

# 2026

## Monitorización de un despliegue de Apollo Server

### Equipo

Andrés Miguel Pilay Salvatierra

Alberto Soriano Eusebio

Dante Gabriel Masache Silva

Luis Patricio Cuascota Tanicuchi

### ELASTIC STACK (ELK)



Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

## Monitorización de un despliegue de Apollo Server

### Objetivos

En esta actividad aprenderás a monitorizar unas aplicaciones mediante Elastic Stack con unos ejemplos variados, a fin de que conozcáis más a fondo las funcionalidades que ofrece. Para tal fin, utilizaremos una aplicación de ejemplo basada en el tutorial de Apollo Server. Dichas tareas estarán asociado al proyecto transversal de la empresa FinTech Solutions S.A., en el ámbito DevOps, y desarrollado por la empresa de desarrollo software TechOps Solutions, con experiencia en tecnologías y herramientas DevOps y Cloud Computing

### Pautas de elaboración

En esta actividad monitorizaremos un Apollo Server utilizando las diferentes herramientas de Elastic Stack.

- ▶ Criterio 1. Generación de templates de instalación completos (Apollo, STACK ELK y APM). Para comenzar, deberán desplegar el Apollo Server sobre un servidor con Node.js. Pueden utilizar una imagen generada en esta asignatura u otra para que utilice Nginx y Node.js. El despliegue se hará sobre AWS; y, sobre esta instancia, arrancaremos un Apollo Server de ejemplo:  
<https://www.apollographql.com/docs/apollo-server/getting-started/#step-8-execute-your-first-query>. Esta será nuestra aplicación de ejemplo, a la que deberán monitorizar.

---

Para poder capturar estos datos y desplegar un stack de Elastic con Kibana, ElasticSearch, Logstash y APM, pueden usar esta guía en el siguiente enlace:

<https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-elastic-stack.html>

---

Asignatura	Actividad	Fecha
<b>Herramientas DevOps</b>	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

Además, en la misma máquina (u otra) instalarán el servidor de APM siguiendo la guía del siguiente enlace:

<https://www.elastic.co/guide/en/apm/server/7.9/getting-started-apm-server.html>

Una vez configurados los servicios en el servidor, podrán monitorizar la aplicación y generar los dashboards.

- ▶ Criterio 2. Selección de Beats y justificación de estos (teórica). Selección de Beats por usar o justificación de estos indicando qué aporta su uso sobre el ejemplo en el que nos estamos basando.
  
- ▶ Criterio 3. Estrategia de monitorización y creación de Dashboards (teórico). Una vez realizado este paso, llevarán a cabo un análisis de las diferentes posibilidades de monitorización y propondrán un plan de monitorización y visibilidad para el servidor. Se espera que el plan analice al menos tres características del sistema completo. Una de ellas será la monitorización de recursos de CPU y RAM, tanto del servidor con Apollo como el servidor con Elastic Stack. Otra de ellas será el análisis de transacciones y queries al servidor Apollo mediante APM. La tercera queda libre a elección.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

## Criterio 1: Generación de templates de instalación completos (Apollo, STACK ELK y APM).

Para la configuración y despliegue del servicio, se ha decidido por usar Packer para la creación de la AMI que contendrá la aplicación de Apollo en nodeJS con NGINX y el AMI que contendrá los servicios de ElasticSearch, APM Server y el Kibana. Para el despliegue de las maquinas, redes privadas lo realizamos con Terraform, donde desplegamos los servidores asignando correctamente las IPs del APN server al servidor node.

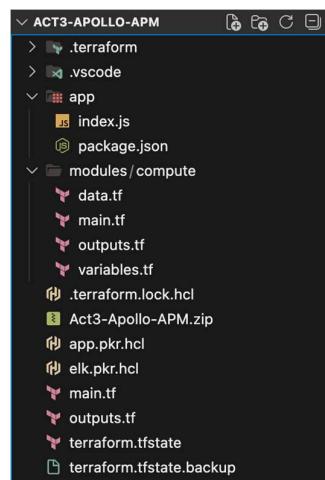


Imagen 1 Módulos Terraform

Comenzamos describiendo el packer para la creación del AMI con Apollo Server, primero tenemos lo que es el index.js donde describimos el código del servidor, dentro de este archivo se encuentran 2 secciones importantes, la configuración del APM node, para poder medir el servidor nodeJS, aquí lo configuramos mediante variables de entorno que tendrán la IP y token para la conexión al APM Server, éstas las colocaremos mediante terraform en el despliegue.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

```
index.js > ...
const apm = require("elastic-apm-node").start({
  serviceName: process.env.APM_SERVICE_NAME || "apollo-fintech-app",
  // URL del servidor APM.
  // Usamos una variable de entorno para inyectarla desde Terraform.
  serverUrl: process.env.APM_SERVER_URL || "http://localhost:8200",
  secretToken: process.env.APM_SECRET_TOKEN || "",
  environment: process.env.NODE_ENV || "production",
});

// Verificación en consola para saber si APM arrancó
if (apm.isStarted()) {
  console.log("✅ Elastic APM Agent iniciado correctamente.");
} else {
  console.log(
    "❌ Elastic APM Agent no se ha iniciado (revisa la configuración.)",
  );
}
```

Imagen 2 Página js

Seguido tenemos la configuración del servidor de apollo donde hemos definido un recurso Book, 10 datos de ejemplo con la estructura de Book, y finalmente los resolvers para obtener los datos y encender el sevidor.

```
const { ApolloServer, gql } = require("apollo-server");

const typeDefs = gql` 
  type Book {
    title: String
    author: String
  }

  type Query {
    books: [Book]
  }
`;

const books = [
  {
    title: "The Awakening",
    author: "Kate Chopin",
  },
  {
    title: "City of Glass",
    author: "Paul Auster",
  },
  {
    title: "To Kill a Mockingbird",
    author: "Harper Lee",
  },
  {
    title: "1984",
    author: "George Orwell",
  },
  {
    title: "Pride and Prejudice",
    author: "Jane Austen",
  },
  {
    title: "The Hobbit",
    author: "J.R.R. Tolkien",
  },
]
```

Imagen 3 Estructura de datos

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

```
// Resolvers
const resolvers = {
  Query: {
    books: () => books,
  },
};

// Inicialización del servidor
const server = new ApolloServer({ typeDefs, resolvers });

const PORT = process.env.PORT || 4000;

server
  .listen({ port: PORT })
  .then(({ url }) => {
    console.log(`⚡ Server ready at ${url}`);
  })
  .catch((err) => {
    console.error("🔴 Error starting server:", err);
  });

```

Imagen 4 Datos del servidor Apollo Server

En el archivo de Packer app.pkr.hcl usaremos una máquina t2.micro con una imagen base de Ubuntu Jammy 22, definimos un tag y proyecto para luego poder obtener el AMI-id desde Terraform.

```
app.pkr.hcl
1 packer {
2   required_plugins {
3     amazon = {
4       version = ">= 1.2.8"
5       source  = "github.com/hashicorp/amazon"
6     }
7   }
8 }
9
10 variable "aws_region" {
11   type    = string
12   default = "us-east-1"
13 }
14
15 source "amazon-ebs" "apollo-app" {
16   ami_name      = "Apollo-App-Image-{{timestamp}}"
17   instance_type = "t2.micro"
18   region        = var.aws_region
19
20   source_ami_filter {
21     filters = [
22       name      = "ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"
23       root-device-type = "ebs"
24       virtualization-type = "hvm"
25     ]
26     most_recent = true
27     owners      = ["099720109477"]
28   }
29
30   ssh_username = "ubuntu"
31
32   # Tags para identificar la AMI creada, incluyendo el Role que Terraform buscará al desplegar
33   tags = {
34     Name      = "Apollo-App-Image"
35     Project   = "FinTech-DevOps-Act3"
36     Role      = "app-server"      # Terraform buscará este tag
37     Environment = "Production"
38   }
39 }
```

Imagen 5 Fichero packer

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

Para el build cargamos los archivos del servidor y definimos la consola no interactiva.

```

build {
    name      = "build-app"
    sources  = ["source.amazon-ebs.apollo-app"]

    # 1. Subimos tu código al servidor temporal
    provisioner "file" {
        source      = "./app"
        destination = "/home/ubuntu/app"
    }

    # 2. Configuración del sistema (Shell Script)
    provisioner "shell" {
        environment_vars = [
            "DEBIAN_FRONTEND=noninteractive"
        ]
    }
}

```

Imagen 6 Builder y provisioners

Usamos script he bash aquí desplegamos el servidor con Node.js y PM2 para mantenerlo siempre activo, además aquí es donde agregamos el MetricBeat para poder parametrizar la máquina virtual. Finalmente configuramos el Nginx para servir el servidor que está en PM2 en el puerto 4000 al puerto 80 mediante proxy inverso.

```

inline = [
    "cloud-init status --wait",
    "sleep 10",

    # Instalación de Node.js y PM2
    "curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -",
    "sudo apt-get update && sudo apt-get install -y nodejs",
    "sudo npm install -g pm2",

    # Instalación de Nginx
    "sudo apt-get install -y nginx",

    # Instalación de Metricbeat
    "curl -fsSL https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /usr/share/keyrings/elastic-archive-keyring.gpg",
    "echo 'deb [signed-by=/usr/share/keyrings/elastic-archive-keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main' | sudo tee /etc/apt/sources.list.d/elastic-8.x.list",
    "sudo apt-get update && sudo apt-get install -y metricbeat",

    # Configurar módulo system en Metricbeat
    "sudo metricbeat modules enable system",
    "sudo systemctl enable metricbeat",

    # Instalar dependencias de la aplicación
    "cd /home/ubuntu/app && npm install",

    # Inicializar PM2 para la aplicación como usuario ubuntu
    "cd /home/ubuntu/app && sudo -u ubuntu pm2 start index.js --name apollo-app",
    "sudo -u ubuntu pm2 save",
    "sudo env PATH=\$PATH:/usr/bin pm2 startup systemd -u ubuntu --hp /home/ubuntu",

    # Configurar Nginx como proxy inverso
    "sudo rm /etc/nginx/sites-enabled/default",
    "echo 'server { listen 80; location / { proxy_pass http://localhost:4000; proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade; proxy_set_header Connection \'upgrade\'; } }' | sudo tee /etc/nginx/sites-available/apollo",
    "sudo ln -s /etc/nginx/sites-available/apollo /etc/nginx/sites-enabled/",
    "sudo systemctl restart nginx"
]

```

Imagen 7 Shell de configuración

Ahora para la creación del AMI de elastic tenemos el archivo elk.pkr.hcl, en este de igual forma definimos ubuntu jammy 22, y los tags para encontrar el AMI-id con terraform, pero a este le asignamos una máquina t3.medium por el peso de elastik.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

```

packer {
  required_plugins {
    amazon = {
      version = ">= 1.2.8"
      source  = "github.com/hashicorp/amazon"
    }
  }

  variable "aws_region" {
    type  = string
    default = "us-east-1"
  }

  source "amazon-ebs" "elk-stack" {
    ami_name      = "ELK-Stack-Image-{{timestamp}}"
    instance_type = "t3.medium"
    region        = var.aws_region

    source_ami_filter {
      filters = [
        name          = "ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"
        root_device-type = "ebs"
        virtualization-type = "hvm"
      ]
      most_recent = true
      owners       = ["699720109477"]
    }

    ssh_username = "ubuntu"

    # Identificadores para búsqueda de AMI por Terraform
    tags = {
      Name      = "ELK-Stack-Image"
      Project   = "FinTech-DevOps-Act3"
      Role      = "monitor-server"
    }
  }
}

```

Imagen 8 Source para generación de AMI.

