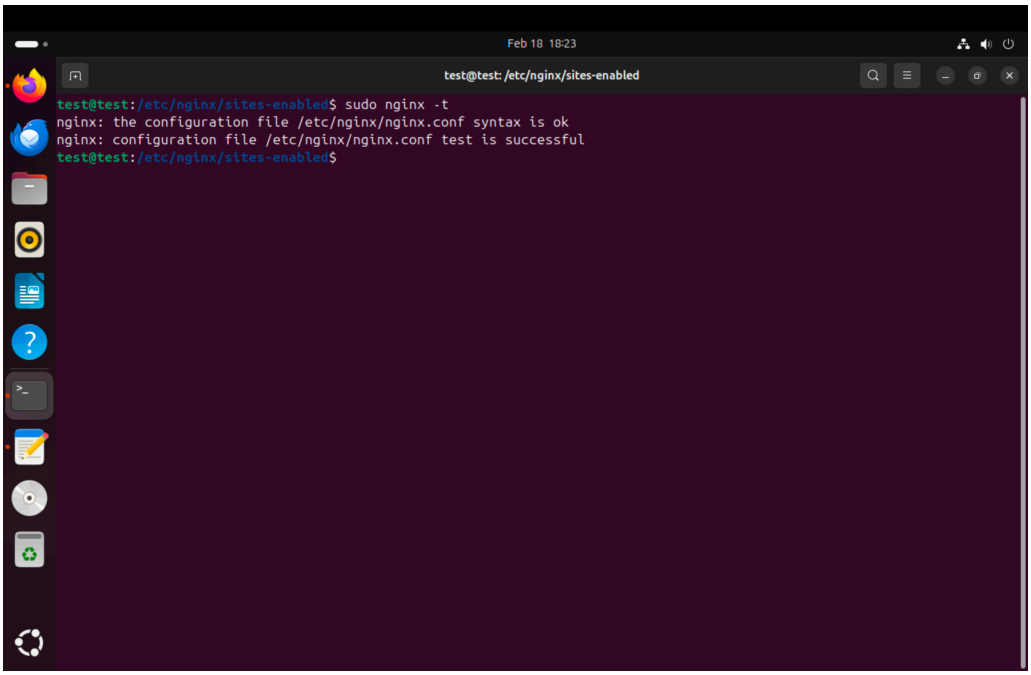


/etc/nginx/sites-enabled/server.conf — the reverse proxy config

Test and enable

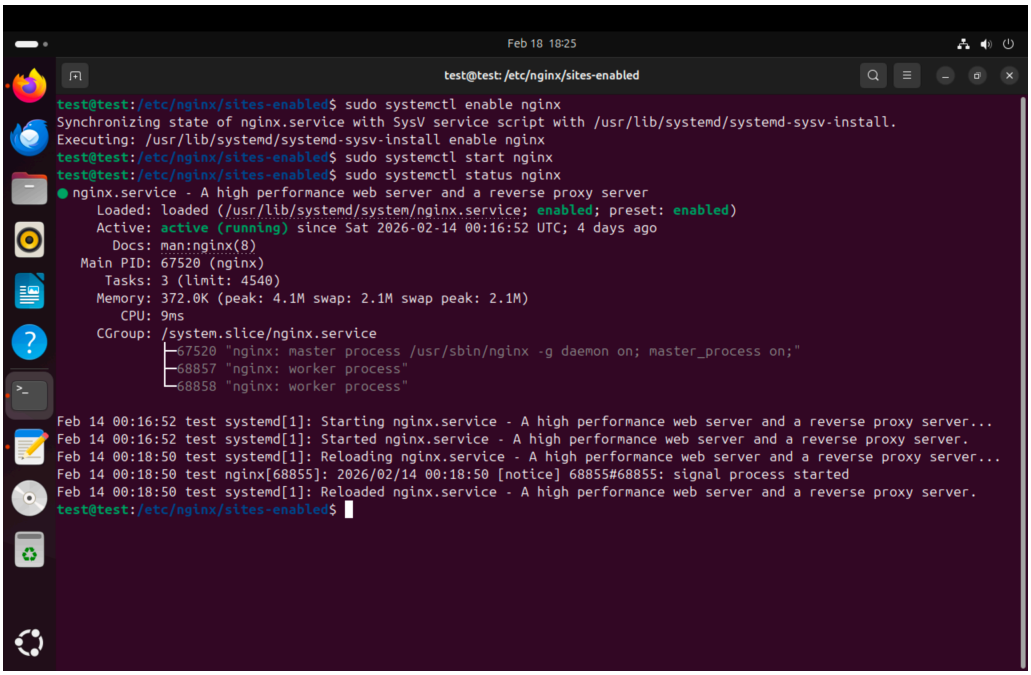
I ran nginx -t to validate the config before restarting. A syntax error here takes the whole Nginx server down, so it's always worth checking first.

```
sudo nginx -t
```



