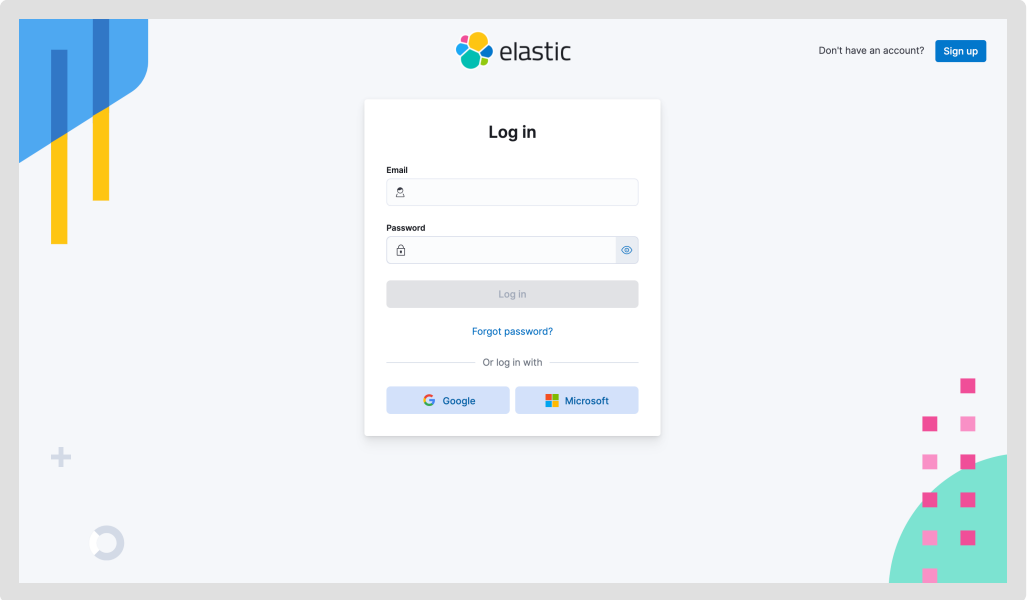
Lab Guide: Observability Fundamentals

Lab 1: Elastic Observability

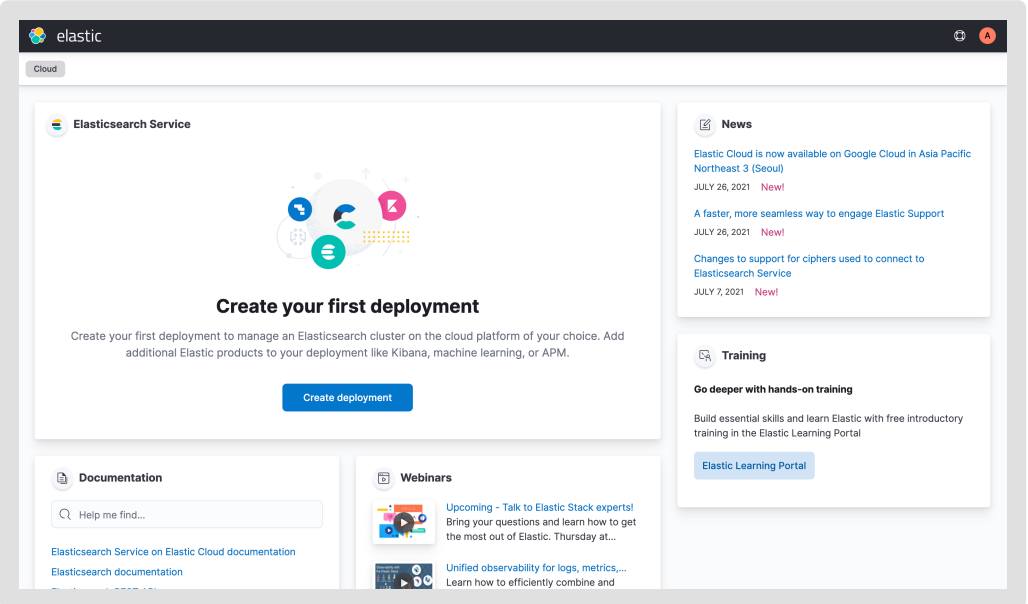
Objective: In this lab, you are going to familiarize yourself with the Elastic Cloud Elasticsearch Service. The Elastic Cloud Elasticsearch Service is an Elasticsearch-as-a-Service product which eases the creation, configuration, and maintenance of large and complex infrastructures.

1. First of all, you need to log in to Elastic Cloud Elasticsearch Service. Click here to open the main page. If you already have a cloud account just log in. If you do not have one, click on the link **Sign up**, enter your email as well as (strong!)

password and click **Create account**.

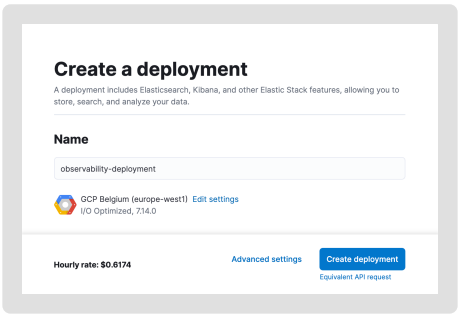


2. After login you will reach the Elastic Cloud home page. There you will find information about deployments, news, training, documentation, and webinars. Note that you do not have any deployments because you just created your account. Click on **Start your free trial** (or **Create deployment**) to create your first deployment.

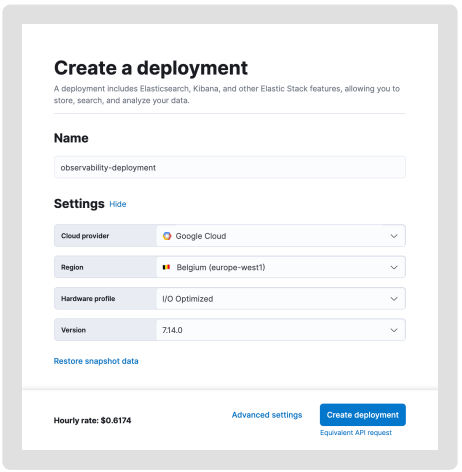


3. To create your first deployment you will need to give it a meaningful name. For this training we will be calling it

```
observability-deployment .
```

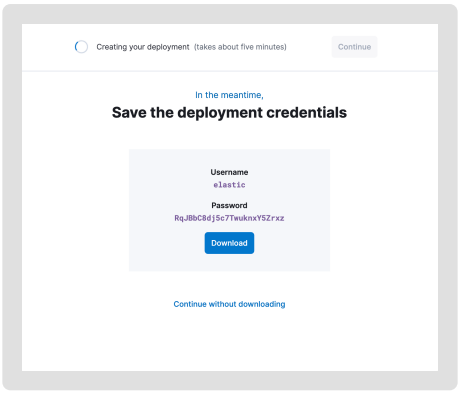


4. You will also need to select a cloud provider and region, choose a hardware profile, and define the Elastic Stack version that you will be using. For this training select the **Google Cloud Platform**, but note that you might also select either AWS or Azure for your future deployments. Also select one region that is close to you and the latest Elastic Stack version that is available. For hardware profile you can use **I/O Optimized**, which is ideal for observability data.



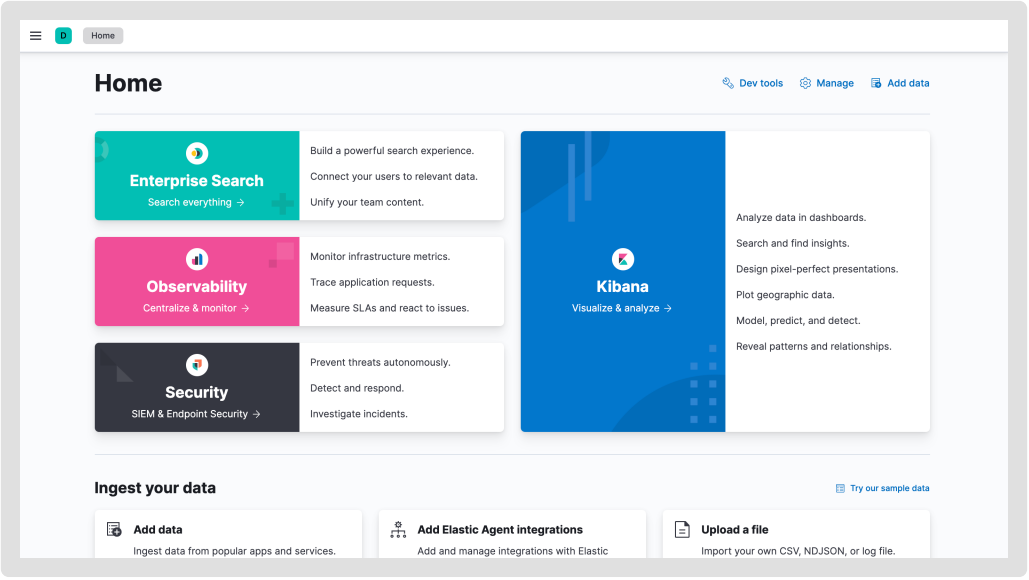
5. When you are all set click on **Create deployment**.

6. Once you click create deployment, you will see a username and password for accessing both Elasticsearch and Kibana. You should download or copy the password to a safe place. You can use the `elastic` credentials to easily send data to your deployment.



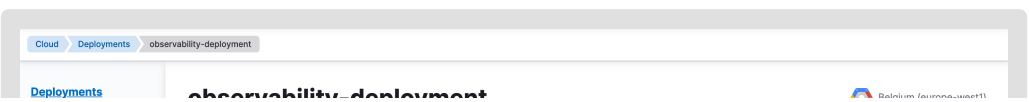
7. It will take a few minutes until your deployment is up and running. When it is ready click on **Continue** to open Kibana.