En el build de este AMI, comenzamos actualizando los repositorios, creando y asignando el par de llaves para Elastic, descargamos Elasticsearch, y corremos el servicio.

```

build {
  name  = "build-elk"
  sources = ["source.amazon-ebs.elk-stack"]

  provisioner "shell" {
    environment_vars = [
      "DEBIAN_FRONTEND=noninteractive"
    ]

    inline = [
      "cloud-init status --wait",
      "sleep 10",

      # Instalación de dependencias y repositorios
      "#sudo apt-get update",
      "#sudo apt-get install -y openjdk-11-jdk apt-transport-https gnupg2",

      # Agregar Llave de Elastic
      "#wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg",
      "#echo 'deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main' | sudo tee /etc/apt/sources.list.d/elasticsearch-8.x.list",
      "#sudo apt-get update",

      # Instalación de Elasticsearch
      "#sudo apt-get install -y elasticsearch",

      # Crear directorios y asignar permisos
      "#sudo mkdir -p /usr/share/elasticsearch/logs",
      "#sudo mkdir -p /usr/share/elasticsearch/data",
      "#sudo chown -R elastic:elastic /usr/share/elasticsearch /usr/share/elasticsearch",
      "#sudo chmod 755 /usr/share/elasticsearch",

      # Configuración de Elasticsearch
      "#sudo rm /etc/elasticsearch/elasticsearch.yml",
      "#echo 'cluster.name: demo-elk' | sudo tee -a /etc/elasticsearch/elasticsearch.yml",
      "#echo 'network.host: 0.0.0.0' | sudo tee -a /etc/elasticsearch/elasticsearch.yml",
      "#echo 'discovery.type: single-node' | sudo tee -a /etc/elasticsearch/elasticsearch.yml",
      "#echo 'xpack.security.enabled: false' | sudo tee -a /etc/elasticsearch/elasticsearch.yml",
      "#echo 'xpack.security.enrollment.enabled: false' | sudo tee -a /etc/elasticsearch/elasticsearch.yml",
      "#echo 'xpack.security.http.ssl.enabled: false' | sudo tee -a /etc/elasticsearch/elasticsearch.yml",
      "#echo 'xpack.security.transport.ssl.enabled: false' | sudo tee -a /etc/elasticsearch/elasticsearch.yml",
      "#sudo systemctl restart elasticsearch",
      "#sleep 5",
    ]
  }
}

```

Imagen 9 Apovisionamiento de ELK

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

Luego de esto descargamos, y configuramos los servicios de Kibana y APM server para que apunten al Elasticserach local, iniciamos los servicios.

```
# Instalación de Kibana
"sudo apt-get install -y kibana",

# Configuración de Kibana
"sudo rm /etc/kibana/kibana.yml",
"echo 'server.port: 5601' | sudo tee /etc/kibana/kibana.yml",
"echo 'server.host: \"0.0.0.0\"' | sudo tee -a /etc/kibana/kibana.yml",
"echo 'elasticsearch.hosts: [\"http://localhost:9200\"]' | sudo tee -a /etc/kibana/kibana.yml",
"sudo chown kibana:kibana /etc/kibana/kibana.yml",
"sudo systemctl enable kibana",

# Instalación de APM Server
"sudo apt-get install -y apm-server",

# Configuración de APM Server - forzar escucha en todas las interfaces
"sudo sed -i 's/apm-server.host:.*$/apm-server.host: \"0.0.0.0:8200\"/' /etc/apm-server/apm-server.yml",
"sudo sed -i 's/host: \"localhost:8200\"/host: \"0.0.0.0:8200\"/' /etc/apm-server/apm-server.yml",
"sudo sed -i 's/host: \"127.0.0.1:8200\"/host: \"0.0.0.0:8200\"/' /etc/apm-server/apm-server.yml",
"echo 'output.elasticsearch.hosts: [\"localhost:9200\"]' | sudo tee -a /etc/apm-server/apm-server.yml",
"sudo systemctl enable apm-server",
"sudo systemctl restart apm-server",
"sleep 3"
}
```

### *Imagen 10 Aprovisionamiento de Kibana y APM*

## Build del AMI del servidor de Apollo

```
[*] /Terrafom/act-Apollo-AW          0.26ms  0:10 (02/11)  B82
❯ packer validate app-phr.hcl
[+]
[*] /Terrafom/act-Apollo-AW          0.15s  0:10 (02/11)  B82
❯ packer inspect app-phr.hcl
Packer Inspect: Multi mode

> input-variables:
var.aws_region = "us-east-1"

> local-variables:
->

> builds:
->

> build-app:
sources:
  amazon-eks.apollo-app
provisioners:
  file
  shell
post-processors:
  <post-processor>
->

[*] /Terrafom/act-Apollo-AW          0.55ms  0:10 (02/11)  B1B
❯ packer build app-phr.hcl
[+]
build-app.amazon-eks.apollo-app: output will be in this color.

=> build-app.amazon-eks.apollo-app: Prevalidating any provided VPC information
=> build-app.amazon-eks.apollo-app: Found existing VPC with ID: vpc-0770022663
=> build-app.amazon-eks.apollo-app: Found IAM Role with ID: arn:aws:iam::089398073c9d:role/packer_0989c09-11cf-4ad6-81eb-b73c9d
=> build-app.amazon-eks.apollo-app: Creating temporary keystore: packer_0989c09-11cf-4ad6-81eb-b73c9d
=> build-app.amazon-eks.apollo-app: Creating temporary security group for this instance: packer_0989c09-11cf-4ad6-81eb-b73c9d
=> build-app.amazon-eks.apollo-app: Authorizing access to port 22 from [0.0.0.0/0] in the temporary security groups...
=> build-app.amazon-eks.apollo-app: Launching a source AMI instance...
=> build-app.amazon-eks.apollo-app: Initiating 1 instance(s) in region us-east-1
=> build-app.amazon-eks.apollo-app: Waiting for instance(s) (arn:aws:ec2:us-east-1:089398073c9d:instance/packer_0989c09-11cf-4ad6-81eb-b73c9d) to become ready...
=> build-app.amazon-eks.apollo-app: Using SSH communicator to connect: 54.189.23.252
=> build-app.amazon-eks.apollo-app: Waiting for instance to become available...
=> build-app.amazon-eks.apollo-app: Uploading ./app /home/ubuntu/app
=> build-app.amazon-eks.apollo-app: Provisioning with shell script: /root/.vagrant.d/boxes/ubuntu-18.04/x86_64/vmlinuz
=> build-app.amazon-eks.apollo-app: WantedBy=multi-user.target
=> build-app.amazon-eks.apollo-app: WantedBy=multi-user.target
=> build-app.amazon-eks.apollo-app: Target path
=> build-app.amazon-eks.apollo-app: /etc/systemd/system/multi-ubuntu.service
=> build-app.amazon-eks.apollo-app: Command List
=> build-app.amazon-eks.apollo-app: [ 1 ] /bin/systemctl enable pm2-ubuntu
=> build-app.amazon-eks.apollo-app: [ 2 ] /bin/systemctl start pm2-ubuntu
=> build-app.amazon-eks.apollo-app: [ 3 ] /bin/systemctl stop pm2-ubuntu
=> build-app.amazon-eks.apollo-app: [ 4 ] /bin/systemctl disable pm2-ubuntu
=> build-app.amazon-eks.apollo-app: [ 5 ] /bin/systemctl daemon-reload
=> build-app.amazon-eks.apollo-app: [ 6 ] /bin/systemctl start pm2-ubuntu
=> build-app.amazon-eks.apollo-app: [ 7 ] /bin/systemctl enable pm2-ubuntu
=> build-app.amazon-eks.apollo-app: Creating system /etc/systemd/system/multi-ubuntu.target.wants/pm2-ubuntu.service
=> /etc/systemd/system/multi-ubuntu.service
=> build-app.amazon-eks.apollo-app: [ 002 ] [v] Command successfully executed.
=> build-app.amazon-eks.apollo-app: [ 003 ] [v] Freezing a process list on reboot via: /etc/systemd/system/multi-ubuntu.service
=> build-app.amazon-eks.apollo-app: [ 004 ] [v] pm2 save
=> build-app.amazon-eks.apollo-app: [ 005 ] [v] Remove init script via: /etc/systemd/system/multi-ubuntu.service
=> build-app.amazon-eks.apollo-app: [ 006 ] [v] server { listen 80; location / { proxy_pass http://localhost:8000; proxy_http_version 1.1; proxy_set_header Upgrade $http_upgrade; proxy_set_header Connection "upgrade"; } }
=> build-app.amazon-eks.apollo-app: [ 007 ] [v] Stopping pm2-ubuntu instance...
=> build-app.amazon-eks.apollo-app: [ 008 ] [v] Stopping instance
=> build-app.amazon-eks.apollo-app: [ 009 ] [v] Starting the command to stop...
=> build-app.amazon-eks.apollo-app: [ 010 ] [v] Calling /bin/systemctl stop pm2-ubuntu@770022663 from instance i-0eb60908d0ab7f2cdff
=> build-app.amazon-eks.apollo-app: [ 011 ] [v] Attaching run tags to AMI...
=> build-app.amazon-eks.apollo-app: [ 012 ] [v] Adding tag: "Name": "Apollo"
=> build-app.amazon-eks.apollo-app: [ 013 ] [v] Adding tag: "Name": "Test Ready"
=> build-app.amazon-eks.apollo-app: [ 014 ] [v] Skipping AMI depreciation...
=> build-app.amazon-eks.apollo-app: [ 015 ] [v] Adding tag: "Name": "Apollo Test Protection"
=> build-app.amazon-eks.apollo-app: [ 016 ] [v] Adding tag: "Name": "Apollo Test Protection"
=> build-app.amazon-eks.apollo-app: [ 017 ] [v] Tagging snapshot: snap-0989398073c9d
=> build-app.amazon-eks.apollo-app: [ 018 ] [v] Adding tag: "Role": "app-server"
=> build-app.amazon-eks.apollo-app: [ 019 ] [v] Adding tag: "Environment": "Production"
=> build-app.amazon-eks.apollo-app: [ 020 ] [v] Adding tag: "Project": "FinTech-DevOps-ACT"
=> build-app.amazon-eks.apollo-app: [ 021 ] [v] Adding tag: "Project": "FinTech-DevOps-ACT"
=> build-app.amazon-eks.apollo-app: [ 022 ] [v] Creating snapshot tags
=> build-app.amazon-eks.apollo-app: [ 023 ] [v] Cleaning up any extra instances...
=> build-app.amazon-eks.apollo-app: [ 024 ] [v] No volumes to clean up, skipping
=> build-app.amazon-eks.apollo-app: [ 025 ] [v] Deleting temporary keypair...
=> build-app.amazon-eks.apollo-app: [ 026 ] [v] Finished after 8 minutes 3 seconds
=> Wait completed after 8 minutes 3 seconds
=> Builds finished. The artifacts of successful builds are:
-> build-app.amazon-eks.apollo-app: AMIs were created:
  => us-east-1: amzn-ami-hvm-2019.09.0-v20190904-1494xpyt-0009p/7
```

Imagen 11 Generación de AMI apollo

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

Build del AMI del servidor de elastic.

```

$ ./Terraform/ct3-Apollo-AMI
$ packer validate elk_pkr.hcl
$ terraform/ELK-Stack-Image-AMI
$ packer validate apm-server.hcl
Packer Inspect: HC2 mode
> input-variables:
var.aws_region: "us-east-1"
> local-variables:
> builds:
>   > build-elk:
sources:
amazon-eks-eks-stack
provisioners:
shell
post-processors:
<no post-processors>
$ ./Terraform/ct3-Apollo-AMI
$ packer build elk_pkr.hcl
build-elk.amazon-eks-eks-stack output will be in this color:
$ packer build elk_pkr.hcl
build-elk.amazon-eks-eks-stack: Prevalidating any VPC information
build-elk.amazon-eks-eks-stack: Prevalidating AMI Name: ELK-Stack-Image-1770824651
build-elk.amazon-eks-eks-stack: Prevalidating AMI Name: ELK-Stack-Image-1770824651
build-elk.amazon-eks-eks-stack: Creating temporary keypair: packer_490ca3b0-6272-4696-8d93-6d6f7815e6bd2
build-elk.amazon-eks-eks-stack: Creating temporary security group for this instance: packer_490ca3b0-9631-e00c-023a-a2-40499116
build-elk.amazon-eks-eks-stack: Authorizing access to port 22 from [0.0.0.0/0] in the temporary security groups...
build-elk.amazon-eks-eks-stack: Authorizing access to port 5432 from [0.0.0.0/0] in the temporary security groups...
build-elk.amazon-eks-eks-stack: Instance ID: i-0908d36fe490191c
build-elk.amazon-eks-eks-stack: Waiting for instance (i-0908d36fe490191c) to become ready...
build-elk.amazon-eks-eks-stack: Waiting for instance (i-0908d36fe490191c) to reach state: running
build-elk.amazon-eks-eks-stack: Waiting for SSH to become available...
build-elk.amazon-eks-eks-stack: Waiting for instance (i-0908d36fe490191c) to reach state: running
build-elk.amazon-eks-eks-stack: Provisioning with shell script: /var/folders/rw/880byym08jn1cuhd10vuptd8000gq/t/p
acter-shell|116928325
build-elk.amazon-eks-eks-stack: status: done
build-elk.amazon-eks-eks-stack: http://i-0908d36fe490191c.123.233.179
build-elk.amazon-eks-eks-stack: Getting: http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
build-elk.amazon-eks-eks-stack: Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
build-elk.amazon-eks-eks-stack: Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [128 kB]
build-elk.amazon-eks-eks-stack: Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease [128 kB]
Wait completed after 15 minutes 33 seconds
Builds finished. The artifacts of successful builds are:
build-elk.amazon-eks-eks-stack: AMIs were created:
us-east-1: ami-041943b08d4cf2e2b