Config test passes — syntax is OK

```
sudo systemctl enable nginx
sudo systemctl start nginx
sudo systemctl status nginx
```



Nginx running — reverse proxy is live on port 80

Step 7 — Install and Configure Auditbeat

With the pipeline in place, I needed something to actually generate data. I chose Auditbeat because it hooks directly into the Linux audit subsystem and captures real system-level events (logins, process starts, network connections, user changes) rather than just reading log files. It sends everything as structured JSON, which means it's immediately queryable in Kibana without any parsing work.

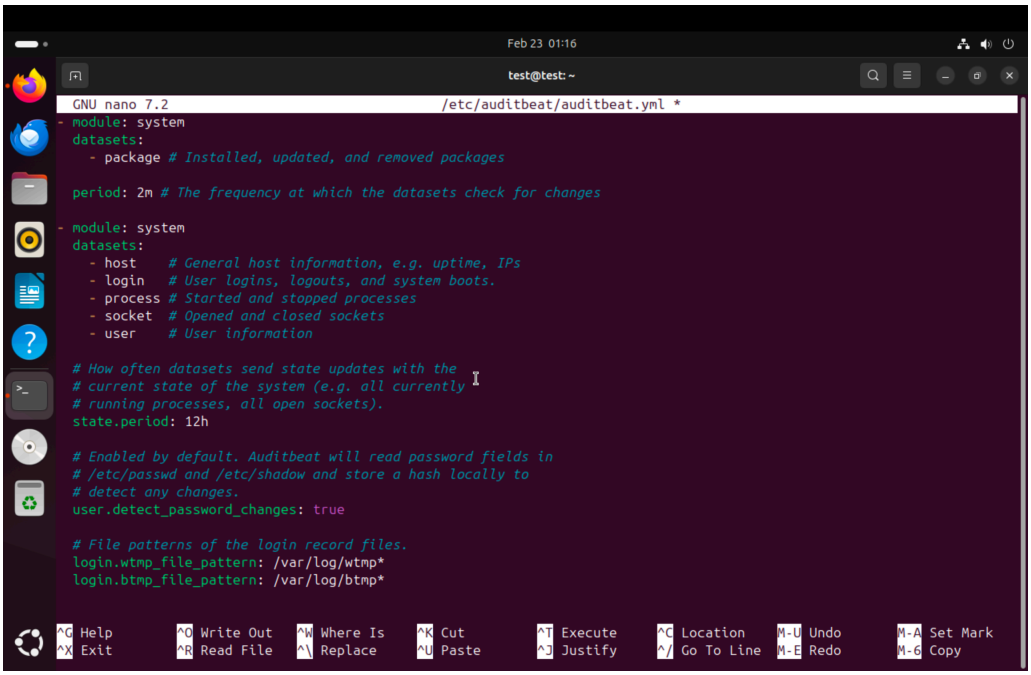
Install

```
sudo apt install auditbeat -y
```