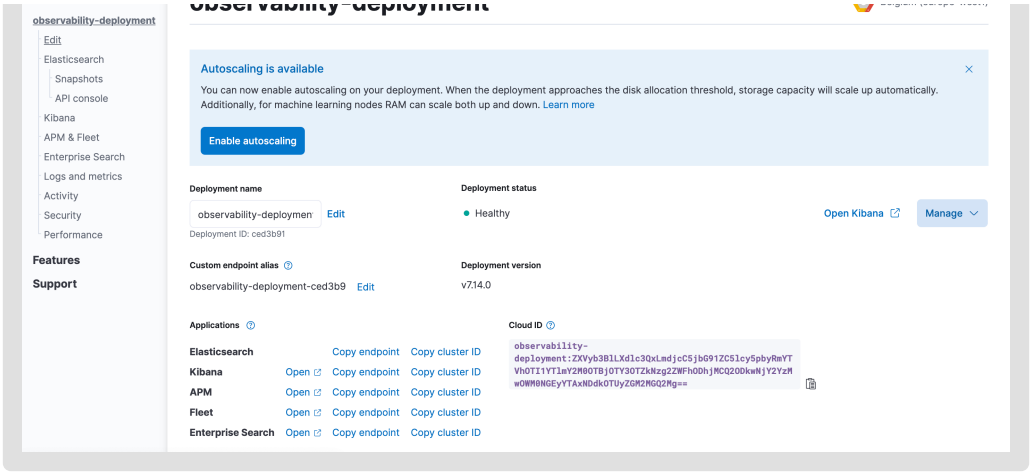
8. After successfully logging in to Kibana you should see a landing page, which looks like this:



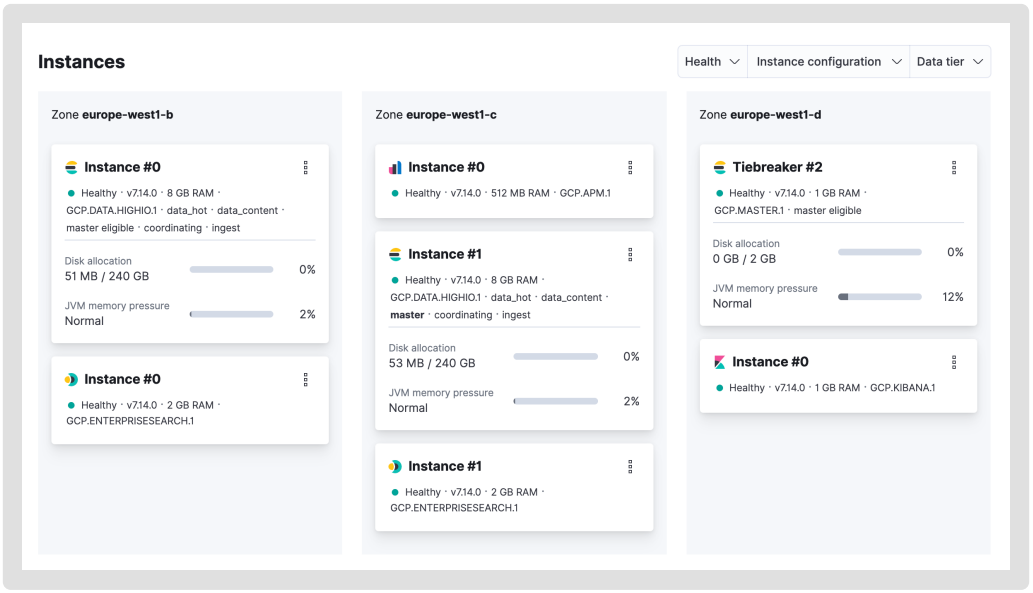
9. Feel free to explore Kibana.

10. If you close Kibana and want to get back to it, you can find the Kibana link by accessing your deployment on Elastic Cloud. There you will also find all the deployment information.





Scroll down to check the information about your deployment and answer some questions about it. Note that whenever you want to check your deployment information you just need to click on **Deployments** on the navigation menu and then access `observability-deployment`.



- What is the Elasticsearch endpoint of your deployment?

  `Show answer`

- What is your Cloud ID?

  `Show answer`

- What is the Kibana endpoint?

  `Show answer`

- What is the APM Server URL?

  Show answer

- How many Elasticsearch nodes are in your cluster?

  Show answer

  In addition to the information above, you can also check what is the Elastic Stack version that your cluster is running, how much memory each node has, what is the JVM memory pressure, what is the current disk usage, etc.

Summary: You created an Elastic Cloud deployment in just a few clicks and now it's ready to start indexing observability data!

End of Lab 1

Lab 2: Logs

Objective: In this lab, you will explore how easy it is to read NGINX log files with Filebeat and index them into Elasticsearch. You will also explore Kibana dashboards and see how easily you can monitor NGINX logs.

1. First, you will download and install (actually just extract) Filebeat. To do that, open a new terminal window and run the following commands. Note that you should download a Filebeat version that matches your Elastic Cloud deployment,

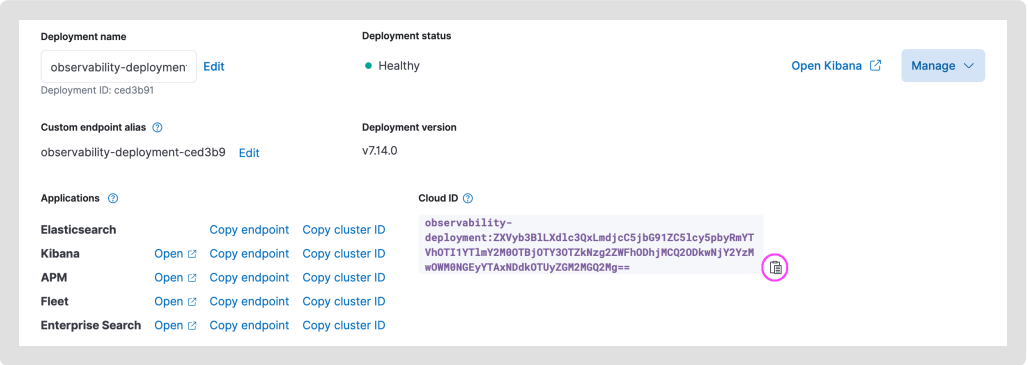
so change `7.14.0` accordingly if your deployment is running on a different version.

```
curl -L -O https://artifacts.elastic.co/downloads/b

tar -xzf filebeat-7.14.0-linux-x86_64.tar.gz

cd filebeat-7.14.0-linux-x86_64
```

2. Next, you will configure Filebeat to output to your cloud instance. You will need the `cloud.id` and the password for the `elastic` user. You can get your `cloud.id` by accessing your `observability-deployment` and clicking on the copy to clipboard button.



3. Did you remember to copy the `elastic` user's password somewhere safe? Great! (If not, you can always reset the `elastic` user password; just remember that you might need to change any applications which have that password stored.)

Show answer

4. Now, edit `filebeat.yml` to configure Filebeat to output to your Elastic Cloud deployment. Insert the following two lines at the end of the config file:

```
cloud.id: "YOUR_CLOUD_ID"
cloud.auth: "elastic:YOUR_CLOUD_PASSWORD"
```

You can use your preferred text editor. If you are not familiar with terminal text editors, click here to see how to use nano.

5. Replace `YOUR_CLOUD_ID` and `YOUR_CLOUD_PASSWORD` with the Cloud ID and the password for the `elastic` user of your deployment, respectively.

Show answer

6. Test whether Filebeat can reach your deployment.

```
./filebeat test output
```

You should an output similar to the one below:

```
elasticsearch: https://196fdefef0f64d08a64fe63bff7a
  parse url... OK
  connection...
    parse host... OK
    dns lookup... OK
    addresses: 35.195.130.253
    dial up... OK
  TLS...
    security: server's certificate chain verificati
    handshake... OK
    TLS version: TLSv1.2
    dial up... OK
  talk to server... OK
  version: 7.14.0
```

7. Next, enable the NGINX module.

```
./filebeat modules enable nginx
```

8. Use the following command to check the NGINX configuration:

```
cat modules.d/nginx.yml
```

You should see the following configurations:

```
# Module: nginx
# Docs: https://www.elastic.co/guide/en/beats/fileb

- module: nginx
  # Access logs
  access:
    enabled: true

    # Set custom paths for the log files. If left e
    # Filebeat will choose the paths depending on y
    #var.paths:

  # Error logs
  error:
    enabled: true

    # Set custom paths for the log files. If left e
    # Filebeat will choose the paths depending on y
    #var.paths:

  # Ingress-nginx controller logs. This is disabled
  ingress_controller:
```

```
    enabled: false


    # Set custom paths for the log files. If left e
    # Filebeat will choose the paths depending on y
    #var.paths:
```

The configuration file has three parts. The first and second parts collect access and error logs from the default paths `/var/log/nginx/access.log` and `/var/log/nginx/error.log`, respectively. They are enabled by default and you can use `var.paths` to change paths in case your NGINX logs are not stored in the default path. The third part is disabled by default, but you can enable it in case you are using `ingress-nginx` as an ingress controller for your Kubernetes environment through NGINX as a reserve proxy and load balancer.

9. Load the Elasticsearch index templates, Kibana dashboards, and ingest pipelines. (*This might take a minute to run.*)

```
./filebeat setup --dashboards --index-management --
```

10. Run Filebeat to start collecting logs. (*The `-e` option logs to the terminal instead of syslog/file output.*)

```
./filebeat -e
```

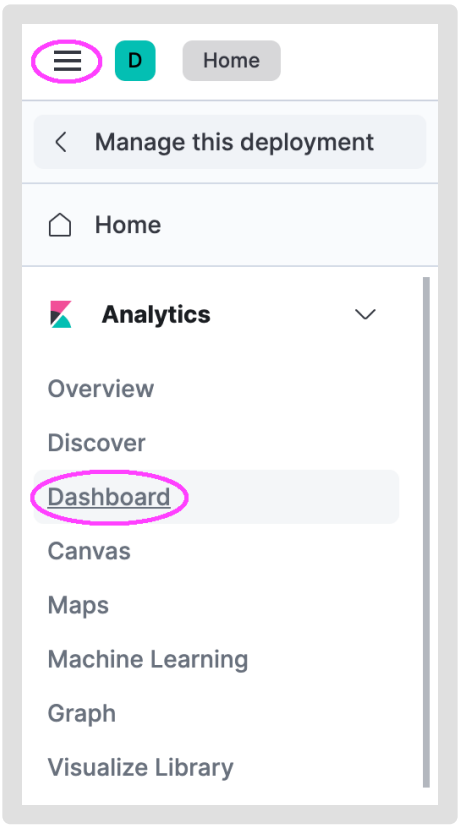
11. Now, open a new terminal window and run the following script to simulate load on your NGINX server.

```
./artificial_load.sh
```

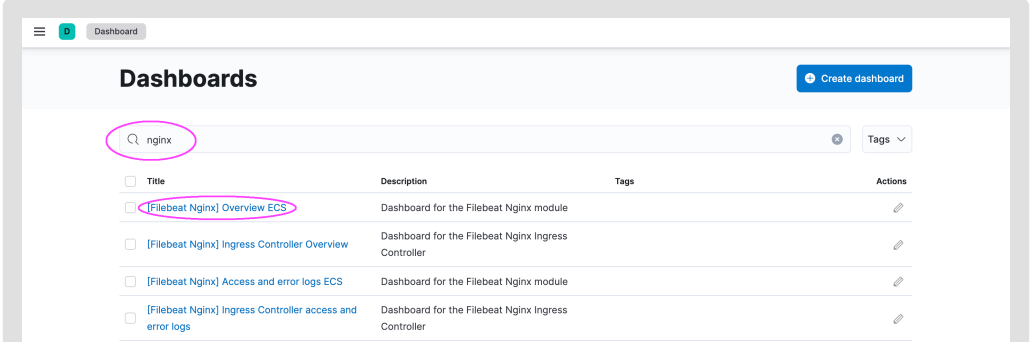
12. Launch Kibana to start exploring dashboards that were included as part of the setup process.