```

Imagen 12 AMI de elastic

AMIs creadas en AWS

Amazon Machine Images (AMIs) (2) <a href="#">Info</a>					
<a href="#">Owned by me</a>		<a href="#">Find AMI by attribute or tag</a>			
<input type="checkbox"/>	Name	AMI name	AMI ID	Source	Owner
<input type="checkbox"/>	ELK-Stack-Image	ELK-Stack-Image-1770824451	ami-041943b08d4cf2e2b	058264523953/ELK-Stack-Image-1770...	058264523953
<input type="checkbox"/>	Apollo-App-Image	Apollo-App-Image-1770822663	ami-0aa5b40705650c72c	058264523953/Apollo-App-Image-177...	058264523953

Imagen 13 Verificación de AMIs creadas

Ahora para Terraform, definimos un archivo main.tf y outputs.tf, para manejar el despliegue, además definimos un módulo “compute”, en el cual se realizará el despliegue de las 2 máquinas.

Dentro del módulo “compute”, definimos variables.tf, donde le diremos a la aplicación el id de la subnet, el id del grupo de seguridad, los tags de proyecto, y aplicación para buscar el AMI-id de cada uno y la clave privada que tendrá el elastic.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

```

variable "subnet_id" {
  description = "ID de la Subnet donde se desplegarán las instancias"
  type        = string
}

variable "security_group_id" {
  description = "ID del Security Group para las instancias"
  type        = string
}

variable "project_tag" {
  description = "Tag del proyecto para buscar las AMIs"
  type        = string
  default     = "FinTech-DevOps-Act3"
}

variable "app_role_tag" {
  description = "Tag Role para la AMI de la app"
  type        = string
  default     = "app-server"
}

variable "elk_role_tag" {
  description = "Tag Role para la AMI de ELK"
  type        = string
  default     = "monitor-server"
}

variable "elk_private_ip" {
  description = "IP Privada estática para el servidor ELK"
  type        = string
  default     = "10.0.1.50"
}

```

Imagen 14 Fichero terraform de variables

Tenemos un archivo data.tf donde realizamos la búsqueda de los AMIs según los tags tanto para el AMI de Apollo y de Elastic.

```

# Buscar la última imagen de APP creada con Packer
data "aws_ami" "app_latest" {
  most_recent = true
  owners      = ["self"]

  filter {
    name  = "tag:Project"
    values = [var.project_tag]
  }

  filter {
    name  = "tag:Role"
    values = [var.app_role_tag] # Debe coincidir con tu app.pkr.hcl
  }
}

# Buscar la última imagen de ELK creada con Packer
data "aws_ami" "elk_latest" {
  most_recent = true
  owners      = ["self"]

  filter {
    name  = "tag:Project"
    values = [var.project_tag]
  }

  filter {
    name  = "tag:Role"
    values = [var.elk_role_tag] # Debe coincidir con tu elk.pkr.hcl
  }
}

```

Imagen 15 Filtros de datos

En main.tf del módulo “compute”, creamos primero la instancia de Elastic, para luego pasar la ip privada de este a la configuración de entorno de Apollo. Valida que se inicie primero elasticsearch antes de los servicios de kibana y apm-server.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

```
# --- 1. Instancia ELK (Monitorización) ---
resource "aws_instance" "elk_server" {
  ami           = data.aws_ami.elk_latest.id
  instance_type = "t3.medium"
  subnet_id     = var.subnet_id
  vpc_security_group_ids = [var.security_group_id]
  private_ip    = var.elk_private_ip

  # Asegurar que los servicios arrancan en orden correcto
  user_data = <<-EOF
    #!/bin/bash
    # Esperar a que Elasticsearch esté listo antes de iniciar los demás servicios
    sudo systemctl start elasticsearch

    # Esperar hasta que Elasticsearch responda
    for i in $(seq 1 30); do
      if curl -s http://localhost:9200 > /dev/null 2>&1; then
        break
      fi
      sleep 5
    done

    # Ahora iniciar Kibana y APM Server
    sudo systemctl start kibana
    sudo systemctl start apm-server
  EOF

  tags = {
    Name  = "ELK-Monitor-Server-Act3"
    Project = var.project_tag
  }
}
```

Imagen 16 Recurso de recursos

Desplegamos el Apollo Server con el AMI, agregamos el ip privada de Elastick tanto a MetricBeat como para el APM-node, luego reiniciamos el servidor PM2 para ver los cambios.

```
# --- 2. Instancia APP (Apollo Server) ---
resource "aws_instance" "app_server" {
  ami           = data.aws_ami.app_latest.id
  instance_type = "c2.micro"
  subnet_id     = var.subnet_id
  vpc_security_group_ids = [var.security_group_id]

  depends_on = [aws_instance.elk_server]

  # USER DATA: Script que se ejecuta al arrancar, insertamos la configuración para que la APP apunte al ELK
  user_data = <<-EOF
    # Configurar Metricbeat para apuntar al servidor ELK
    echo "output.elasticsearch.hosts: ['$aws_instance.elk_server.private_ip':9200']" | sudo tee -a /etc/metricbeat/metricbeat.yml
    echo "setup.kibana.host: '$aws_instance.elk_server.private_ip:5601'" | sudo tee -a /etc/metricbeat/metricbeat.yml
    sudo systemctl restart metricbeat

    # Configurar variablos de entorno APM_SERVER_URL
    echo "export APM_SERVER_URL=http://$aws_instance.elk_server.private_ip:8200" | sudo tee -a /home/ubuntu/.bashrc
    echo "APM_SERVER_URL=http://$aws_instance.elk_server.private_ip:8200" | sudo tee -a /etc/environment

    # Reiniciar PM2 como usuario ubuntu para que tome la nueva variable
    sudo -u ubuntu bash -c 'export APM_SERVER_URL=http://$aws_instance.elk_server.private_ip:8200 && cd /home/ubuntu/app && pm2 restart apollo-app --update-env || pm2 start index.js --name apollo-app'
    sudo -u ubuntu pm2 save
  EOF

  tags = {
    Name  = "Apollo-App-Server"
    Project = var.project_tag
  }
}
```

Imagen 17 Recurso para la aplicación

Finalmente, en el archivo outputs.tf del módulo compute, retornamos las ips públicas del app de Apollo como de Elastic, y el url para ingresar a Kibana con su puerto.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

```

output "app_public_ip" {
  value = aws_instance.app_server.public_ip
}

output "elk_public_ip" {
  value = aws_instance.elk_server.public_ip
}

output "kibana_url" {
  value = "http://${aws_instance.elk_server.public_ip}:5601"
}

```

Imagen 18 Fichero de outputs

Ahora en nuestro main.tf principal, definimos el provedor AWS, creamos el VPC, la subnet pública, el Internal Gateway y la tabla de enrutamiento.

```

terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 4.0"
    }
  }

  provider "aws" {
    region = "us-east-1"
  }

  # 1. Crear una VPC
  resource "aws_vpc" "fintech_vpc" {
    cidr_block          = "10.0.0.0/16"
    enable_dns_hostnames = true
    tags = {
      Name = "Act3-FinTech-VPC"
    }
  }

  # 2. Subnet Pública
  resource "aws_subnet" "public_subnet" {
    vpc_id           = aws_vpc.fintech_vpc.id
    cidr_block       = "10.0.1.0/24"
    map_public_ip_on_launch = true # Para que tengan IP accesible desde internet
    availability_zone = "us-east-1a"
    tags = {
      Name = "Act3-FinTech-Public-Subnet"
    }
  }

  # 3. Internet Gateway
  resource "aws_internet_gateway" "igw" {
    vpc_id = aws_vpc.fintech_vpc.id
    tags = {
      Name = "Act3-FinTech-IGW"
    }
  }

  # 4. Tabla de Enrutamiento
  resource "aws_route_table" "public_rt" {
    vpc_id = aws_vpc.fintech_vpc.id

    route {
      cidr_block = "0.0.0.0/0"
      gateway_id = aws_internet_gateway.igw.id
    }

    tags = {
      Name = "Act3-FinTech-Public-RT"
    }
  }

  resource "aws_route_table_association" "public_assoc" {
    subnet_id     = aws_subnet.public_subnet.id
    route_table_id = aws_route_table.public_rt.id
  }
}

```

Imagen 19 Recursos de VPN y subredes

Definimos los grupos de seguridad para Elastik y para Apollo, en estos permitimos los puertos y accesos comunes tanto para Elasticserach, APM server, Kibana, y el puerto 80 para Apollo y SSH.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

```
# 5. Security Group para la App y ELK
resource "aws_security_group" "common_sg" {
  name      = "act3-fintech-sg"
  description = "Security Group for Apollo and ELK"
  vpc_id    = aws_vpc.fintech_vpc.id

  # Puerto 22: SSH
  ingress {
    from_port  = 22
    to_port    = 22
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Puerto 80: Nginx
  ingress {
    from_port  = 80
    to_port    = 80
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Puerto 5601: Kibana (Para los Dashboards)
  ingress {
    from_port  = 5601
    to_port    = 5601
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Puerto 8200: APM Server (Para recibir datos del agente)
ingress {
  from_port  = 8200
  to_port    = 8200
  protocol   = "tcp"
  cidr_blocks = ["10.0.0.0/16"] # Solo tráfico interno de la VPC
}

# Puerto 9200: Elasticsearch
ingress {
  from_port  = 9200
  to_port    = 9200
  protocol   = "tcp"
  cidr_blocks = ["10.0.0.0/16"]
}

# Salida: Permitir todo
egress {
  from_port  = 0
  to_port    = 0
  protocol   = "-1"
  cidr_blocks = ["0.0.0.0/0"]
}
```

Imagen 20 Recursos para ELK

Contodo esto listo llamamos a nuestro módulo “compute” para lanzar las máquinas con los AMIs enviando la configuración.

```
# 6. Módulo para crear las Instancias EC2 (App + ELK)
module "app_stack" {
  source = "./modules/compute"

  subnet_id      = aws_subnet_public_subnet.id
  security_group_id = aws_security_group.common_sg.id

  # IP Privada Fija para el servidor ELK (Elasticsearch + Kibana)
  elk_private_ip = "10.0.1.50"

  # Tag del proyecto (Para buscar las AMIs correctas de Packer)
  project_tag    = "FinTech-DevOps-Act3"

  # Tags Role para buscar las AMIs correctas
  app_role_tag    = "app-server"
  elk_role_tag    = "monitor-server"
}
```

Imagen 21 Módulo para crear EC2

Finalmente tenemos nuestro archivo outputs.tf principal donde mostramos el url de Apollo y el url con el puerto de Kibana.