Configure — auditbeat.yml

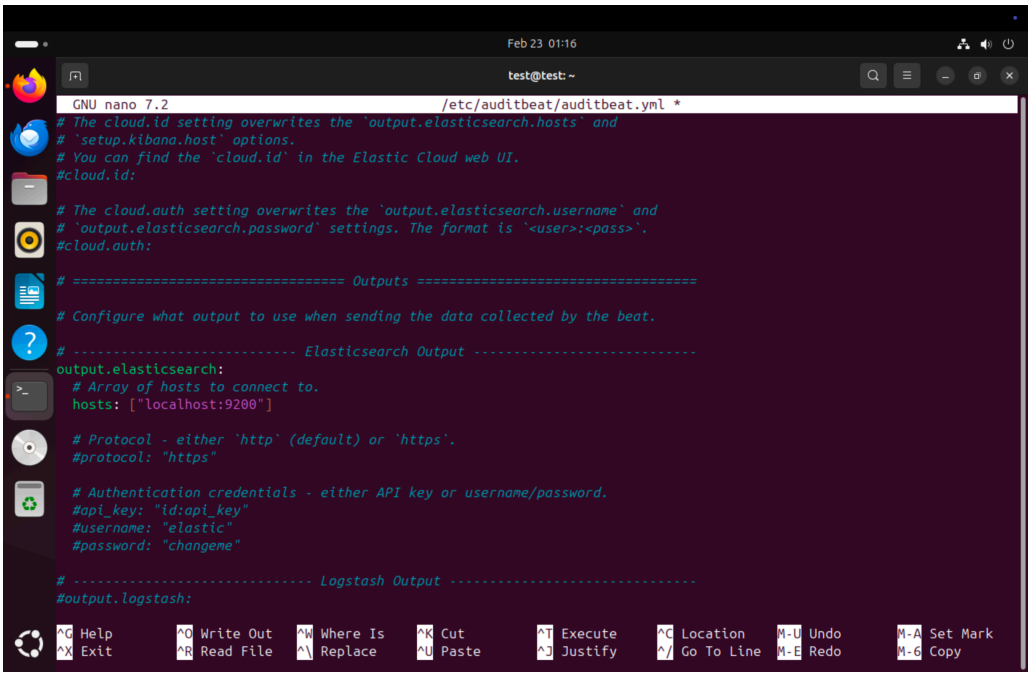
Auditbeat's system module is what I wanted. It monitors several datasets and I enabled all of them to get as much visibility as possible:

- **host** : basic system info: hostname, IPs, OS version, uptime
- **login** : login and logout events, including SSH sessions and failed attempts
- **process** : every process that starts or stops on the system
- **socket** : network connections being opened and closed
- **user** : changes to user accounts on the system
- **package** : packages being installed, updated, or removed



auditbeat.yml — system module with all datasets enabled

For the output I pointed Auditbeat directly at Elasticsearch on port 9200. I could have routed it through Logstash on port 5044 instead, but since Auditbeat already sends clean structured JSON there's nothing to transform, so going direct is simpler.



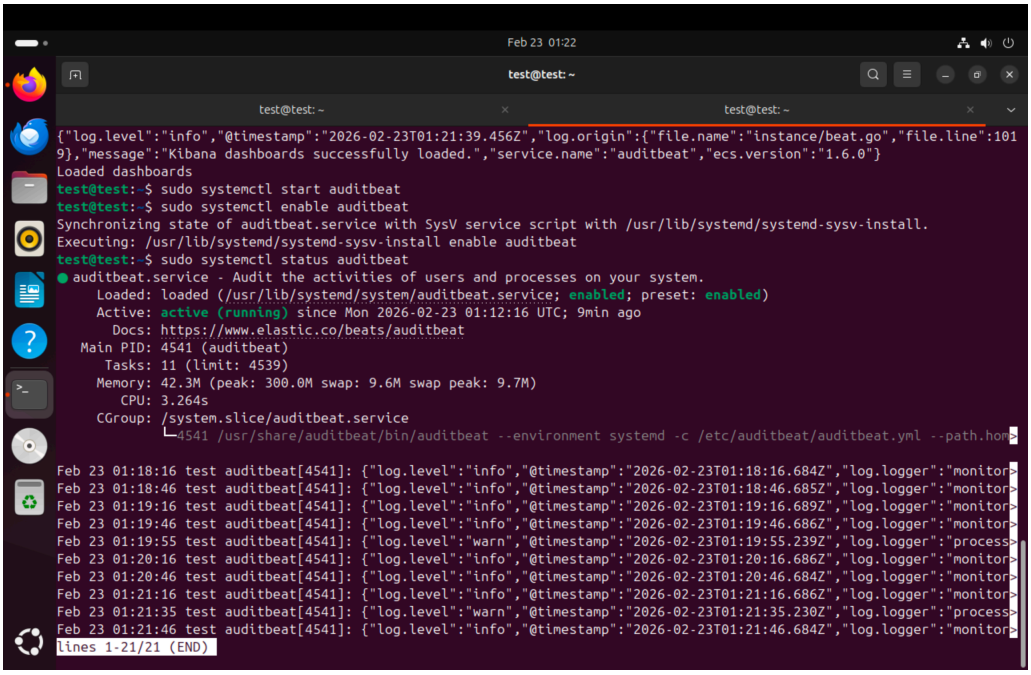
auditbeat.yml — output.elasticsearch set to localhost:9200