Show answer

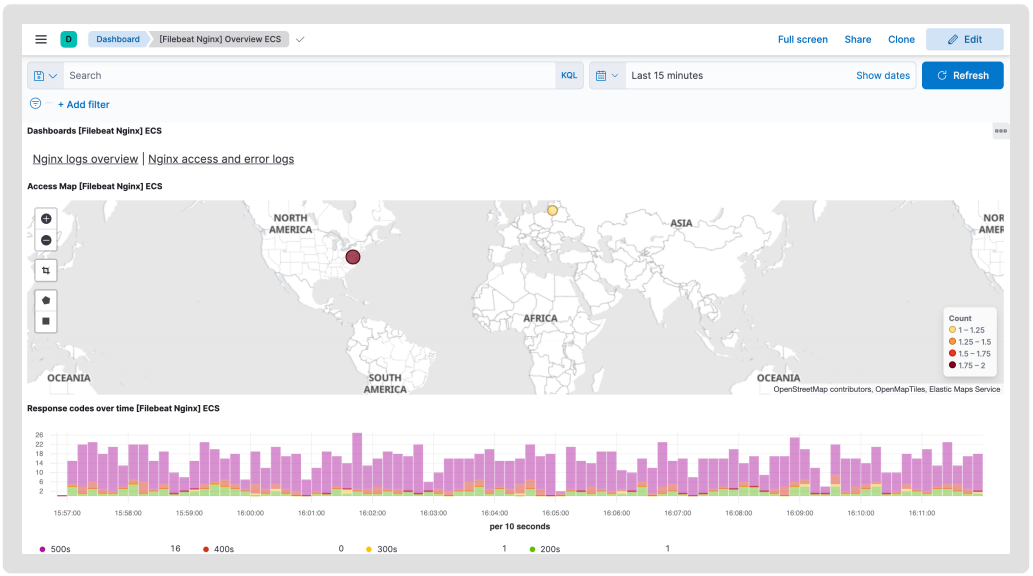
13. Then, access **Dashboard** through the main menu.



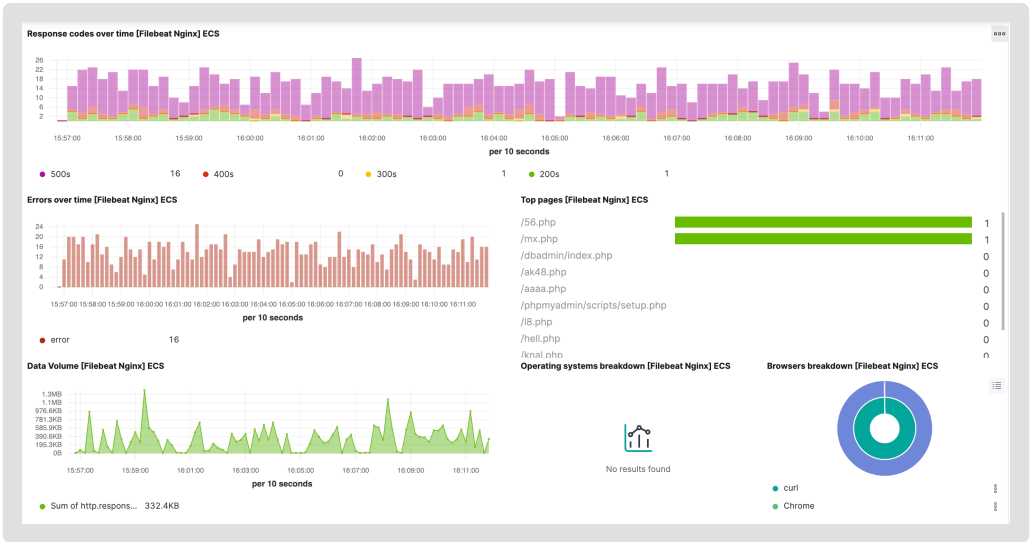
14. Search for **nginx** and open the *[Filebeat Nginx] Overview ECS dashboard*.



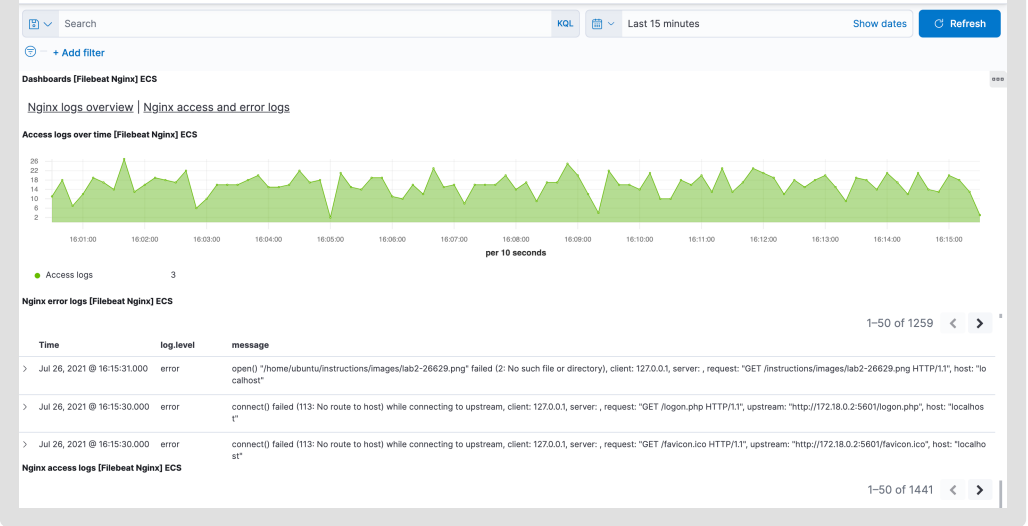
You will see something similar to the page below.



15. Scroll down and check what is the most used browser. You will note that it should be `curl` because it is being used by the load simulation script.



16. Now, click on *Nginx access and error logs* at the top of the current dashboard to see the dashboard with the access and error logs that Filebeat collected from NGINX.



Summary: In this lab, you explored how easy it is to read NGINX log files with Filebeat and index them into Elasticsearch. You also explored Kibana dashboards and saw how easily you can monitor NGINX logs.

End of Lab 2

Lab 3: Metrics

Objective: In this lab, you will explore how easy it is to get system and NGINX metrics with Metricbeat and index them into Elasticsearch. You will also explore Kibana dashboards and see how easily you can monitor these metrics.

1. First, you will download and install (actually just extract) Metricbeat. To do that, open a new terminal window and run the following commands. Note that you should download a Metricbeat version that matches your Elastic Cloud deployment, so change `7.14.0` accordingly if your

deployment is running on a different version.

```
curl -L -O https://artifacts.elastic.co/downloads/b

tar -xzf metricbeat-7.14.0-linux-x86_64.tar.gz

cd metricbeat-7.14.0-linux-x86_64
```

2. Next, you will configure Metricbeat to output to your cloud instance. You will need the `cloud.id` and the password for the `elastic` user. You can get your `cloud.id` by accessing your `observability-deployment` and clicking on the copy to clipboard button.

3. Did you remember to copy the `elastic` user's password somewhere safe? Great! (If not, you can always reset the `elastic` user password; just remember that you might need to change any applications which have that password stored.)

Show answer

4. Now, edit `metricbeat.yml` to configure Metricbeat to output to your Elastic Cloud deployment. Insert the following two lines at the end of the config file:

```
cloud.id: "YOUR_CLOUD_ID"
cloud.auth: "elastic:YOUR_CLOUD_PASSWORD"
```

You can use your preferred text editor. If you are not familiar with terminal text editors, click here to see how to use nano.

5. Replace `YOUR_CLOUD_ID` and `YOUR_CLOUD_PASSWORD` with the Cloud ID and the password for the `elastic` user of your deployment, respectively.

Show answer

6. Test whether Metricbeat can reach your deployment.

```
./metricbeat test output
```

You should an output similar to the one below:

```
elasticsearch: https://196fdefef0f64d08a64fe63bff7a
  parse url... OK
  connection...
    parse host... OK
    dns lookup... OK
    addresses: 35.195.130.253
    dial up... OK
  TLS...
    security: server's certificate chain verificati
    handshake... OK
    TLS version: TLSv1.2
    dial up... OK
  talk to server... OK
  version: 7.14.0
```

7. The system module is already enabled by default. Use the

following command to check the its configuration:

```
cat modules.d/system.yml
```

You should see the following configurations:

```
# Module: system
# Docs: https://www.elastic.co/guide/en/beats/metri

- module: system
  period: 10s
  metricsets:
    - cpu
    - load
    - memory
    - network
    - process
    - process_summary
    - socket_summary
    #- entropy
    #- core
    #- diskio
    #- socket
    #- service
    #- users
  process.include_top_n:
    by_cpu: 5      # include top 5 processes by CPU
    by_memory: 5   # include top 5 processes by mem
  # Configure the mount point of the host's filesys
  #system.hostfs: "/hostfs"

- module: system
```

```
  period: 1m
  metricsets:
    - filesystem
    - fsstat
  processors:
  - drop_event.when.regexp:
      system.filesystem.mount_point: '^/(sys|cgroup

- module: system
  period: 15m
  metricsets:
    - uptime

#- module: system
#  period: 5m
#  metricsets:
#    - raid
#  raid.mount_point: '/'
```

The configuration file has three parts. The first one collects metrics about CPU usage, CPU load average, memory usage, network I/O, processes and sockets every ten seconds. The second one collects file system statistics every minute. The third one collects system uptime every fifteen minutes.