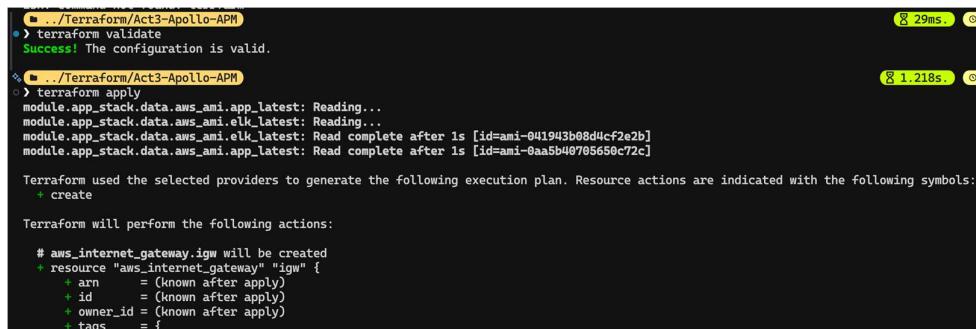
```
output "apollo_app_url" {
  description = "URL pública de tu aplicación Apollo Server"
  value       = "http://${module.app_stack.app_public_ip}"
}

output "kibana_dashboard_url" {
  description = "URL pública para acceder a Kibana"
  value       = module.app_stack.kibana_url
}
```

Imagen 22 Output de dashboard

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

Validamos y ejecutamos Terraform.



```

$ ./Terraform/Act3-Apollo-APM
> terraform validate
Success! The configuration is valid.

$ ./Terraform/Act3-Apollo-APM
> terraform apply
module.app_stack.data.aws_ami.app_latest: Reading...
module.app_stack.data.aws_ami.elk_latest: Reading...
module.app_stack.data.aws_ami.elk_latest: Read complete after 1s [id=ami-041943b88d4cf2e2b]
module.app_stack.data.aws_ami.app_latest: Read complete after 1s [id=ami-0aa5b40705650c72c]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

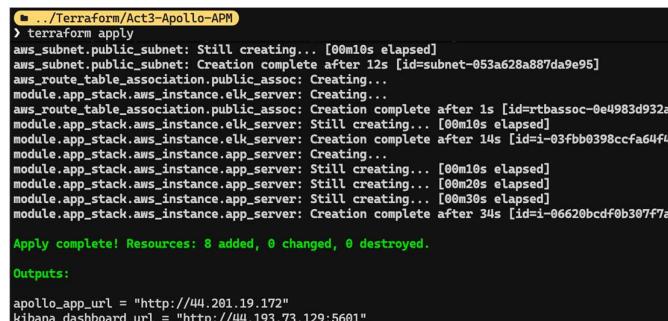
Terraform will perform the following actions:

# aws_internet_gateway.igw will be created
+ resource "aws_internet_gateway" "igw" {
  + arn      = (known after apply)
  + id       = (known after apply)
  + owner_id = (known after apply)
  + tags     =
}

```

Imagen 23 Ejecución de terraform

Servidores desplegados.



```

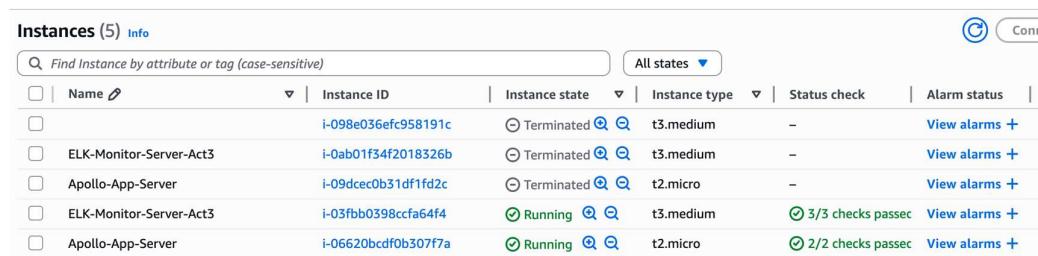
$ ./Terraform/Act3-Apollo-APM
> terraform apply
aws_subnet.public_subnet: Still creating... [00m10s elapsed]
aws_subnet.public_subnet: Creation complete after 12s [id=subnet-053a628a887da9e95]
aws_route_table_association.public_assoc: Creating...
module.app_stack.aws_instance.elk_server: Creating...
aws_route_table_association.public_assoc: Creation complete after 1s [id=rtbassoc-0e4983d932a]
module.app_stack.aws_instance.elk_server: Still creating... [00m10s elapsed]
module.app_stack.aws_instance.elk_server: Creation complete after 14s [id=i-03fbb0398ccfa64f4]
module.app_stack.aws_instance.app_server: Creating...
module.app_stack.aws_instance.app_server: Still creating... [00m10s elapsed]
module.app_stack.aws_instance.app_server: Still creating... [00m20s elapsed]
module.app_stack.aws_instance.app_server: Still creating... [00m30s elapsed]
module.app_stack.aws_instance.app_server: Creation complete after 34s [id=i-06620bcd0b307f7a]

Apply complete! Resources: 8 added, 0 changed, 0 destroyed.

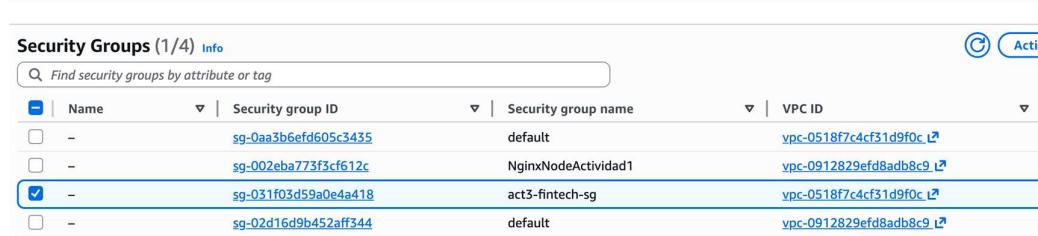
Outputs:
apollo_app_url = "http://44.201.19.172"
kibana_dashboard_url = "http://44.193.73.129:5601"

```

Imagen 24 Apply correcta



Name	Instance ID	Instance state	Instance type	Status check	Alarm status
i-098e036efc958191c	Terminated	t3.medium	-	<a href="#">View alarms</a> +	
ELK-Monitor-Server-Act3	i-0ab01f34f2018326b	Terminated	t3.medium	<a href="#">View alarms</a> +	
Apollo-App-Server	i-09dcec0b31df1fd2c	Terminated	t2.micro	<a href="#">View alarms</a> +	
ELK-Monitor-Server-Act3	i-03fbb0398ccfa64f4	Running	t3.medium	3/3 checks passed <a href="#">View alarms</a> +	
Apollo-App-Server	i-06620bcd0b307f7a	Running	t2.micro	2/2 checks passed <a href="#">View alarms</a> +	

Name	Security group ID	Security group name	VPC ID
-	sg-0aa3b6efd605c3435	default	vpc-0518f7c4cf31d9f0c
-	sg-002eba773f3cf612c	NginxNodeActividad1	vpc-0912829ef8adb8c9
<input checked="" type="checkbox"/>	sg-031f03d59a0e4a418	act3-fintech-sg	vpc-0518f7c4cf31d9f0c
-	sg-02d16d9b452aff344	default	vpc-0912829ef8adb8c9

Imagen 25 Instancia y Security Groups en AWS

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

Verificamos al abrir el output de los servidores que Apollo Server esté ejecutándose.

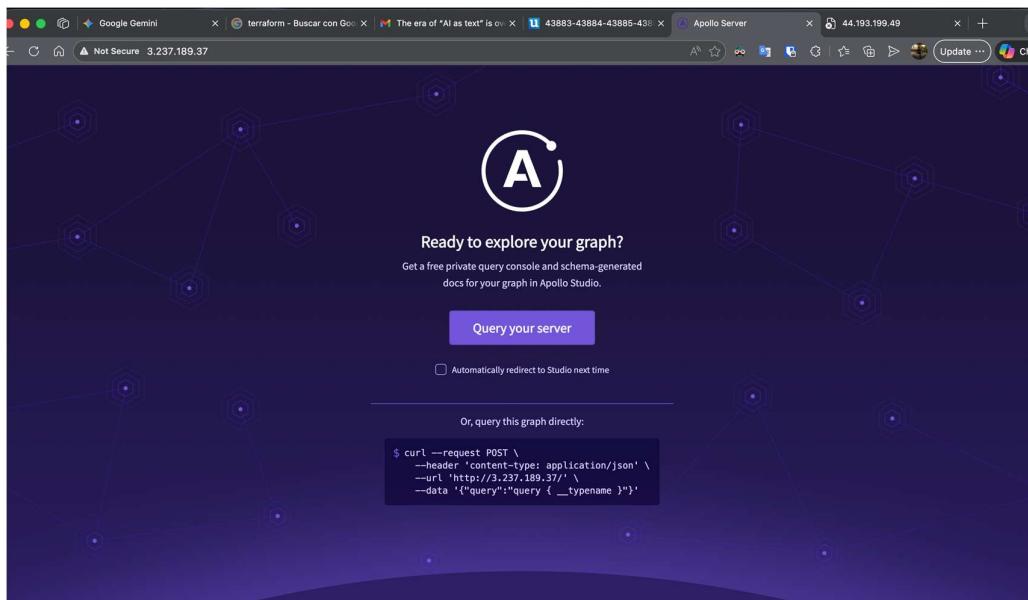


Imagen 26 Verificación de Apollo Server

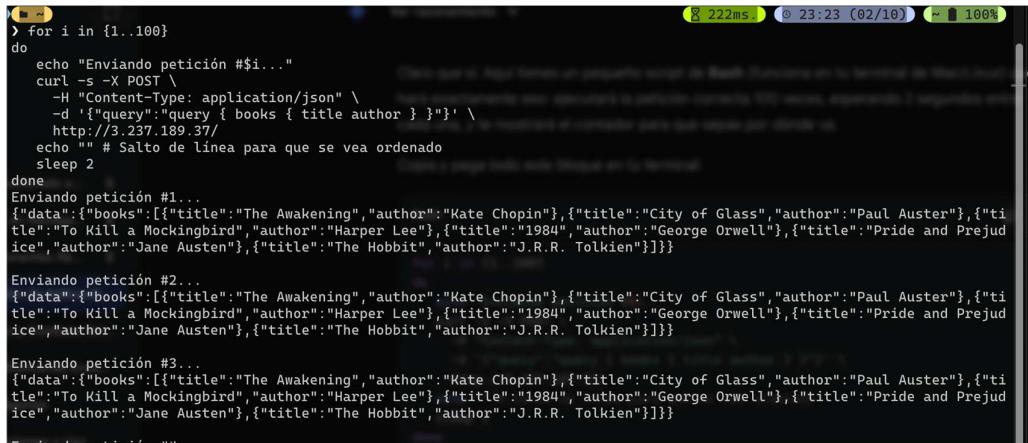
Verificamos desde la terminal si podemos hacer una query de graphql al servidor, así verificamos que el servidor ahora nos devuelve los datos de Moq de los libros que almacenamos quemados en el servidor.

```
$ curl --request POST \
--header 'content-type: application/json' \
--url 'http://3.237.189.37/' \
--data '{"query":"query { books { title author } }"}'
```