Load dashboards, start, and enable

Auditbeat can push its own pre-built dashboards into Kibana automatically. I ran the setup command first so the dashboards were ready as soon as data started flowing. Then I started the service and enabled it so it runs on reboot.

```
sudo auditbeat setup -e
sudo systemctl start auditbeat
sudo systemctl enable auditbeat
```

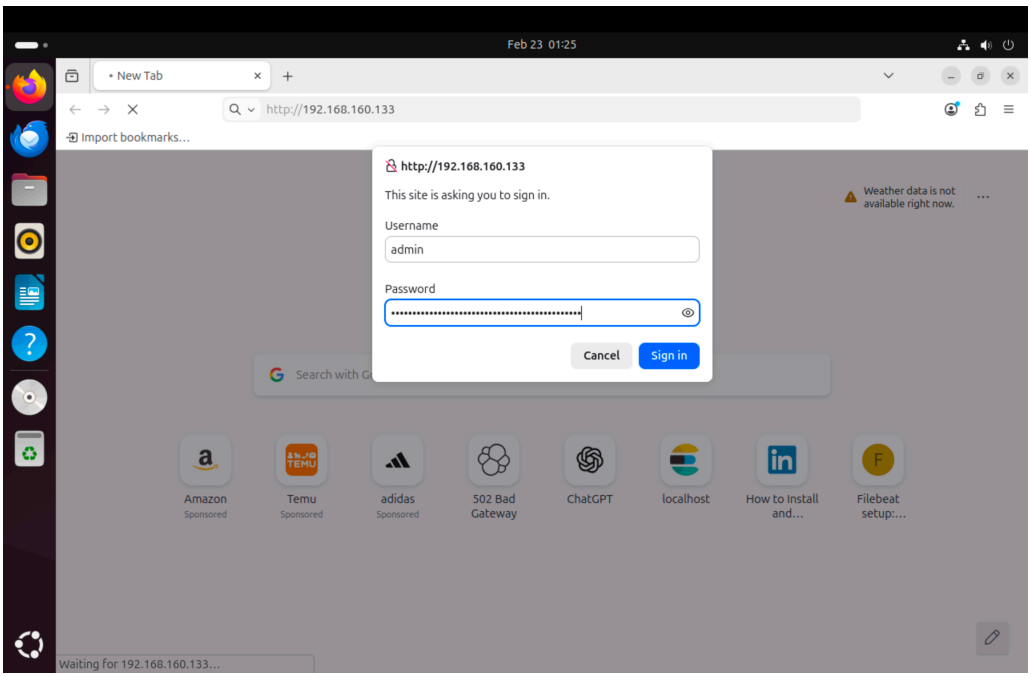
```
sudo systemctl status auditbeat
```



Auditbeat running — dashboards loaded into Kibana

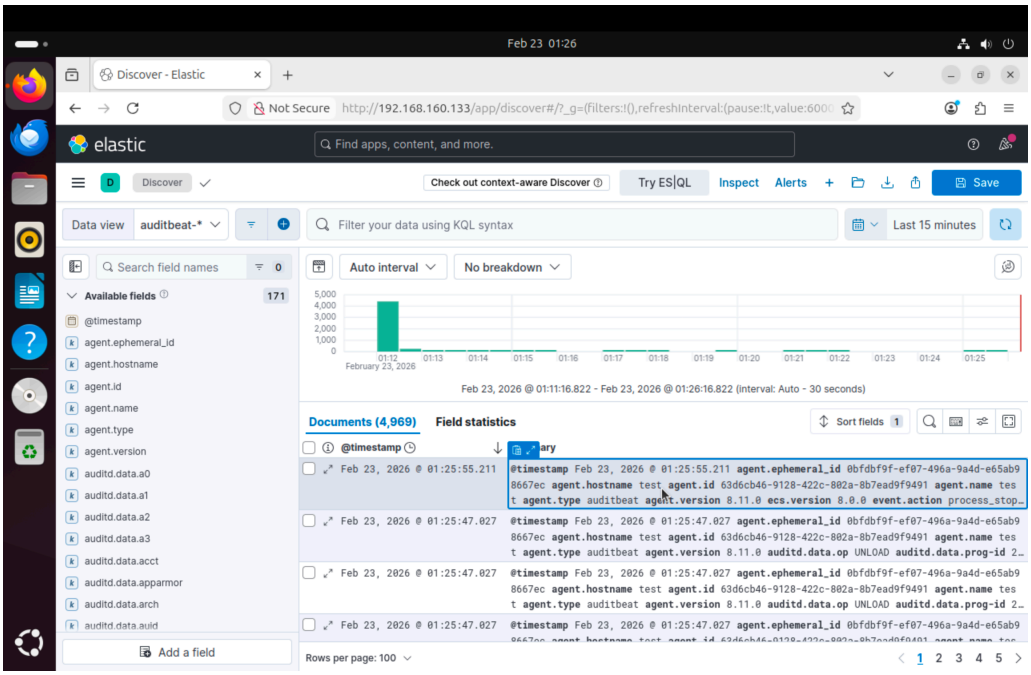
Result — Data Flowing Through the Pipeline

With everything running, I navigated to http://192.168.160.133 and got the Nginx Basic Auth prompt, which confirmed that the reverse proxy was working and the dashboard wasn't just openly accessible.



Basic Auth prompt from Nginx before reaching Kibana

After logging in, I opened the Discover view in Kibana and selected the auditbeat-* index pattern. The data was already there nearly 5,000 documents indexed in the first 15 minutes, with 171 available fields covering everything from process names to network socket details. Every event had a timestamp, the hostname it came from, and full context about what happened.



Kibana Discover — 4,969 auditbeat documents already indexed

The full pipeline was working: Auditbeat collecting events at the kernel level, Logstash receiving and forwarding them, Elasticsearch indexing and storing everything, Kibana making it all searchable, and Nginx handling the authentication layer in front. From here the next step is building custom dashboards and writing KQL queries to search through the data and spot anything interesting.

What's Next

This project gave me a solid foundation — a working SIEM pipeline that collects real system data, indexes it, and makes it searchable. But there's a lot more I want to build on top of it, and I'll be documenting everything the same way as I go.

The next writeup will cover detection rules and how to actually use the Security app in Kibana — creating custom rules, triggering alerts by simulating real attack techniques, and walking through what an alert investigation looks like from start to finish. It's the part that turns this from a monitoring setup into something that can actively detect threats.

After that I want to integrate Wazuh, which is an open-source EDR and SIEM platform that pairs well with the ELK Stack. Wazuh adds host-based intrusion detection, file integrity monitoring, vulnerability detection, and a dedicated agent that goes much deeper than Auditbeat alone. Setting it up alongside what's already here would make this a genuinely complete security lab.

Further down the line: adding a second VM as a target machine to monitor, simulating attack scenarios with tools like Hydra or Metasploit, and building Kibana dashboards that visualise the results in a way that tells a clear story. Everything will be documented step by step and added to this repository as it gets done.