8. Next, enable the NGINX module.

```
./metricbeat modules enable nginx
```

9. Use the following command to check the NGINX

configuration:

```
cat modules.d/nginx.yml
```

You should see the following configurations:

```
# Module: nginx
# Docs: https://www.elastic.co/guide/en/beats/metri

- module: nginx
  #metricsets:
  #   - stubstatus
  period: 10s

  # Nginx hosts
  hosts: ["http://127.0.0.1"]

  # Path to server status. Default server-status
  #server_status_path: "server-status"

  #username: "user"
  #password: "secret"
```

This configuration file basically tells Metricbeat which is the server host as well as its status path for collecting NGINX metrics every ten seconds. However, note that it assumes NGINX does not have authentication, which is not the case of your lab environment.

10. Edit `modules.d/nginx.yml` to uncomment `username` and `password`. Also change `user` by `training` and `secret` by `nonprodpwd`.

Show answer

11. Load the Elasticsearch index templates and Kibana dashboards. (*This might take a minute to run.*)

```
./metricbeat setup
```

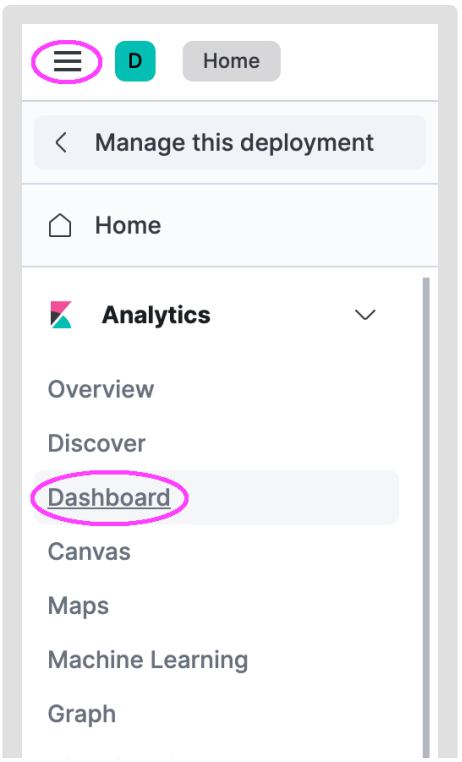
12. Run Metricbeat to start collecting metrics. (*The `-e` option logs to the terminal instead of syslog/file output.*)

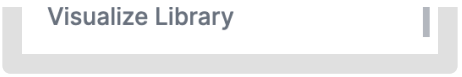
```
./metricbeat -e
```

13. Launch Kibana to start exploring dashboards that were included as part of the setup process.

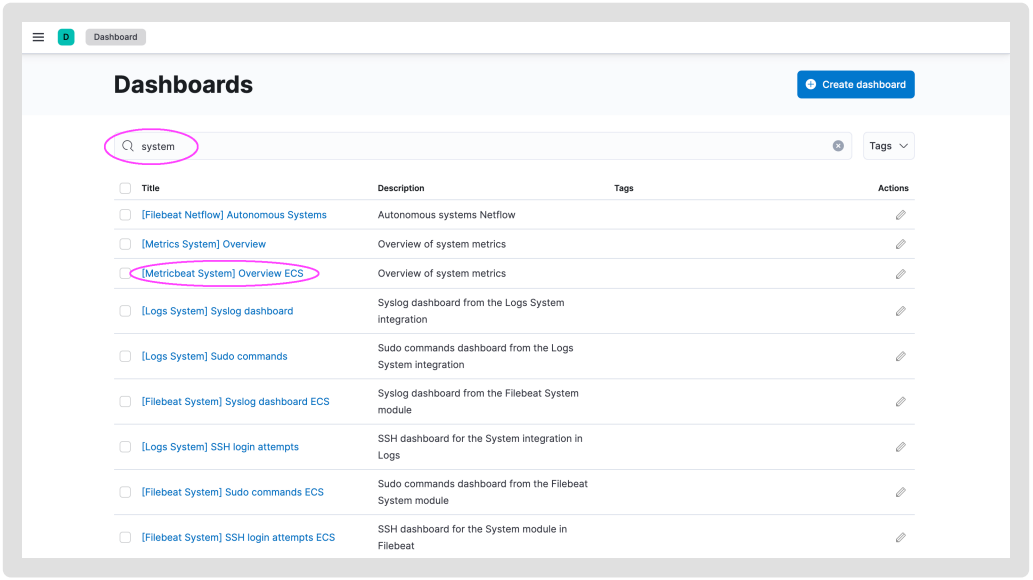
Show answer

14. Then, access **Dashboard** through the main menu.

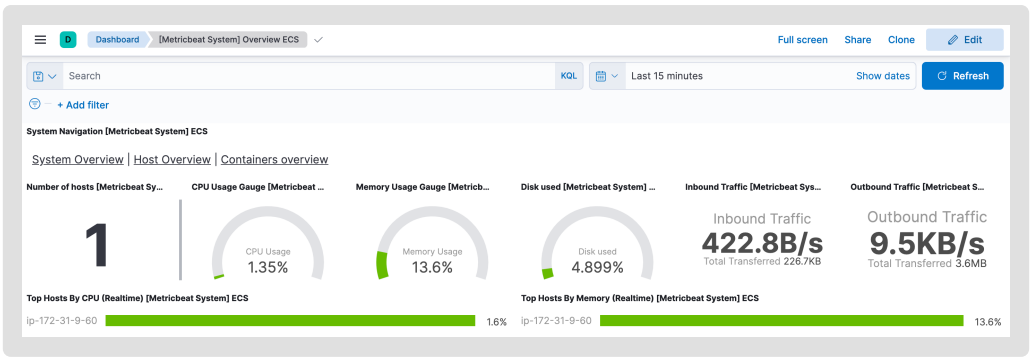




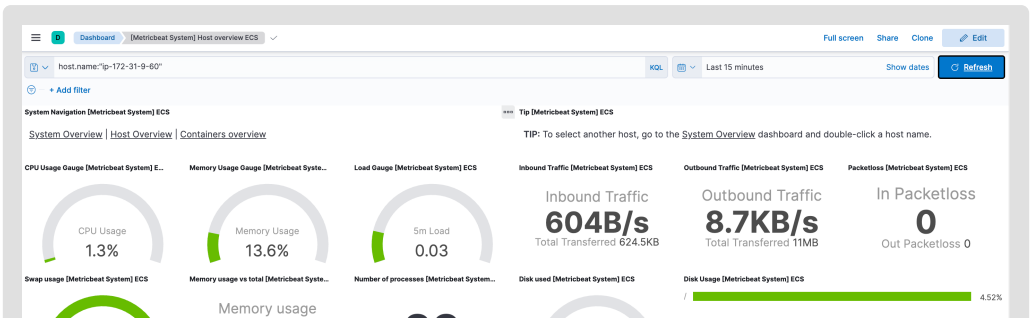
15. Search for **system** and open the *[Metricbeat System] Overview ECS*.

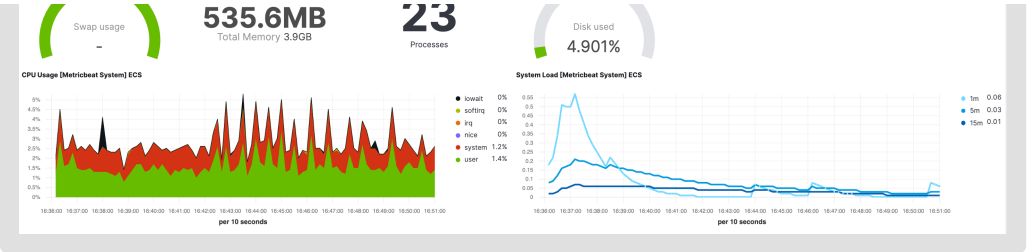


You will see something similar to the page below.

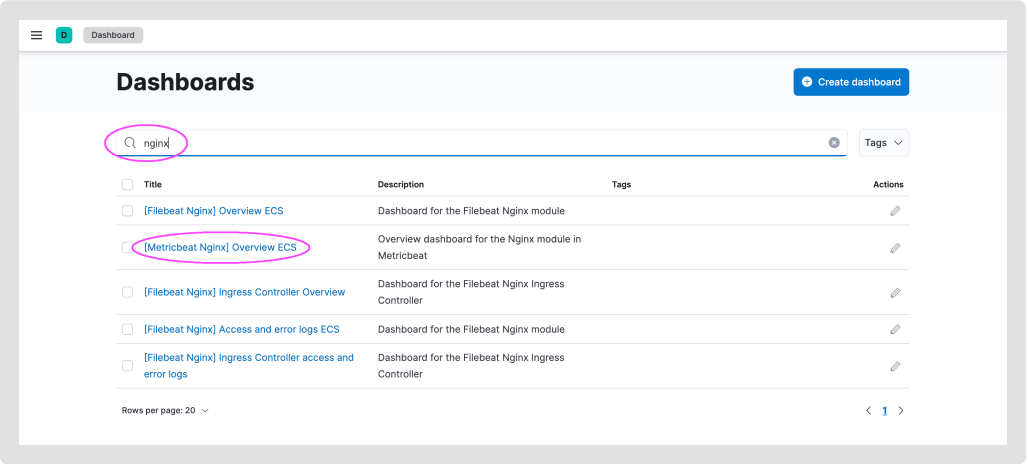


16. Click on *Host Overview* at the top of the current dashboard to see a dashboard with more information that Metricbeat collected from your lab environment host.