Imagen 27 Verificación en terminal

Para generar tráfico al servidor y poder ver métricas dentro de Kibana mediante APN y MetricBeat, corremos un script que envíe la petición de GET cada 2 segundos, y haga 100 peticiones.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026



```

for i in {1..100}
do
    echo "Enviando petición #$i..."
    curl -s -X POST \
        -H "Content-Type: application/json" \
        -d '{"query": "query { books { title author } }"}' \
        http://3.237.189.37/
    echo "" # Salto de linea para que se vea ordenado
    sleep 2
done
Enviando petición #1...
{"data": {"books": [{"title": "The Awakening", "author": "Kate Chopin"}, {"title": "City of Glass", "author": "Paul Auster"}, {"title": "To Kill a Mockingbird", "author": "Harper Lee"}, {"title": "1984", "author": "George Orwell"}, {"title": "Pride and Prejudice", "author": "Jane Austen"}, {"title": "The Hobbit", "author": "J.R.R. Tolkien"}]}

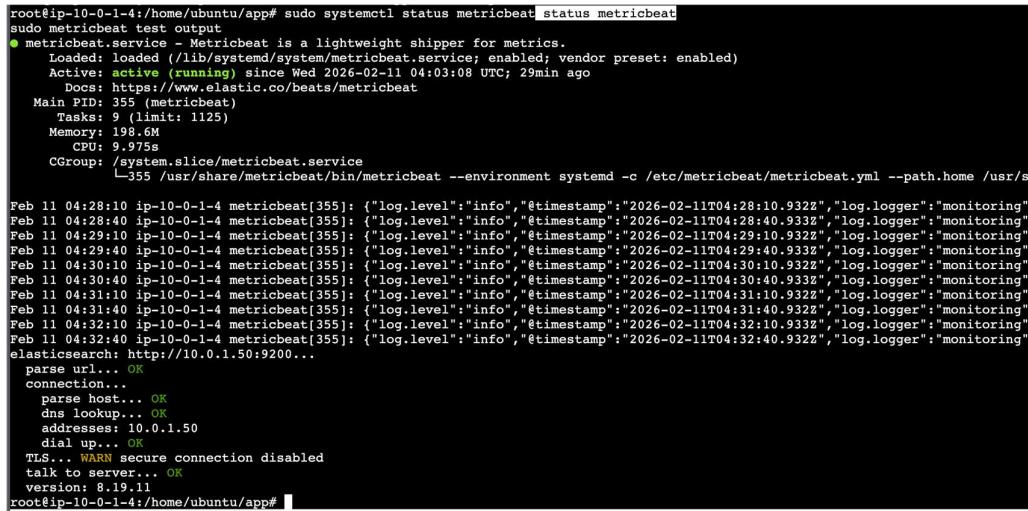
Enviando petición #2...
{"data": {"books": [{"title": "The Awakening", "author": "Kate Chopin"}, {"title": "City of Glass", "author": "Paul Auster"}, {"title": "To Kill a Mockingbird", "author": "Harper Lee"}, {"title": "1984", "author": "George Orwell"}, {"title": "Pride and Prejudice", "author": "Jane Austen"}, {"title": "The Hobbit", "author": "J.R.R. Tolkien"}]}

Enviando petición #3...
{"data": {"books": [{"title": "The Awakening", "author": "Kate Chopin"}, {"title": "City of Glass", "author": "Paul Auster"}, {"title": "To Kill a Mockingbird", "author": "Harper Lee"}, {"title": "1984", "author": "George Orwell"}, {"title": "Pride and Prejudice", "author": "Jane Austen"}, {"title": "The Hobbit", "author": "J.R.R. Tolkien"}]}}

```

Imagen 28 Generación de datos de tráfico

Dentro del acceso SSH de AWS ingresamos a la máquina que contiene el servidor de apolo para validar que el servicio de MetricBeat esté conectado y ejecutándose correctamente.



```

root@ip-10-0-1-4:/home/ubuntu/app# sudo systemctl status metricbeat
● metricbeat.service - Metricbeat is a lightweight shipper for metrics.
   Loaded: loaded (/lib/systemd/system/metricbeat.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2026-02-11 04:03:08 UTC; 29min ago
     Docs: https://www.elastic.co/beats/metricbeat
 Main PID: 355 (metricbeat)
   Tasks: 9 (limit: 1125)
    Memory: 198.6M
      CPU: 9.975s
     CGroup: /system.slice/metricbeat.service
             └─355 /usr/share/metricbeat/bin/metricbeat --environment systemd -c /etc/metricbeat/metricbeat.yml --path.home /usr/sh
Feb 11 04:28:10 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:28:10.932Z", "log.logger": "monitoring",
Feb 11 04:28:40 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:28:40.933Z", "log.logger": "monitoring",
Feb 11 04:29:10 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:29:10.932Z", "log.logger": "monitoring",
Feb 11 04:29:40 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:29:40.933Z", "log.logger": "monitoring",
Feb 11 04:30:10 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:30:10.932Z", "log.logger": "monitoring",
Feb 11 04:30:40 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:30:40.933Z", "log.logger": "monitoring",
Feb 11 04:31:10 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:31:10.932Z", "log.logger": "monitoring",
Feb 11 04:31:40 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:31:40.932Z", "log.logger": "monitoring",
Feb 11 04:32:10 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:32:10.933Z", "log.logger": "monitoring",
Feb 11 04:32:40 ip-10-0-1-4 metricbeat[355]: {"log.level": "info", "@timestamp": "2026-02-11T04:32:40.932Z", "log.logger": "monitoring",
elasticsearch: http://10.0.1.50:9200...
  parses url... OK
  connection...
  parses host... OK
  dns lookup... OK
  addresses: 10.0.1.50
  dial up... OK
TLS... WARN secure connection disabled
talk to server... OK
version: 8.19.11
root@ip-10-0-1-4:/home/ubuntu/app#

```

Imagen 29 ssh de AWS

Ahora ingresamos al servidor que contiene ElasticSearch con Kibana en el puerto 5601 y validamos que correctamente está ejecutándose el servicio.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

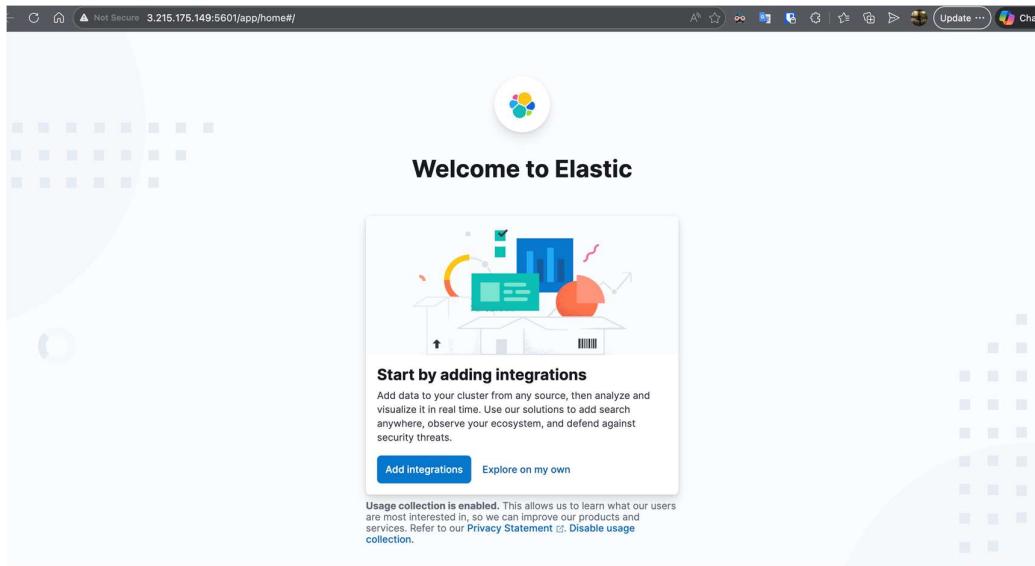


Imagen 30 Ingreso a Elastic

Vistas del home dashboard de Kibana.

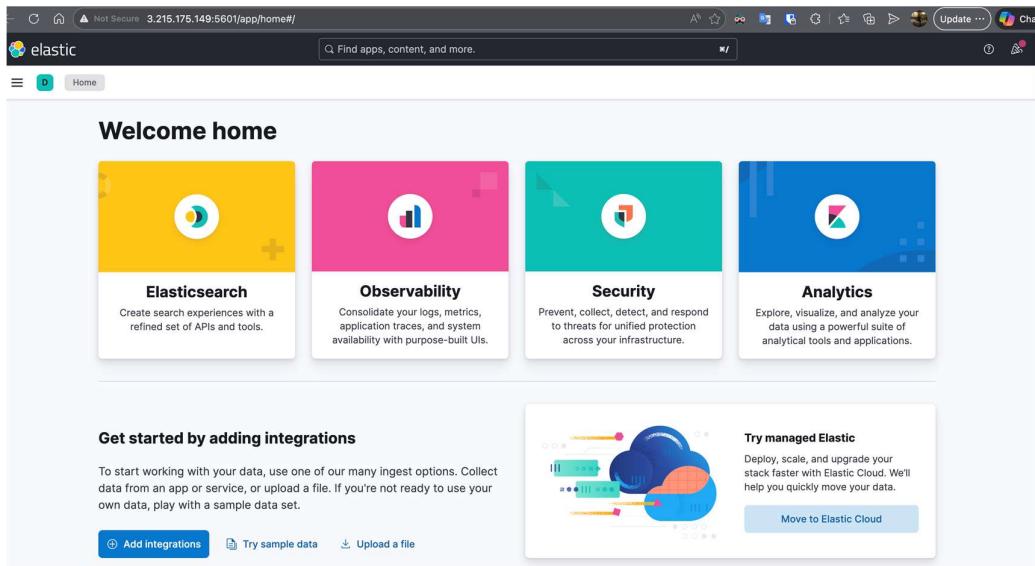


Imagen 31 Ingreso a Kibana

Ahora dentro del entorno ingresamos a la sección de observability, para verificar que el nodo de Apolo esté conectado, y hemos observado que si se encuentra con su identificador con la IP privada virtual con ip: 10.0.1.4 y está realizando las mediciones correctamente.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

The screenshot shows the Observability Overview page. On the left, there's a sidebar with categories like Overview, Alerts, SLOs, Cases, AI Assistant, Logs (with sub-options like Explorer BETA, Logs Anomalies, Logs Categories, Settings), Infrastructure (with sub-options like Infrastructure inventory, Metrics Explorer, Hosts), Applications (with sub-options like Service Inventory, Traces, Dependencies), and Synthetics. The main area has a title 'Overview' and a sub-section 'Collect and analyze logs in observability'. It shows a table for 'Hosts' with one entry: Uptime 15m 12s, Hostname ip-10-0-1-4, CPU % 2.89%, Load 15m 0.00, RX 282B/s, TX 1KB/s. Below that is a section for 'Services' showing 1 service at 0.1 tpm with a throughput chart from February 10, 2026, to February 11, 2026.

Imagen 32 Observabilidad de servidor

## Métricas del servidor.

The screenshot shows the Infrastructure Metrics detail page for host ip-10-0-1-4. The sidebar is identical to the previous screenshot. The main area has a title 'ip-10-0-1-4' and tabs for Overview, Metadata, Metrics, Processes, Logs, Anomalies, and Osquery. Under 'Metrics', there are four cards: CPU Usage (Average 2.9%), Normalized Load (Average 0.4%), Memory Usage (Average 47.6%), and Disk Usage (Max 39.6%). Below these are sections for 'Metadata' (Host IP 10.0.1.4, Host OS version 22.04.5 LTS (Jammy Jellyfish), Cloud provider aws, Operating system Ubuntu), 'Alerts' (No active alerts), 'Services' (We were unable to find services running on this host. Click here to instrument your services with APM. Troubleshooting), and 'Metrics'.

Imagen 33 Métricas del servidor

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

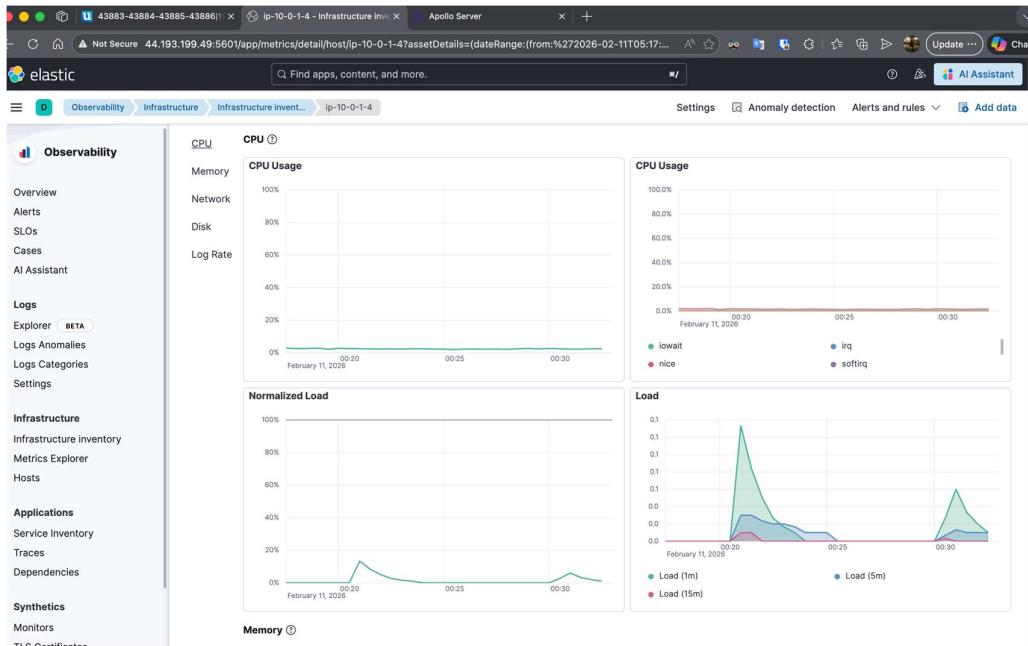


Imagen 34 Observabilidad de CPU

Para poder observar a más detalle, agregamos una vista de metricbeat\* con @timestamp, para que lea todos los servidores conectados, como solo tenemos el Apollo no hay problema. Así podemos ya ver las métricas del MetricBeat ya dentro de Discover.