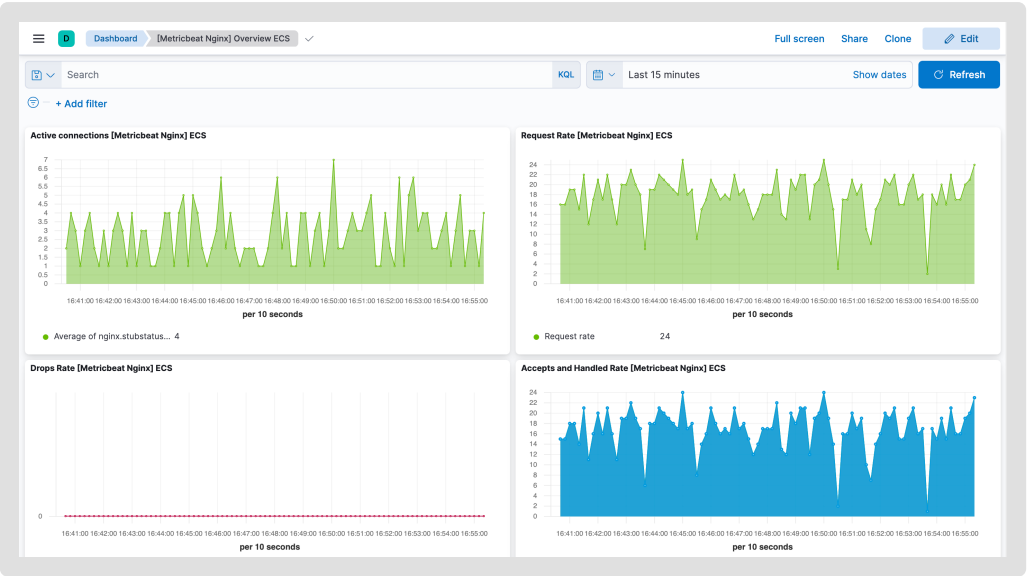




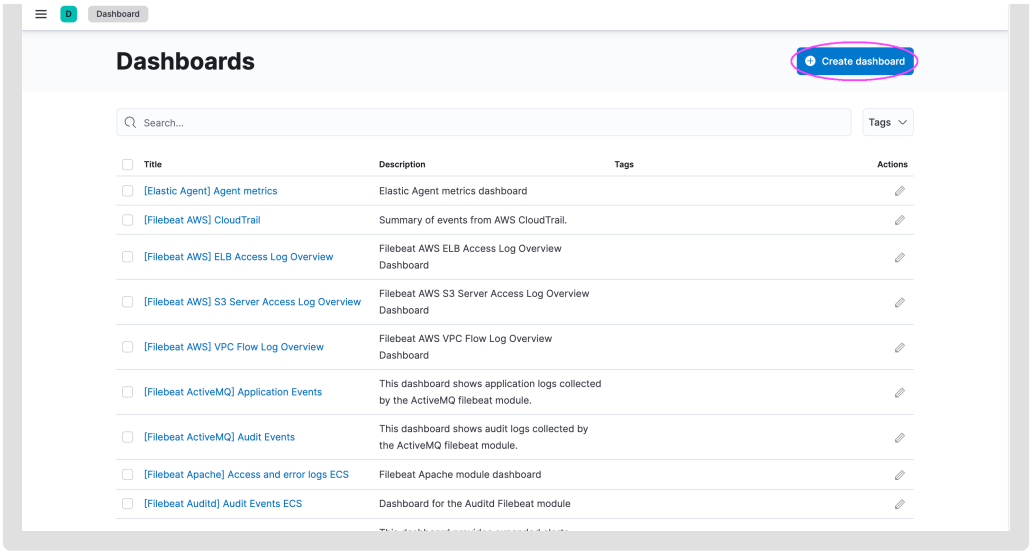
17. Go back to the Kibana dashboards page, search for **nginx** and open the *[Metricbeat Nginx] Overview ECS* dashboard.



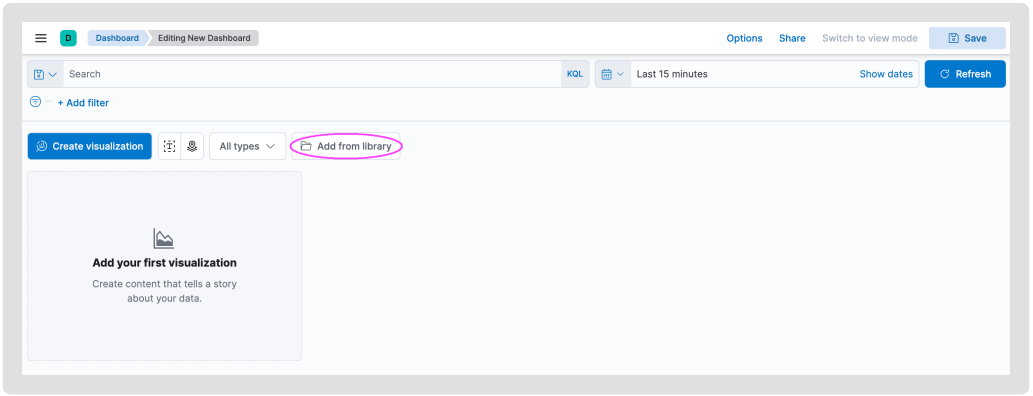
You will see something similar to the page below.



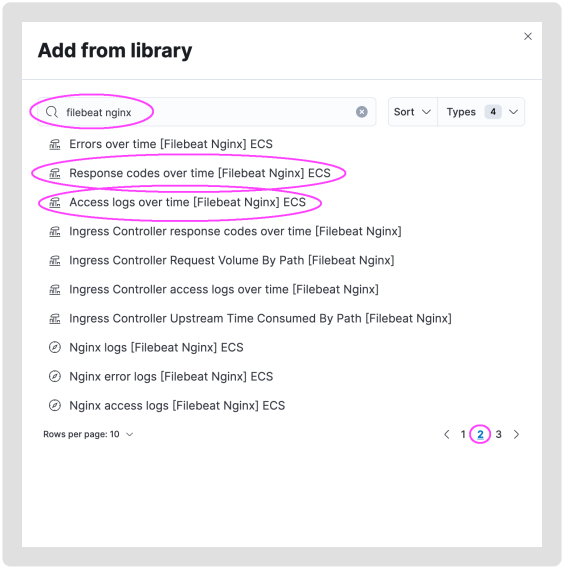
18. Go back to the Kibana dashboards and click on **Create new dashboard** to compare the NGINX data collected from Filebeat and Metricbeat.



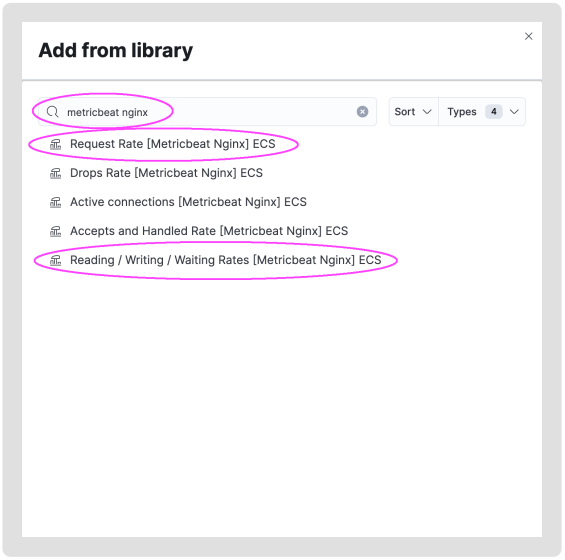
Click **Add from library** to start adding visualizations to your dashboard.



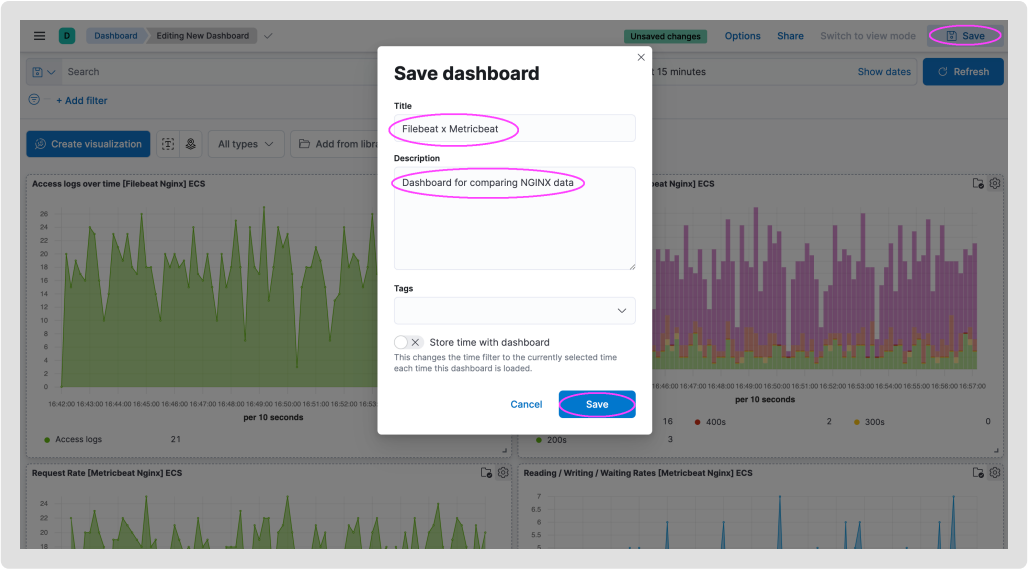
Use the query bar to filter by **filebeat nginx** visualizations and add the following Filebeat visualizations: **Access logs over time** and **Response codes over time**.



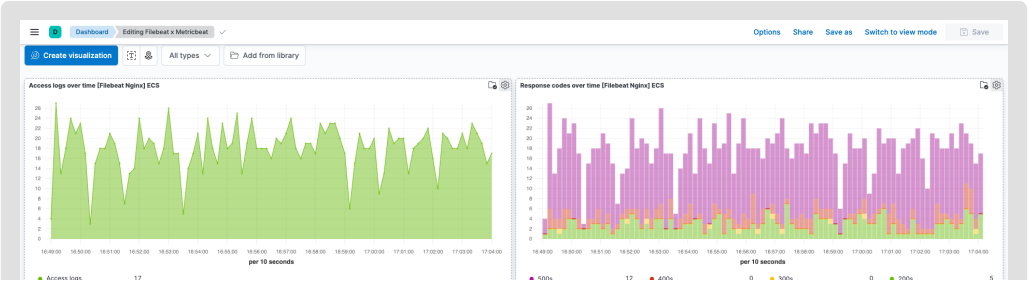
Now filter by **metricbeat nginx** visualizations to add the following Metricbeat visualizations: **Request Rate** and **Reading / Writing / Waiting Rates**.