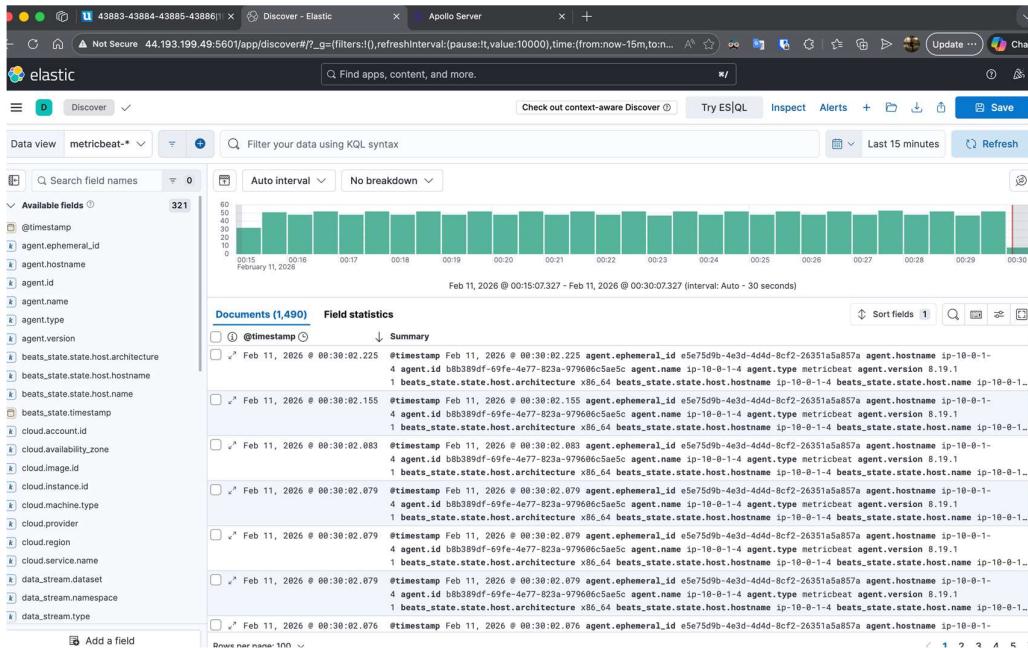


Imagen 35 Logs en Discover

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

Ahora validamos de igual forma en Discover que existe una vista que se creó automáticamente para el APM server, y validamos que estemos recibiendo los logs de las peticiones que se están procesando dentro de nuestro servidor node de Apollo.

Imagen 36 Servicios del servidor

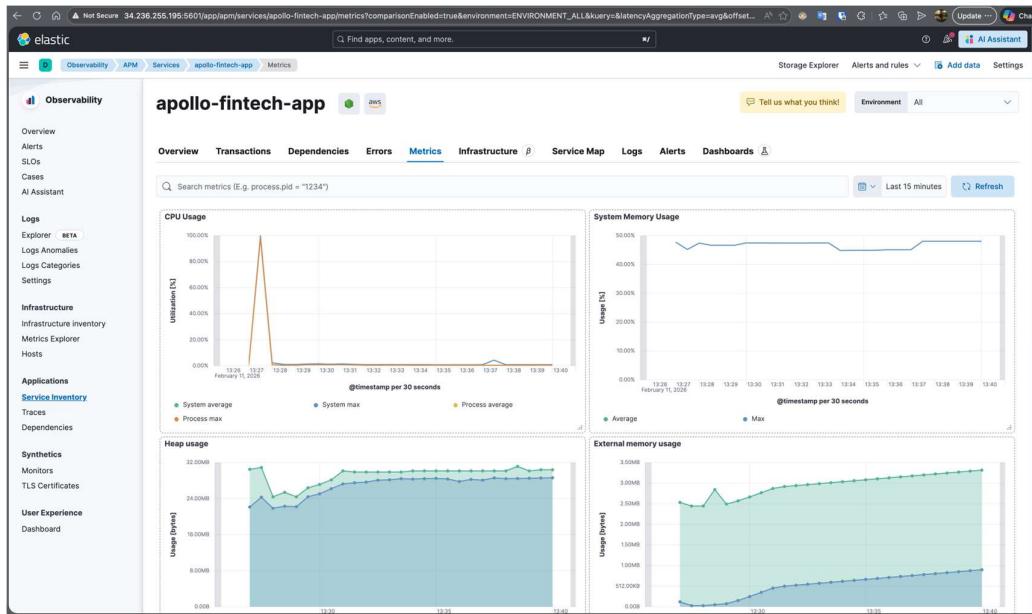
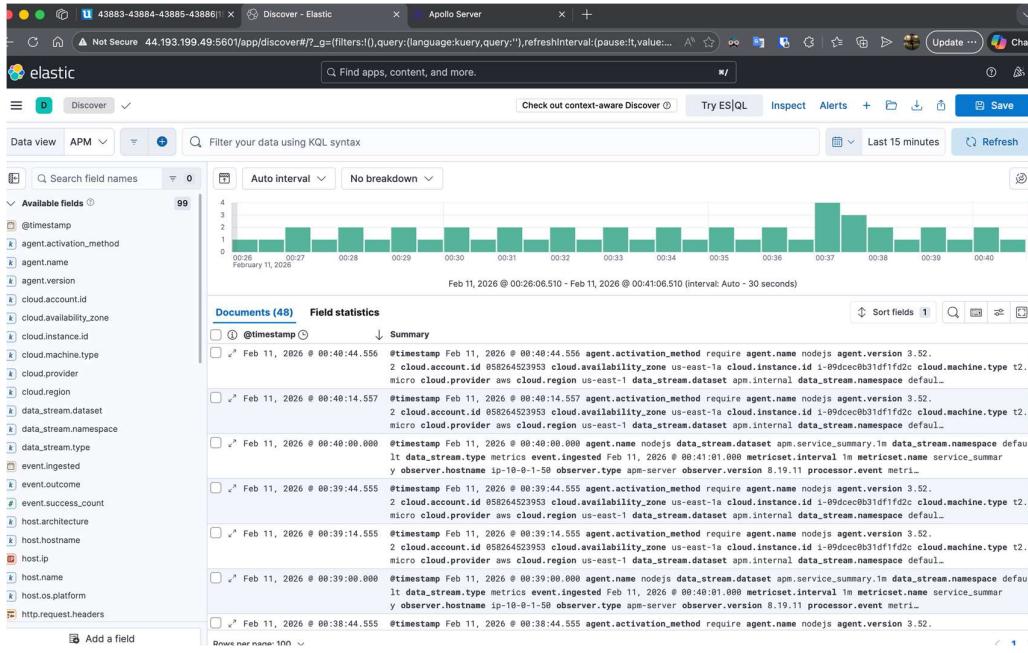
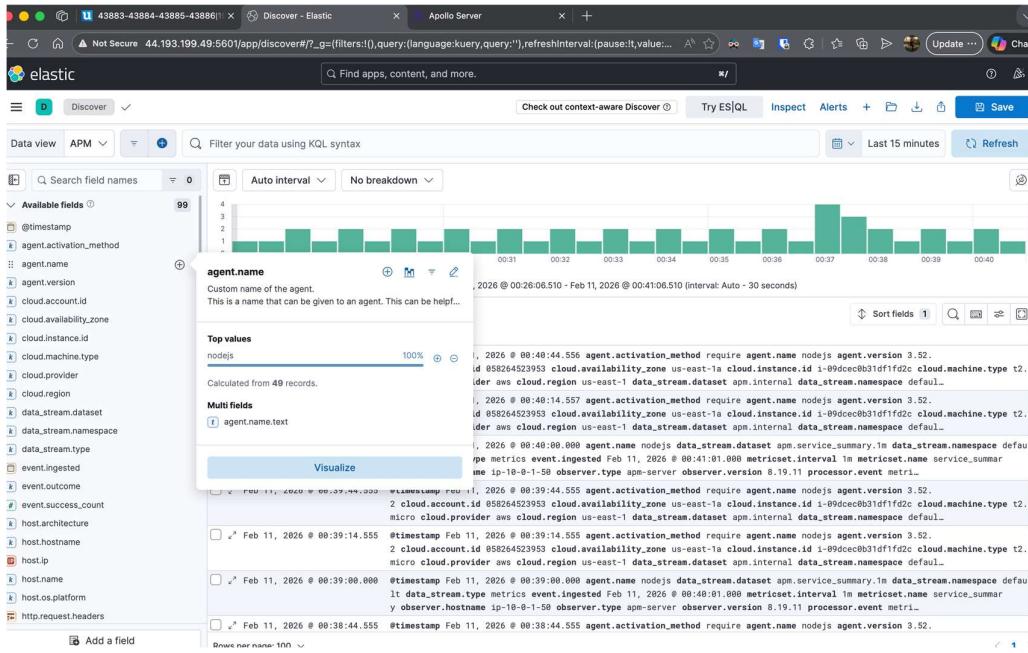


Imagen 37 Métricas del servidor

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026



### *Imagen 38 Logs del servidor*



### Imagen 39 Logs por agente.name

Con esto estaría finalizada la ejecución de la estructura del proyecto Apollo Server / ELK / APM, y continuar con la estructura del dashboard.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

## Criterio 3: Estrategia de monitorización y creación de Dashboards.

Una buena estrategia no busca llenar pantallas de gráficos, sino reducir el **MTTR** (*Mean Time To Repair*).

### 1. Estrategia de Monitorización y Creación de Dashboards

La monitorización moderna se basa en tres pilares: **Métricas, Logs y Traces**. Para que un Dashboard sea efectivo, debe seguir la jerarquía de visibilidad:

- **Dashboards de Nivel Alto (Ejecutivos/SRE)**: Semáforos de salud global (Uptime, latencia promedio).
- **Dashboards Operativos**: El "corazón" del sistema. Se basan en el método **USE** (Utilización, Saturación y Errores) para infraestructura, y el método **RED** (Requests, Errors, Durations) para servicios.
- **Dashboards de Drill-down**: Vistas técnicas detalladas para depuración profunda.

### 2. Análisis y Plan de Monitorización Propuesto

A continuación, detallo el plan técnico para cubrir las tres dimensiones solicitadas, utilizando el Elastic Stack como núcleo.

#### Característica 1: Infraestructura (CPU y RAM)

**Objetivo:** Controlar el consumo de recursos tanto del nodo de aplicación (Apollo) como del nodo de datos (Elastic Stack).

- **Herramienta: Metricbeat** (módulo system).
- **Métricas Clave:**
  - ✓ **Node CPU Usage / Memory Swap**: Vital para detectar *memory leaks* en el proceso de Node.js (Apollo).
  - ✓ **JVM Heap Usage**: Crítico para el servidor de Elastic Stack. Si el Heap llega al 90%, el sistema dejará de indexar.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

- **Aporte:** Permite dimensionar correctamente los servidores (Right-sizing). Si el servidor Apollo consume poco pero Elastic está al límite, sabemos dónde invertir presupuesto de escalado.

### Característica 2: Rendimiento de Aplicación (APM en Apollo)

**Objetivo:** Visibilidad interna de las queries GraphQL y transacciones.

- **Herramienta:** **Elastic APM Agent** (integrado en el código de Apollo Server).
- **Análisis de Transacciones:**
  - ✓ **Trace Spans:** Ver cuánto tarda exactamente cada *resolver* de GraphQL en consultar la base de datos o servicios externos.
  - ✓ **Error Tracking:** Captura automática de excepciones no controladas en Apollo sin que el usuario tenga que reportarlas.
- **Aporte:** Identifica "N+1 queries" (un problema común en GraphQL) y cuellos de botella en la lógica de negocio antes de que afecten a todos los usuarios.

### Característica 3: Monitorización de Seguridad y Accesos (HTTP Audit)

**Objetivo:** Auditoría de quién accede al sistema y desde dónde.

- **Herramienta:** **Filebeat** (módulo nginx o apache si hay un proxy, o parsing de logs de acceso de Apollo).
- **Métricas Clave:**
  - ✓ **Códigos de Respuesta (4xx, 5xx):** Un aumento en errores 403/401 indica intentos de intrusión o fallos de permisos.
  - ✓ **Geolocalización (GeoIP):** Mapa de calor de las peticiones entrantes.
- **Aporte:** Seguridad proactiva. Detectar picos de tráfico desde regiones geográficas inusuales permite aplicar reglas de firewall (WAF) de forma inmediata.

### 3. Resumen del Plan de Visibilidad

Para el servidor, propongo un **Dashboard Unificado en Kibana** que organice la información de la siguiente manera:

Nivel	Componente	Visualización sugerida
Infraestructura	CPU/RAM (Beats)	Gráfico de áreas apiladas (para ver saturación total).

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026
Aplicación	APM (Apollo)	Gráfico de líneas de percentiles (p95, p99) para latencia de transacciones.
Seguridad	Logs de Acceso	Mapa de coordenadas y contador de errores 5xx en tiempo real.

### Definición de los umbrales de alerta (Thresholds) para estas métricas

Definir umbrales (thresholds) y canales de notificación es lo que transforma una "gráfica bonita" en un sistema de respuesta ante incidentes (Incident Response). El Site Reliability Engineering, se propone una estrategia de alertas basada en severidad, para evitar la "fatiga de alertas" (recibir tantos avisos que terminas ignorándolos).

#### 1. Estrategia de Alertas por Capas

Dividiremos las notificaciones en dos canales:

- **Slack/Discord/Teams:** Para alertas de advertencia (Warning) y críticas.
- **Email/PagerDuty:** Solo para estados de "Sistema Caído" (Critical).

#### 2. Definición de Umbrales (Thresholds) Técnicos

A continuación, los valores recomendados para las tres características de nuestro plan:

##### A. Infraestructura (Metricbeat)

- **CPU Usage (Apollo o Elastic):**
  - ✓ **Warning (80% durante 5 min):** Indica que estamos cerca del límite. Posible necesidad de escalado horizontal.
  - ✓ **Critical (95% durante 2 min):** Riesgo inminente de *kernel panic* o congelamiento.
- **RAM Usage (JVM Heap en Elastic):**
  - ✓ **Critical (85%):** Elasticsearch empieza a tener problemas con el Garbage Collector. Si llega al 90%, puede bloquear las escrituras de logs.

##### B. Rendimiento de Aplicación (APM en Apollo)

- **Latencia p95 (Transaction Duration):**
  - ✓ **Warning (> 2 segundos):** El 5% de tus usuarios experimenta lentitud.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

- ✓ **Critical (> 5 segundos):** La experiencia de usuario es inaceptable.
- **Error Rate:**
  - ✓ **Critical (> 5% de transacciones con error):** Indica un bug desplegado o caída de una dependencia (Base de Datos).

### C. Seguridad y Accesos (Filebeat)

- **Errores 4xx (Client Errors):**
  - ✓ **Warning:** Más de 100 errores 404 en 1 minuto ( posible escaneo de vulnerabilidades/directorios).
- **Errores 5xx (Server Errors):**
  - ✓ **Critical:** Cualquier pico súbito de errores 500 (indica que el servidor Apollo está vivo pero incapaz de procesar).

### 3. Implementación Práctica: Ejemplo de Alerta en Kibana (Elastic Threshold Rule)

Para configurar esto en la sección de **Stack Management > Rules and Connectors**, la lógica sería:

1. **Nombre:** [CRITICAL] Apollo Server - High Memory Usage
2. **Condición:** When system.memory.actual.used.pct is ABOVE 0.9 (90%) for the last 5 minutes.
3. **Acción:** Enviar mensaje a Slack Webhook.
4. **Mensaje:** > "⚠ Alerta Crítica: El servidor Apollo está al {{context.value}} de uso de RAM. Revisar posibles Memory Leaks o reiniciar servicio."

### 4. Automatización con Acciones de Recuperación

En un entorno DevOps maduro, no solo notificamos. Podemos configurar que, ante una alerta Critical de RAM en el servidor Apollo, para:

1. Reiniciar el contenedor.
2. Limpiar caché.
3. Escalar una instancia adicional.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

## Dashboards.

Página para la generación de dashboards.

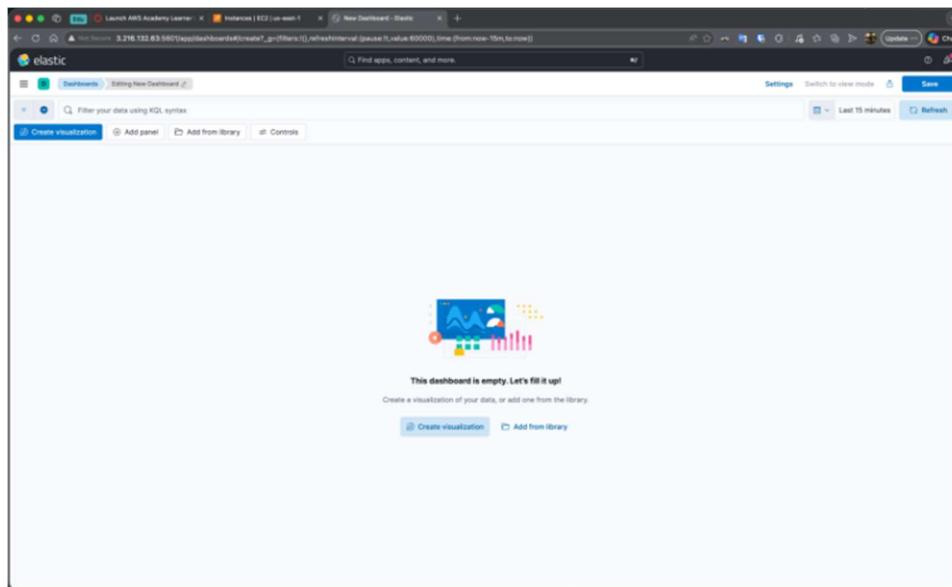


Imagen 40 Generación de dashboard

Apollo Server ejecuta de forma síncrona o asíncrona según como se ejecutaron las peticiones. En el siguiente dashboard podemos observar el consumo de la CPU en cierto tiempo.

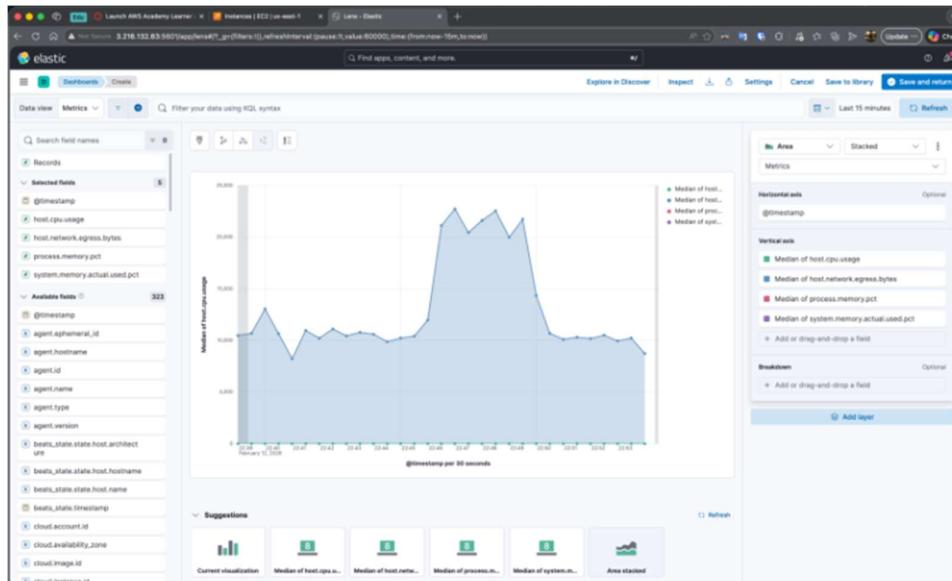


Imagen 41 Dashboard de uso CPU

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

El motivo más importante para tener este dashboard es para monitorear las fugas de memoria en Node.js. El dashboard permite correlacionar un salto de memoria en el momento exacto en que se ejecutó cierto tipo de query.

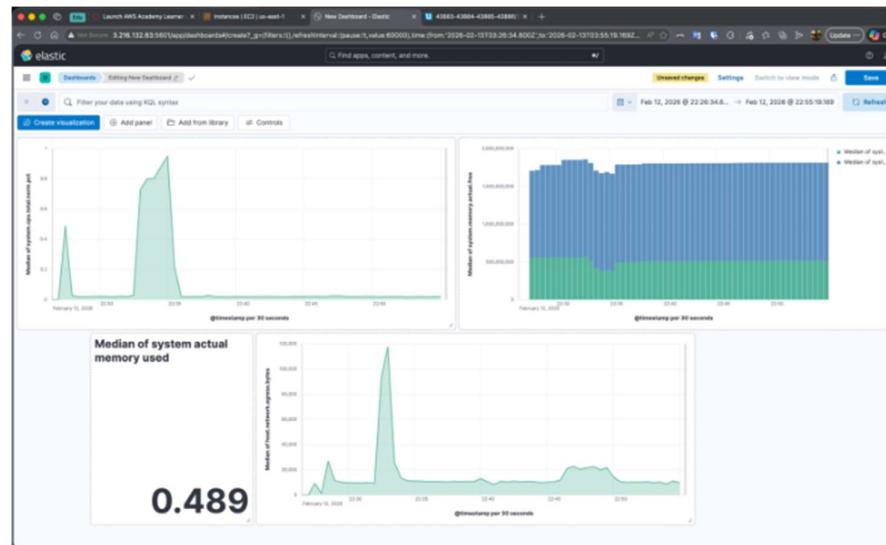


Imagen 42 Dashboard de uso de memoria

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

## Política de Retención

La Política de Retención es el equilibrio entre dos fuerzas: la necesidad de tener datos para investigar incidentes históricos y el coste (o rendimiento) de almacenar terabytes de logs innecesarios.

### 1. Estrategia de Retención: El Modelo "Hot-Warm-Cold"

Para tu servidor con Apollo y Elastic Stack, propone una política basada en la relevancia temporal de los datos.

#### Fases del Ciclo de Vida:

##### 1. Fase Hot (Caliente):

- a. **Contenido:** Logs de errores de Apollo, trazas de APM y métricas de CPU/RAM de hoy.
- b. **Almacenamiento:** Discos SSD rápidos.
- c. **Retención:** 7 días.
- d. **Acción:** Escritura y lectura intensiva para alertas en tiempo real.

##### 2. Fase Warm (Templada):

- a. **Contenido:** Datos de la semana pasada hasta hace un mes.
- b. **Almacenamiento:** Discos más económicos. Los índices se vuelven de "solo lectura" y se optimizan (Force Merge).
- c. **Retención:** 30 días.

##### 3. Fase Cold / Delete (Borrado):

- a. **Acción:** Los datos se eliminan automáticamente o se mueven a un snapshot en la nube (S3/Azure Blob) si hay requisitos legales.

### 2. Definición Técnica de la Política

Aquí tienes la definición teórica de cómo se vería la política para tus logs y métricas. Esta política asegura que el servidor no se quede sin espacio en disco (un error clásico que tumba el nodo de Elastic).

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

### Especificaciones por tipo de dato:

Tipo de dato	Retención sugerida	Justificación
Métricas (Metricbeat)	15 Días	Las métricas de CPU/RAM pierden valor rápidamente. Solo necesitamos el histórico para ver tendencias.
Logs de Error (Filebeat)	90 Días	Necesarios para auditorías de seguridad o para investigar errores que los usuarios reportan tarde.
Trazas APM	7 Días	El APM genera un volumen de datos masivo. 7 días es suficiente para debugear el rendimiento actual.