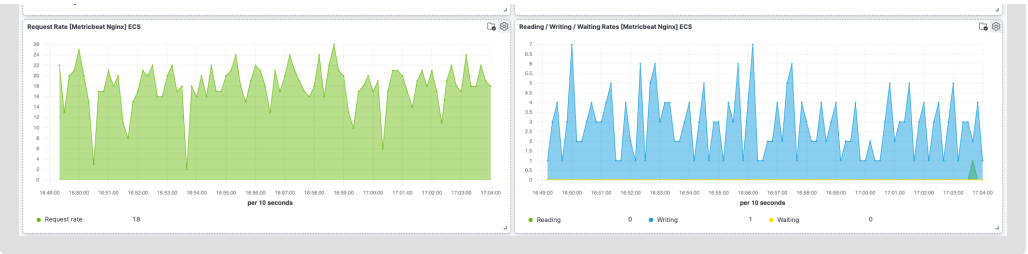


Close the **Add from library** dialog and save your dashboard as **Filebeat x Metricbeat**.



Check the different information you can get from NGINX logs and metrics in the dashboard you just created.





Notice that you can use both Filebeat and Metricbeat data to check how many requests per seconds you have, as shown in **Access logs over time** and **Request Rate** visualizations. The open source version of NGINX does not provide metrics about response codes, but you can use Filebeat to get this information from NGINX logs, as you can see in the **Response codes over time** visualization. With NGINX logs you can also check the location, operating system, browser, duration, and many other information about the requests. However, NGINX logs do not tell you how many connections were accepted, handled, and dropped. In this case, you can use Metricbeat to collect this information as well as the number of active connections and how many connections are reading, writing, and waiting, as you can see in the **Reading** / **Writing** / **Waiting Rate** visualization.

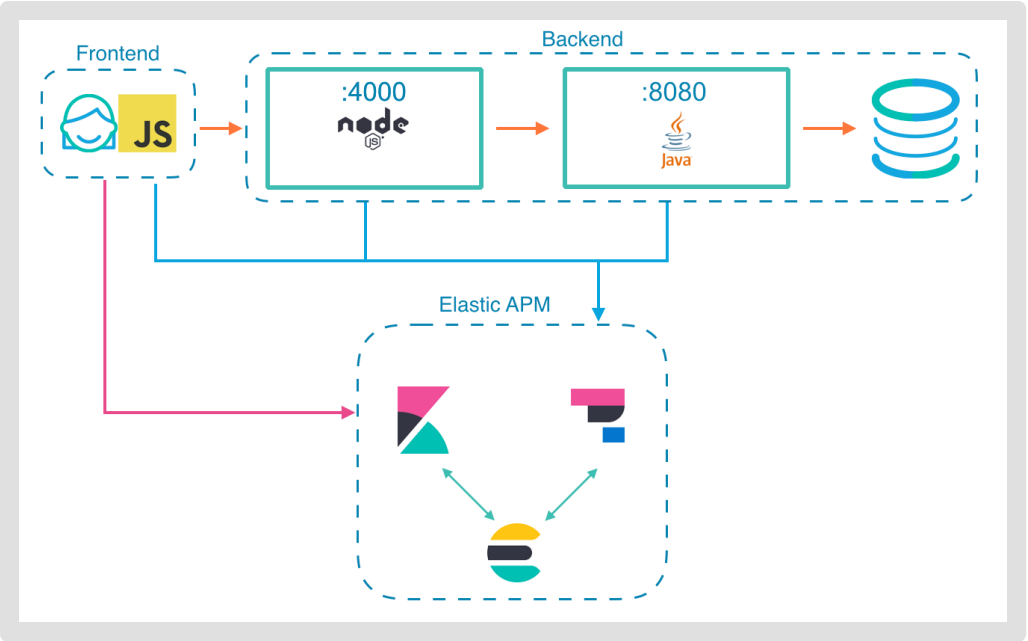
Summary: In this lab, you explored how easy it is to get system and NGINX metrics with Metricbeat and index them into Elasticsearch. You also explored Kibana dashboards and saw how easily you can monitor these metrics.

End of Lab 3

Lab 4: APM

Objective: In this lab, you will learn how easy it is to use APM to instrument applications for collecting detailed performance information as well as errors and send them to your Elasticsearch deployment. You will also explore the Kibana APM app and see how easily you can monitor application performance.

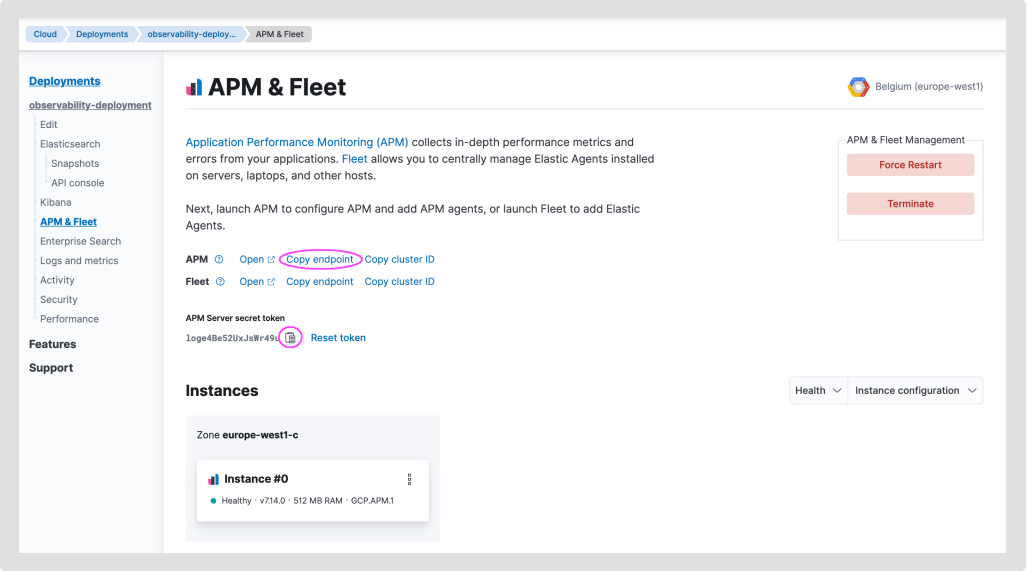
1. Before you get started, review the architecture you are implementing through the lab steps as described in the picture below.



You already have your Elasticsearch deployment running with APM Server set to send data there and Kibana to retrieve data from there. In the next steps you will instrument Petclinic to then explore distributed tracing through the APM app in Kibana. Petclinic is a demo application composed by several different applications and services. The frontend is a **React** application that runs on the client's browser. The frontend is served by a **Node.js** backend server that listens on port `4000`. The backend server proxies all frontend requests to the application core, so it can handle the requests. The application core is a REST API implementation that connects to the databases. It is implemented with the **Spring** framework and listens on port `8080`. To monitor this

architecture, you will use the **Java** agent for the application core, the **Node.js** agent for the backend server, and the **RUM** agent for the frontend.

2. First, you need to get the APM Server endpoint and secret token from Elastic Cloud, so you can configure the APM agents to send data to your deployment. You can access these information by clicking on **APM & Fleet** at the side navigation of your `observability-deployment`.



Copy both APM Server endpoint and secret token somewhere, as you will need them in the next lab steps.

3. Next, you will start the application core. To do that, open a new terminal.

4. The next step would be downloading the latest release of the agent jar file, but the lab environment already has the version you need to run the labs.

```
ls petclinic/elastic-apm-agent-1.21.0.jar
```

5. Use the following command to start the Petclinic application core. Before running it replace `APM_SERVER_ENDPOINT` and `APM_SERVER_TOKEN` with the APM Server endpoint and secret token you copied from your `observability-deployment`.

```
java -javaagent:/home/elastic/petclinic/elastic-apm
  -Delastic.apm.service_name=petclinic-spring \
  -Delastic.apm.server_urls=APM_SERVER_ENDPOINT:443
  -Delastic.apm.secret_token=APM_SERVER_TOKEN \
  -Delastic.apm.environment=production \
  -Delastic.apm.application_packages=org.springfram
  -jar /home/elastic/petclinic/spring-petclinic-1.5
```

Note that you don't need to declare a dependency to the agent in your application, but you only need to download the agent and add the `-javaagent` flag with the path to the jar agent when starting your application. You also need to specify the `elastic.apm.service_name` setting as the name of the service that will appear in the APM app, while `elastic.apm.server_urls` is the setting that defines where the APM Server is running and `elastic.apm.secret_token` provides the credentials to access the APM Server. The `elastic.apm.environment` and `elastic.apm.application_packages` settings are optional, though the former helps navigating through APM data from a specific environment when you have more than one, while the latter helps the APM app to collapse the stack frames of library code and highlight the stack frames originated from your application.

Show answer

6. Next, you will start the application backend that serves the frontend. To do that, open a new terminal window.

7. Access the `petclinic/frontend` directory:

```
cd petclinic/frontend
```

8. Both the backend server and the frontend already have installed the Node.js and RUM agents, respectively. This means that you only need to make sure their configurations are correct before starting the backend server. So, check how the Node.js agent is required and started at the top of the `bin/www` main file.

```
head bin/www
```

You should see the following configurations:

```
#!/usr/bin/env node
const settings = require('../config')
var apm = require('elastic-apm-node').start({
  serviceName: settings.apm_service_name,
  serviceVersion: settings.apm_service_version,
  serverUrl: settings.apm_server,
  ....
```

Note how the `config.js` file is used to set the `serviceName`, `serviceVersion`, and `serverUrl` settings for the Node.js agent.

9. You also need to specify the `secretToken`, so the Node.js agent can access the APM Server that is running on your cloud deployment. Edit `bin/www` and add the following line to the configuration object.

```
  secretToken: settings.apm_server_token,
```

After editing the top of the `bin/www` file, it should look like this:

```
#!/usr/bin/env node
const settings = require('../config')
var apm = require('elastic-apm-node').start({
  serviceName: settings.apm_service_name,
  serviceVersion: settings.apm_service_version,
  serverUrl: settings.apm_server,
  secretToken: settings.apm_server_token,
  ...
```

10. Check the configurations set by the `config.js` file.

```
cat config.js
```

You should see the following object declaration:

```
var config = {
  apm_server: process.env.ELASTIC_APM_SERVER_URL ||
  apm_server_js: process.env.ELASTIC_APM_SERVER_JS_
  apm_service_name: process.env.ELASTIC_APM_SERVICE
  apm_client_service_name: process.env.ELASTIC_APM_
  apm_service_version: process.env.ELASTIC_APM_SERV
  api_server: process.env.API_SERVER || 'http://loc
  api_prefix: process.env.API_PREFIX || '/petclinic
  address_server: process.env.ADDRESS_SERVER || 'ht
  distributedTracingOrigins: process.env.DISTRIBUTE
}
```

Note how `apm_service_name` , `apm_service_version` , and `apm_server` definitions relate to the Node.js agent configuration in the previous step. You will edit this configuration file in the next steps to make sure the Node.js and RUM agents can reach your APM Server.