### 3. Implementación de la Política en JSON

En Elastic Stack, esto se define mediante una API. Aquí tienes la lógica para que el sistema gestione el borrado solo:

```
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": { "max_size": "50gb", "max_age": "30d" }
        }
      },
      "delete": {
        "min_age": "90d",
        "actions": { "delete": {} }
      }
    }
  }
}
```

#### ¿Qué aporta esta política a tu servidor?

- Automatización:** No tienes que preocuparte por scripts de limpieza cron.
- Predicibilidad:** Sabes exactamente cuánto espacio en disco necesitas.
- Rendimiento:** Al mover datos viejos a fases "Warm", liberas la memoria RAM (Heap) del servidor para los datos nuevos y críticos.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

## Criterio 2: Selección de Beats y justificación.

### ¿Qué son los Beats?

En la arquitectura clásica de ELK, el mayor cuello de botella era el consumo de recursos. Logstash, al ser una herramienta basada en la JVM (Java Virtual Machine), es potente pero pesada para ejecutarse en cada servidor o contenedor. Los Beats nacen como la respuesta "lightweight" (ligera). Son agentes transportadores de datos de un solo propósito, escritos en Go. Su filosofía es: "Haz una sola cosa, hazla extremadamente rápido y consume el mínimo de CPU y RAM posible".

### Características Clave:

- ✓ Eficiencia Nativa: Se ejecutan como binarios estáticos sin dependencias externas.
- ✓ Backpressure Sensitivity: Si el destino (Logstash o Elasticsearch) está saturado, el Beat reduce la velocidad de envío para no colapsar la red ni perder datos.
- ✓ Módulos Out-of-the-box: Ya traen configuraciones predefinidas para logs de Nginx, MySQL, auditoría de sistema, etc.

### Selección de Beats y Justificación Teórica

Para un stack DevOps de alto nivel, no se trata de instalarlos todos, sino de elegir los que cubren las dimensiones críticas de la telemetría:

- ▶ Filebeat

Función: Recolecta y centraliza archivos de log.

Justificación: En sistemas distribuidos, los logs son efímeros (especialmente en Docker/K8s). Filebeat garantiza que, aunque un contenedor muera, su rastro persista. Utiliza un puntero (registry) para saber exactamente dónde se quedó leyendo, evitando duplicidad de datos tras un reinicio.

- ▶ Metricbeat

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

Función: Recolecta métricas del sistema y de servicios (CPU, RAM, carga de disco, métricas de Redis, etc.).

Justificación: Mientras Filebeat nos dice *qué pasó*, Metricbeat nos dice *cómo está la salud del hardware/software*. Es el sustituto moderno de agentes pesados de monitoreo, permitiendo correlacionar un pico de CPU con un error específico en los logs.

- ▶ Packetbeat

Función: Sniffer de paquetes que analiza protocolos de red (HTTP, DNS, SQL) en tiempo real sin latencia.

Justificación: Permite entender la latencia entre microservicios. Si una consulta SQL es lenta, Packetbeat la detecta sin que tengas que modificar ni una sola línea de código de tu aplicación.

### Justificación:

Crucial para el cumplimiento (compliance) y DevSecOps. Detecta si alguien modificó un archivo binario sensible o si se ejecutó un proceso sospechoso.

Si comparamos el uso de Beats frente a un esquema donde solo usamos Logstash o agentes genéricos (como el envío por Syslog), los Beats aportan tres pilares fundamentales:

Característica	Con Beats	Sin Beats
Consumo de Recursos	Mínimo. Ideal para arquitecturas de microservicios y Edge Computing.	Alto. Logstash en cada nodo consume demasiada memoria.
Estandarización	Utiliza el Elastic Common Schema (ECS), facilitando la correlación de datos.	Formatos heterogéneos que requieren mucho parsing manual.
Seguridad y Resiliencia	Soporte nativo para TLS y manejo inteligente de caídas de red.	Riesgo de pérdida de datos si el buffer de red se llena.

Al implementar Beats, dejas de ver "datos aislados" y pasas a tener Observabilidad. El gran aporte es la correlación inmediata: puedes navegar en Kibana y ver cómo un

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

despliegue (detectado por Filebeat) afectó la latencia de la base de datos (detectada por Packetbeat) y elevó el consumo de memoria del nodo (detectado por Metricbeat).

Para aterrizar esta teoría en algo tangible, nos enfocamos en **Filebeat** y **Metricbeat**, que son los pilares de cualquier implementación DevOps. Imagina que tenemos un microservicio corriendo en un contenedor. Vamos a ver cómo se configuraría la recolección de datos y qué aporta exactamente cada sección.

### 1. Filebeat: Recolección Inteligente de Logs

En lugar de simplemente "enviar texto", Filebeat estructura la información.

#### Ejemplo de configuración (filebeat.yml):

filebeat.inputs:

- type: container

paths:

- /var/log/containers/\*.log

processors:

- add\_kubernetes\_metadata:
  - host: \${NODE\_NAME}

output.elasticsearch:

hosts: ["elasticsearch:9200"]

- **¿Qué aporta aquí? \* Metadatos:** El procesador add\_kubernetes\_metadata no solo envía el log; le añade el nombre del Pod, el Namespace y los Labels.
  - **Justificación:** En un entorno dinámico, un log que dice "Error 500" no sirve de nada si no sabes a qué versión del microservicio o a qué nodo pertenecía. Filebeat resuelve esto en el origen.

### 2. Metricbeat: Visibilidad de Recursos

Metricbeat utiliza "módulos" para entender de qué servicio está hablando.

#### Ejemplo de configuración (metricbeat.yml):

metricbeat.modules:

- module: docker

metricsets:

- "container"

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

```

- "cpu"
- "memory"
- "network"
hosts: ["unix:///var/run/docker.sock"]
period: 10s

```

```

output.elasticsearch:
hosts: ["elasticsearch:9200"]

```

- **¿Qué aporta aquí?**

- **Frecuencia y Granularidad:** Al definir un period: 10s, obtienes una resolución temporal alta para detectar "picos" que herramientas tradicionales de monitoreo (que promedian cada 5 minutos) ignorarían.
- **Justificación:** Te permite correlacionar: si el log de Filebeat muestra un error de *Timeout*, Metricbeat te confirmará si en ese mismo segundo el contenedor llegó al límite de su CPU (*Throttling*).

### 3. Recomendación

La arquitectura recomendada que justifica el uso de Beats en un entorno de producción es la siguiente:

1. **Beats** (En el host/contenedor): Recolecta y filtra mínimamente.
2. **Logstash** (Capa intermedia): Recibe de los Beats, enriquece los datos (Geolocalización, parseo complejo de mensajes) y los distribuye.
3. **Elasticsearch**: Almacena y permite la búsqueda.
4. **Kibana**: Visualiza.

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

## Conclusiones.

- **Patricio Cuascota:** El uso de ELK es un enfoque integral que asegura que no solo se está guardando datos, sino que se está preparado para fallar. La monitorización no evita que el sistema caiga, pero reduce drásticamente el tiempo que pasas ciego intentando descubrir por qué cayó. Usar Beats directamente hacia Elasticsearch es excelente para proyectos pequeños o medianos. Sin embargo, en arquitecturas de alto tráfico, la combinación Beats / Logstash / Elasticsearch es el estándar efectivo porque Logstash actúa como un amortiguador (buffer) y un transformador pesado, dejando que los Beats sigan siendo ligeros y rápidos en los nodos de aplicación. Metricbeat nos dirá si es el servidor, APM nos dirá si es el código, y los logs nos dirán quién estaba ahí cuando ocurrió.

No todo puede ir a un dashboard. Si un dato no requiere una acción humana cuando falla, es una métrica de diagnóstico, no una alerta.

Y en el contexto de Apollo Server, la combinación de Filebeat (logs) + APM Node.js Agent (GraphQL) + Kibana (visualización) crea un circuito cerrado de observabilidad que permite pasar de algo falla a esta query específica tiene una latencia de 2 segundos de 100 peticiones de los casos de prueba (ver Imagen 28), lo cual es exactamente el valor diferencial que justifica la complejidad de infraestructura añadida por el stack.

- **Andrés Pilay:** Hemos validado que el uso de agentes ligeros (Beats) escritos en Go permite una recolección de datos granular sin comprometer el rendimiento del servidor Apollo, a diferencia de las arquitecturas monolíticas basadas solo en Logstash que consumen excesiva JVM. Esto nos permite correlacionar métricas de hardware con logs de aplicación para entender no solo qué pasó, sino por qué pasó. La implementación de Infrastructure as Code mediante Packer para la creación de AMIs y Terraform para el orquestamiento de red y cómputo, garantizó un entorno reproducible. La separación modular entre el servidor de aplicación Apollo/Node y el servidor de monitoreo Elastic/Kibana asegura que la carga de observabilidad no afecte la disponibilidad del servicio principal. La estrategia de monitoreo trasciende la simple visualización; al definir umbrales de alerta basados en la saturación y errores, reducimos el Tiempo Medio de Reparación. Además, la implementación de una política de retención 'Hot-Warm-Cold' asegura la

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

sostenibilidad económica y técnica del proyecto, evitando que el almacenamiento de logs sature el servidor Elastic.

- **Dante Gabriel Masache:** La implementación realizada demuestra que la integración de prácticas DevOps con herramientas de Infrastructure as Code y observabilidad permite construir entornos altamente reproducibles, escalables y medibles. La automatización mediante Packer y Terraform no solo redujo la intervención manual en el despliegue, sino que aseguró consistencia entre entornos, facilitando futuras ampliaciones o recuperaciones ante fallos. Asimismo, la incorporación de Elastic Stack junto con APM y Beats transformó el servidor Apollo en un sistema observable en tiempo real, permitiendo correlacionar infraestructura, aplicación y seguridad en un único ecosistema. Más allá de cumplir con los requisitos técnicos, el proyecto evidencia cómo la monitorización estratégica, acompañada de umbrales de alerta y políticas de retención bien definidas, convierte los datos en información accionable, reduciendo riesgos operativos y fortaleciendo la toma de decisiones basada en métricas.
- **Alberto Soriano:** Se ha optado el usar laC con el objetivo de mantener coherencia entre los distintos entornos y poder separar la creación de las imágenes con Packer del aprovisionamiento con Terraform, lo cual simplificó el proceso y redujo los posibles problemas de configuración del despliegue manual, ya que el stack ELK es bastante pesado y este quedó instalado desde la fase de creación del AMI y no durante la inicialización de cada instancia lo cual acelera el proceso de despliegue. El mayor desafío no estuvo en la automatización en sí, sino en la configuración de la seguridad y en el orden correcto de arranque de los servicios. La versión 8 de Elasticsearch incorpora por defecto mecanismos de seguridad estrictos como SSL/TLS y tokens que, aunque adecuados para producción, dificultan la comunicación directa entre los componentes en este entorno sencillo para pruebas. Para garantizar que el agente APM pudiera conectarse correctamente al servidor, fue necesario desactivar explícitamente estas capas de seguridad en la configuración de la imagen, permitiendo la comunicación interna dentro de la red privada. También surgieron complicaciones relacionadas con el inicio simultáneo de los servicios. No bastaba con que se ejecutaran, era necesario asegurar que una vez injectada la IP privada del servidor Elastic, servicios como Metricbeat y la aplicación Node.js se reiniciaran correctamente para establecer el flujo de

Asignatura	Actividad	Fecha
Herramientas DevOps	Actividad 3 Grupal: Monitorización de un despliegue de Apollo Server	13/02/2026

datos. Ajustar los scripts de configuración “user\_data” fue super importante para ajustar la correcta inicialización y comunicación de los servicios logrando una integración estable. Ahora queda por analizar cómo sería en un entorno productivo, este enfoque debería evolucionar hacia una arquitectura más segura y distribuida. Sería indispensable reactivar la seguridad, gestionar certificados y credenciales mediante un servicio especializado, como un gestor de secretos, en lugar de deshabilitar la seguridad o manejar datos sensibles en texto plano. Asimismo, resultaría más apropiado desacoplar Elasticsearch, Kibana y APM en instancias o contenedores independientes, favoreciendo la escalabilidad y la tolerancia a fallos frente al esquema monolítico utilizado en esta práctica.

## Valoración de equipo

Todo el equipo está de acuerdo con el siguiente cuadro:

Criterio	Si	No	A veces
Todos los miembros se han integrado al trabajo en grupo	X		
Todos los miembros participan activamente	X		
Todos los miembros respetan otras ideas aportadas	X		
Todos los miembros participan en la elaboración del informe	X		
Me he preocupado por realizar un trabajo cooperativo con mis compañeros.	X		
Señala si consideras que algún aspecto del trabajo en grupo no ha sido adecuado.	X		