11. Start by editing `config.js` to add `apm_server_token` to the object object declaration as follows. Note that you need to replace `APM_SERVER_TOKEN` with the secret token you copied from your `observability-deployment`.

```
    apm_server_token: process.env.ELASTIC_APM_SECRET_
```

Show answer

12. Then, edit `config.js` and change `apm_server` to use your APM Server endpoint with port `443` instead of http://localhost:8200. After doing this you configured the settings that make the Node.js agent be able to communicate with the APM Server running on your deployment.

Show answer

13. Next, edit `config.js` and change `apm_server_js` to use your APM Server endpoint with port `443` instead of http://localhost:8200. After doing this you setup the RUM agent from `public/index.js` to collect APM data in the client's browser and send them to your APM Server.

    IMPORTANT: Note that there are two configurations: `apm_server` and `apm_server_js`. The former is used by the backend server, while the latter is used by the frontend. There are two different configurations because the Node.js and RUM agents might access the APM Server through different endpoints. In particular, the frontend will be running on the client's browser and needs to know how to reach the APM Server through a public address. Also note that the secret token is not applicable for the RUM agent, because there is no way to prevent it from being publicly exposed.

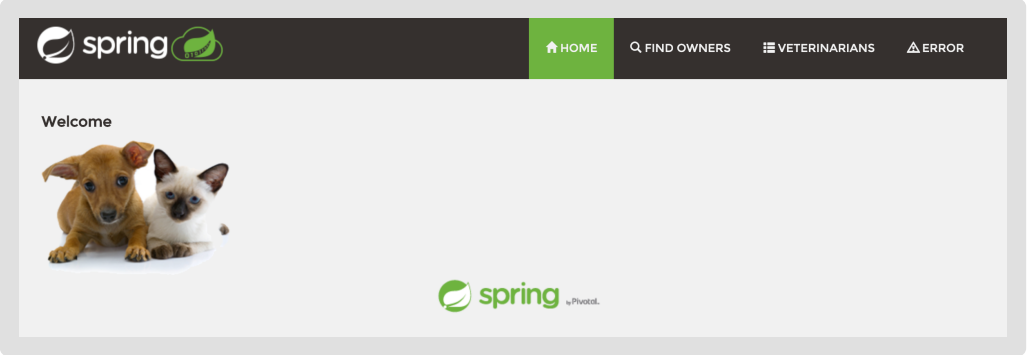
Show answer

14. Start the backend for serving the frontend:

```
npm start
```

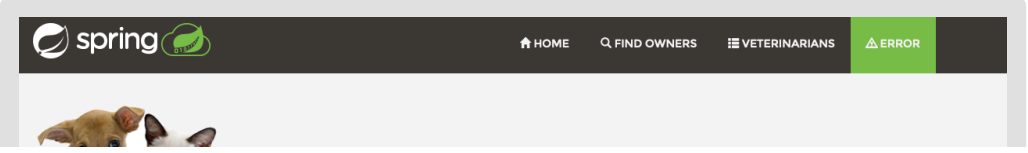
Note that the RUM and Node.js agents are already declared as dependencies to the frontend and backend applications, respectively. The settings `apm_client_service_name` and `apm_service_name` present in the `config.js` file define the name of the service as it will appear in the APM app.

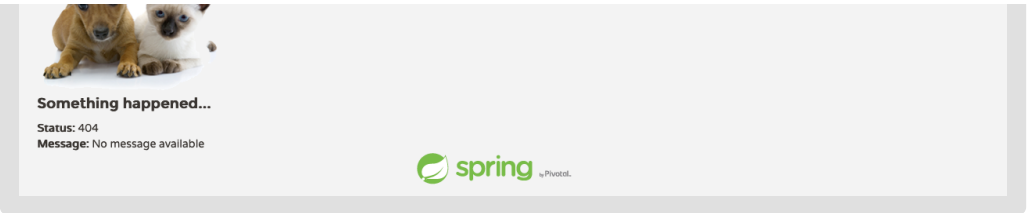
15. Now that all the three micro services are running, access the Petclinic web page and you should see the following home page:



16. Click on **FIND OWNERS** and **VETERINARIANS** to generate performance data to be sent to the APM Server.

17. Click on **ERROR** to generate errors to be sent to the APM Server. Make sure you get the `404` status and the `No message available` error message as follows:



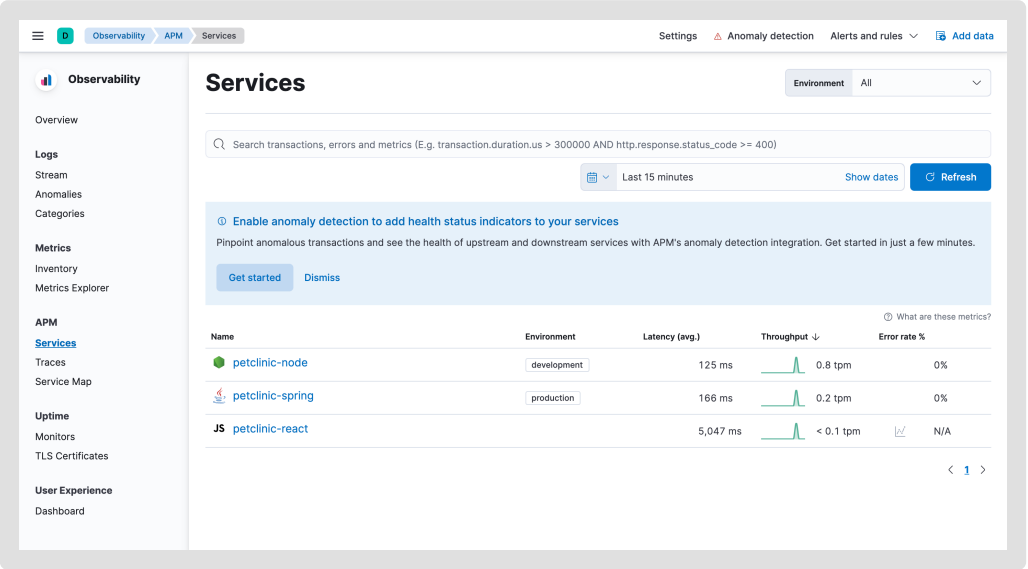


If you don't get this error, click on **HOME** and click back on **ERROR** until you get this error.

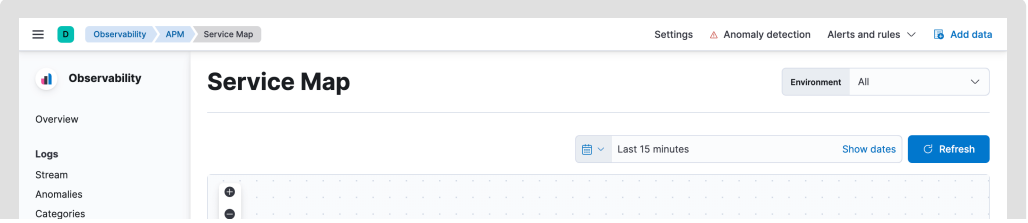
18. Launch the APM app to start exploring the collected data.

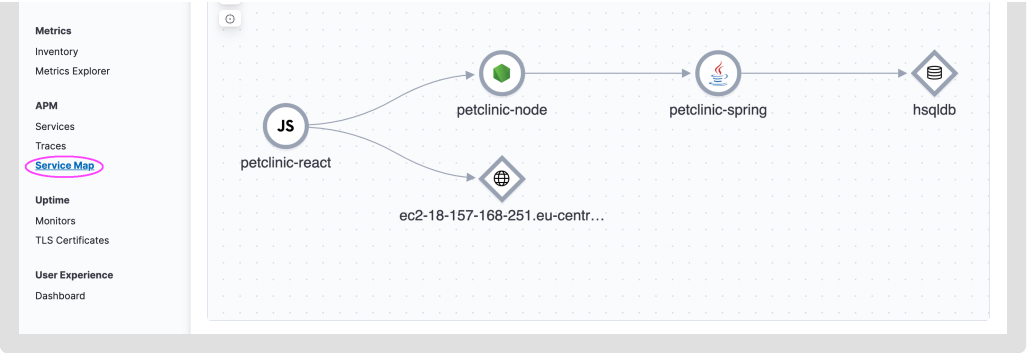
Show answer

19. After launching the APM app you will reach the **Services** overview.

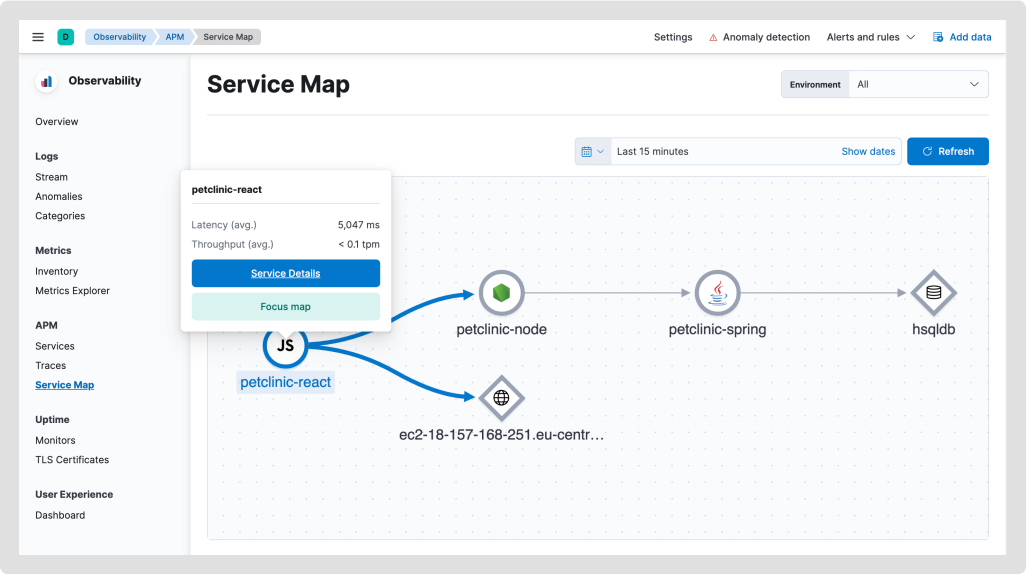


20. Now, click on **Service Map** to check the Petclinic architecture. Note how it describes that **petclinic-react** connects to **petclinic-node** that connects to **petclinic-spring**, which is connected to the database.

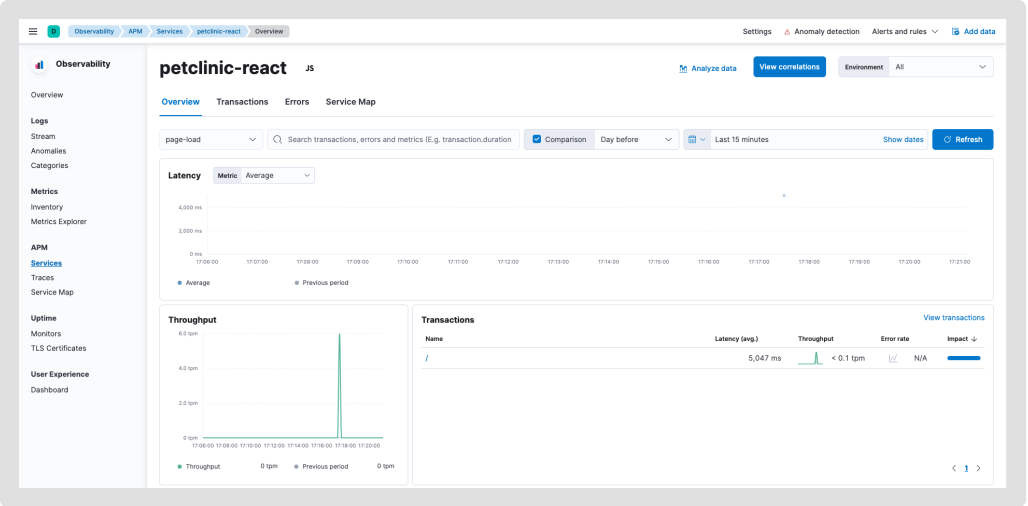




Click on **petclinic-react** to get high-level metrics like average latency and average throughput.

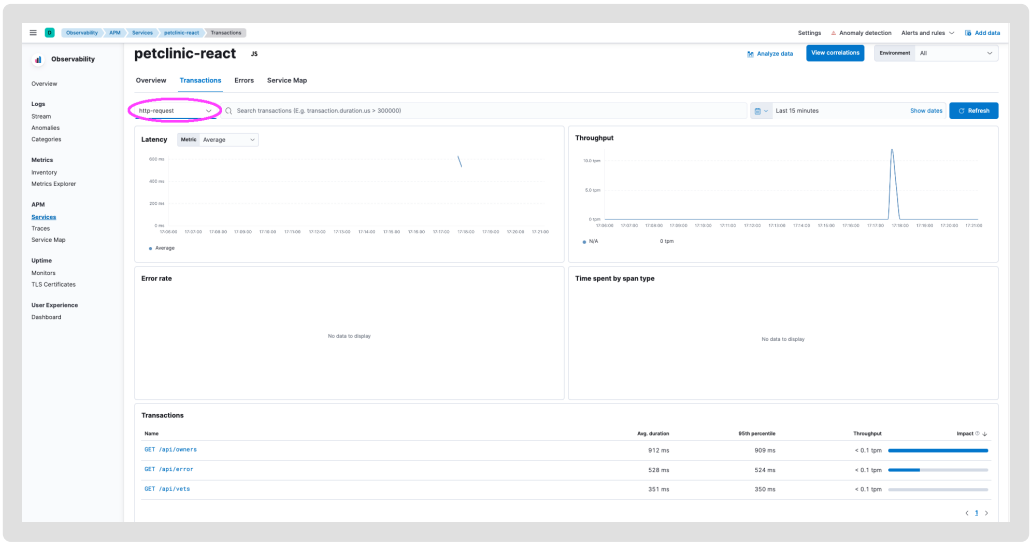


Click on **Service Details** to explore the overall health of the frontend by checking metrics about transactions, errors, and infrastructure.

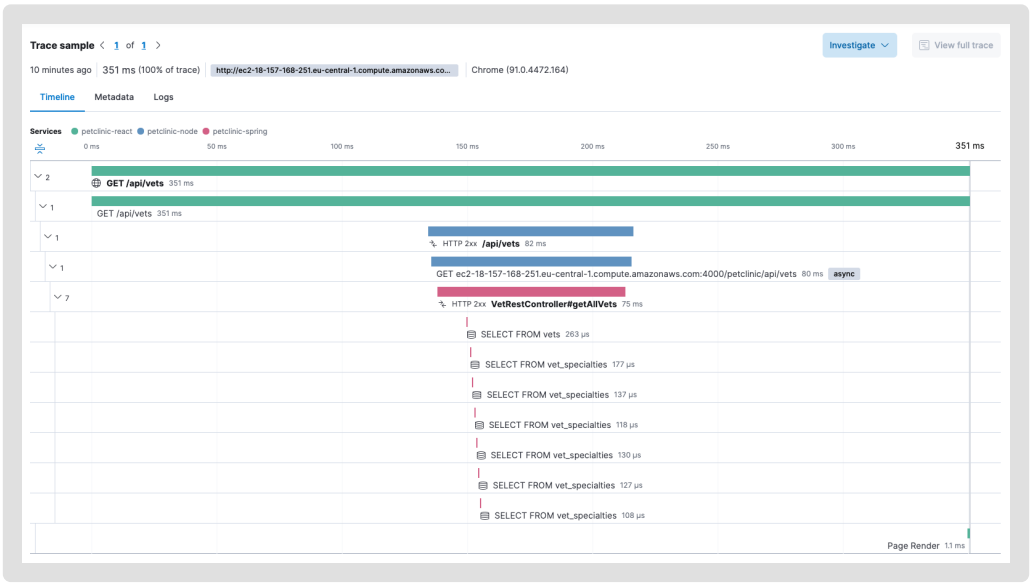


Next, click on the **Transactions** overview and select the

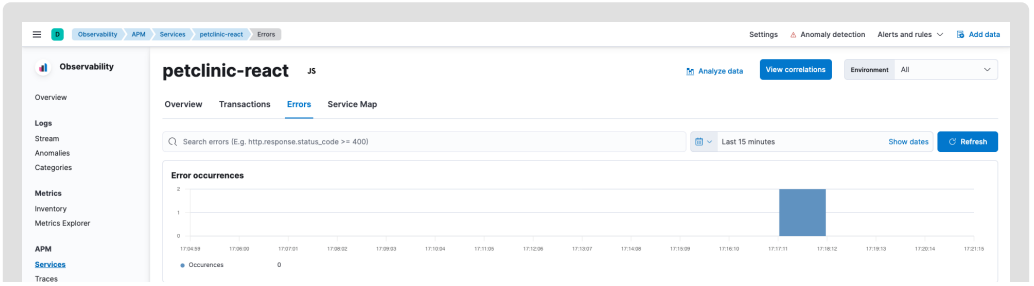
`http-request` transaction type to explore the frontend requests.

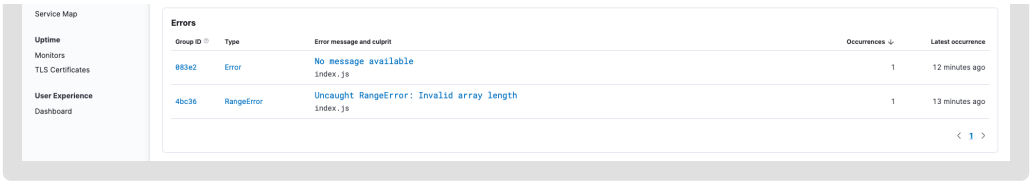


Then, click on `GET /api/vets` to explore distributed tracing and see how the applications and services interact with each other.

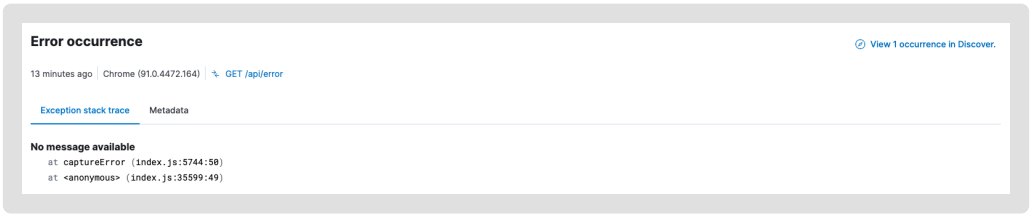


Next, click on the **Errors** overview to explore the application errors.

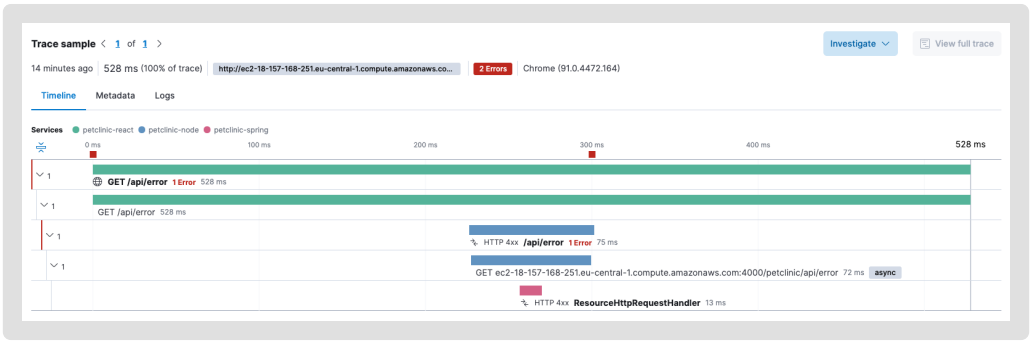




Then, click on the `No message available` error to see its stack trace that points to where the error originated.



Finally, click on the transaction `GET /api/error` to see the related errors and how it was propagated among the services.



Summary: In this lab, you learned how easy it is to setup APM agents for collecting trace information and errors to index them into Elasticsearch. You also explored the Kibana APM app and saw how easily you can monitor application performance.

End of Lab 4